INTERNATIONAL STANDARD

**ISO 10303-28**

First edition
2007-10-01

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

## Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas

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

*Partie 28: Méthodes d'implémentation: représentations XML de schémas et de données EXPRESS en utilisant des schémas XML*

Reference number
ISO 10303-28:2007(E)



ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

# Contents

Page

 

 

## Figures



## Tables



 

# 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.

International Standards are drafted in accordance with the rules given in the ISO/ IEC Directives, Part 2.

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.

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

ISO 10303-28 was prepared by Technical Committee ISO TC184, *Industrial automation systems and integration*, Subcommittee SC4 *Industrial data*.

ISO 10303-28 constitutes a technical revision of ISO/TS 10303-28:2003, which is provisionally retained in order to support continued use and maintenance of implementations based on it, and to satisfy the normative references of other parts of ISO 10303.

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

Each part of ISO 10303 is a member of one of the following series: description methods, implementation methods, conformance testing methodology and framework, integrated generic resources, integrated application resources, application protocols, abstract test suites, application interpreted constructs, and application modules. This part of ISO 10303 is a member of the implementation methods series.

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

`http://www.tc184-sc4.org/titles/STEP_Titles.htm`

# 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 is a member of the implementation methods series. This part of ISO 10303 specifies means by which schemas specified using the EXPRESS language (defined in ISO 10303-11) and data governed by EXPRESS schemas can be represented as an XML document. This enables product data described in EXPRESS to be exchanged using XML and the many software tools developed to support XML technologies. It also permits product data sets so described to be readily incorporated into "electronic commerce" transactions represented in XML.

Readers of this part of ISO 10303 should have knowledge of the EXPRESS language, the XML Schema language, XML, and XML-related standards in order to understand its technical content.

For the representation of data corresponding to an EXPRESS schema, this part of ISO 10303 formally specifies the structure of conforming exchange documents using the XML Schema language. Some elements of those documents represent data sets conforming to EXPRESS schemas, and this part of ISO 10303 specifies the structure of those elements using XML Schema type definitions and element declarations that are derived from the EXPRESS schema declarations. This part of ISO 10303 also specifies the rules for encoding conforming data in XML to match the derived XML schema. In order to accommodate a number of conflicting requirements for the use of conforming exchange documents, this part of ISO 10303 also defines certain configuration directives that can be used to specify alternative structures in the derived XML schema and alternative encoding rules.

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@tc184-sc4.org.

This part of ISO 10303 constitutes a technical revision of ISO/TS 10303-28:2003, which is provisionally retained to allow for the implementation of the Part 28 late binding. ISO/TS 10303-28:2003 was intended for trial use with emerging XML technologies, and although the fundamental capabilities remain the same, the underlying XML technologies have progressed, and this part of ISO 10303 uses technologies and features that were formerly unavailable or differently provided. This part of ISO 10303 therefore, is not "upwardly compatible" with ISO/TS 10303-28:2003. Neither a document nor a processor that conformed to ISO/TS 10303-28:2003 will conform to the specifications of this part of ISO 10303 without substantial modification.

The major technical changes from ISO/TS 10303-28:2003 are:

— This part of ISO 10303 specifies the structure of XML exchange documents using XML Schema, and specifies the mapping of the EXPRESS data model to an XML Schema data model. ISO/TS 10303-28:2003 specified the structure using Document Type Definitions (DTDs). It is expected that most future XML documents will be validated against an XML schema, instead of a DTD.

— The ISO/TS 10303-28:2003 mapping of EXPRESS schema text into a complex XML structure is no longer included in this part of ISO 10303; representation of an EXPRESS schema as a body of text is the only form retained.

— The ISO/TS 10303-28:2003 mapping of EXPRESS-modelled data sets designated the "late-binding" is not included in this part of ISO 10303.

 

— The ISO/TS 10303-28:2003 mapping of EXPRESS-modelled data sets designated the "ETEB" is not included as such. The major features of that binding, as they appear in the XML form of the data, are included in the "default mapping" in this part of ISO 10303. But this part of ISO 10303 makes use of an XML Schema feature that was not available in DTDs — context names — to simplify the names for the "accessor elements".

— The ISO/TS 10303-28:2003 mapping of EXPRESS-modelled data sets designated the "OSEB" is not included as such. The major features of that binding, as they appear in the XML form of the data, are included in the "attribute-content" mapping in this part of ISO 10303. But in this part of ISO 10303, there are several changes to the representation structures for aggregation data types and SELECT data types that provide for representation options and consistency across them.

— This part of ISO 10303 permits the use of XML Schema "simple list" structures to represent many values of aggregation data types, a feature that is not standardized in XML 1.0 and could not be specified in a DTD.

— Many separate options for re-configuring the EXPRESS schema, configuring the XML schema , and configuring the XML data are now supported by a configuration language, and support for that language is a mandatory feature of conforming processors.

Warning:

This part of ISO 10303 provides a specification intended to be implemented in software. Incompatibilities may result in machine-to-machine communication in the case of software developed on the basis of translations of this part of ISO 10303 into languages other than the official ISO languages. It is accordingly strongly recommended that any implementations be developed only on the basis of the texts in the official ISO languages.

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

# Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas

## 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. This part of ISO 10303 formally specifies the XML representation by specifying an overall XML schema for the exchange document and additional XML schemas that correspond to the EXPRESS schemas that govern the exchange data sets.

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

— specification of the form of XML documents containing EXPRESS schemas and data governed by EXPRESS schemas (see Clause 5);

— for an arbitrary EXPRESS schema, specification of an XML schema that corresponds to the EXPRESS schema and formally describes the XML representation of data governed by that schema (see Clause 6);

— specification of the representation of values of EXPRESS data types as XML element content and as XML attribute values (see Clause 9);

— specification of the set of configuration directives that may be used to specify options for the structure of the XML representation of data sets that conform to EXPRESS schemas (see Clause 10).

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

— specification of XML Schema declarations or definitions that depend on the semantic intent, as distinct from the EXPRESS language statements, of any particular EXPRESS schema;

— specification of mappings from the XML Schema language to the EXPRESS language;

— specification of the mapping to an EXPRESS schema from an XML schema that has been derived from an EXPRESS schema.

## 2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 8824-1:2002, *Information technology – 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 10303-11:2004, *Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual*

ISO 10303-22:1998, *Industrial automation systems and integration – Product data representation and exchange – Part 22: Implementation methods: Standard data access interface*

ISO 639-1:2002, *Codes for the representation of names of languages – Part 1: Alpha 2 code*

ISO 3166-1:2006, *Codes for the representation of names of countries and their subdivisions – Part 1: Country codes*

*Uniform Resource Identifiers (URI): Generic Syntax*. Internet Engineering Task Force RFC 2396 August 1998 [cited 2004-03-15]. Available from World Wide Web:
<http://www.ietf.org/rfc/rfc2396.txt>

*Extensible Markup Language (XML) 1.0*. World Wide Web Consortium Recommendation 4 February 2004 [cited 2004-03-15]. Available from World Wide Web:
<http://www.w3.org/TR/REC-xml>

*Namespaces in XML*. World Wide Web Consortium Recommendation 14 January 1999 [cited 2004-03-15]. Available from World Wide Web:
<www.w3.org/TR/REC-xml-names/>

*XML Schema Part 1: Structures*. World Wide Web Consortium Recommendation, 2 May 2001 [cited 2004-03-15]. Available from World Wide Web:
<http://www.w3.org/TR/xmlschema-1/>

*XML Schema Part 2: Datatypes*. World Wide Web Consortium Recommendation, 2 May 2001 [cited 2004-03-15]. Available from World Wide Web:
<http://www.w3.org/TR/xmlschema-2/>

*Xpointer Framework Version 1.0*. World Wide Web Consortium Recommendation 25 March 2003 [cited 2004-03-15]. Available from World Wide Web:
<http://www.w3.org/TR/xptr-framework/>

# 3 Terms, definitions, abbreviations, and conventions

## 3.1 Terms defined in ISO 10303-1

For the purposes of this document, the following terms defined in ISO 10303-1 apply.

— data;

— information.

## 3.2 Terms defined in ISO 10303-11

For the purposes of this document, the following terms defined in ISO 10303-11 apply.

**3.2.1**
**data type**
domain of values

  

NOTE Because two standards for specification of data types are used extensively in this part of ISO 10303, the term *data type* is always prefixed by "EXPRESS" or "XML Schema", to indicate the context in which each usage is to be understood.

**3.2.2**
**EXPRESS attribute**
property of an EXPRESS entity instance that is represented by a value of an EXPRESS data type and a name that indicates the role that value plays in characterizing the instance

**3.2.3**
**EXPRESS data type**
data type specified in the syntax of the EXPRESS language

**3.2.4**
**EXPRESS entity instance ; entity instance**
named unit of data that represents a unit of information within the domain defined by an entity data type

**3.2.5**
**EXPRESS language element**
concept in the EXPRESS language, and by extension, its syntactic representation

NOTE In general, the term "element" is used in this part of ISO 10303 to refer to the fundamental syntactic component of XML data structures.

**3.2.6**
**fundamental type**
EXPRESS data type used to determine the representation of values of a defined data type

NOTE Because a defined data type can be defined in terms of another defined data type, "fundamental type" is formally defined recursively as: "The fundamental type of a defined type is the fundamental type of the underlying type, and the fundamental type of a data type other than a defined type is the data type itself."

**3.2.7**
**generalized data type**
EXPRESS data type that is used to specify a generalization of certain other data types, and which can only be used in certain very specific contexts

**3.2.8**
**independent entity instance**
EXPRESS entity instance that appears in a schema instance and need not play a role in characterizing some other entity instance in the schema instance

NOTE A *dependent entity* instance is one that appears in a schema instance *because* it is (a component of) the value of an attribute of some other entity instance. An *independent entity* instance is one that does not have that rationale, although it may play such a role.

**3.2.9**
**schema instance**
set of EXPRESS entity instances, grouped for some purpose, that is governed by a single EXPRESS schema

**3.2.10**
**underlying type**
domain of values of a defined data type, as specified by the syntactic object `underlying_type` in the EXPRESS type declaration for the defined data type (cf. fundamental type)

NOTE Adapted from ISO 10303-11:2004.

## 3.3 Terms defined in the XML Standards

For the purposes of this document, the following terms and definitions apply. Terms defined in Recommendations of the World Wide Web Consortium and the Internet Engineering Task Force (IETF) 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.

**3.3.1**
**element; XML element**
bounded component of the logical structure of an XML document that has a type and that may have XML attributes and content [*XML 1.0 Recommendation*]

**3.3.2**
**infoset**
abstract data set providing a consistent set of definitions for use in specifications that need to refer to the information in an XML document [*XML Schema Part 1 – Structures Recommendation*]

**3.3.3**
**qualified name**
complete name of an XML element, attribute or data type, including the local name and a prefix that identifies the namespace in which the local name is defined/declared [*Namespaces in XML Recommendation*]

**3.3.4**
**Uniform Resource Identifier (URI)**
compact string of characters for identifying an abstract or physical resource [IETF RFC 2396: *Uniform Resource Identifiers (URI): Generic Syntax*]

**3.3.5**
**XML attribute**
name/value pair associated with an XML element [*XML 1.0 Recommendation*]

**3.3.6**
**XML document**
(text) data object that conforms to the XML requirements for being well-formed [*XML 1.0 Recommendation*]

**3.3.7**
**XML schema**
body of declarations and definitions in the XML Schema language that specify the structure of a class of XML documents [*XML Schema Part 1 -- Structures Recommendation*]

NOTE "schema" is the XML Schema term that denotes a body of declarations and definitions. But, to distinguish the XML Schema term "schema" from the EXPRESS term "schema", this part of ISO 10303 prefixes the XML Schema term with "XML" (see 3.5). Unfortunately, that means that the case of the the word "schema" is all that distinguishes the name of the XML Schema language from the "schema" concept. This part of ISO 10303, however, deals with two XML schemas in particular, and the text generally uses the specific terms "derived XML schema" and "Base XML Schema" to refer to them.

 

**3.3.8**
**XML Schema component**
concept in the XML Schema language, and by extension, its syntactic representation [*XML Schema Part 1 -- Structures Recommendation*]

NOTE In general, the term "component" is used in this part of ISO 10303 to refer to the values that are members of the collection of values constituting an aggregate value.

**3.3.9**
**XML Schema data type; XML data type**
data type specified in the syntax of the XML schema language [*XML Schema Part 2 -- Data types Recommendation*]

**3.3.10**
**XML Schema language; XML Schema**
formal language used to specify the structure of XML documents, specified in the *XML Schema Part 1 -- Structures Recommendation*

## 3.4 Other terms and definitions

For the purposes of this document, the following terms and definitions apply.

**3.4.1**
**accessor attribute**
XML attribute (of an XML element representing an entity instance) that represents an EXPRESS attribute and its value (cf. accessor element)

**3.4.2**
**accessor element**
XML element that represents an EXPRESS attribute and its value (cf. accessor attribute)

**3.4.3**
**aggregate value**
value of an EXPRESS aggregation data type, or a value of a defined data type whose fundamental type is an EXPRESS aggregation data type

**3.4.4**
**base-type**
for an EXPRESS aggregation data type, data type of the component values

**3.4.5**
**by-reference instance element**
instance element that has no XML content but includes a reference to a by-value instance element whose content is a representation of the value

**3.4.6**
**by-value instance element**
instance element whose content is the representation of a data value, or the complete representation of an EXPRESS entity instance

**3.4.7**
**characterized entity instance**
EXPRESS entity instance for which there is one EXPRESS entity data type in the context schema that is declared to have or inherit all of the properties that are present in the instance

**3.4.8**
**context schema; governing schema**
for a `uos` element, EXPRESS schema that contains the model of the data contained in that element

**3.4.9**
**deepest base-type**
for an aggregation data type that contains nested aggregation data types, base-type at some level of nesting whose values are used as the component values in the XML representation of the (possibly multi-dimensional) aggregate values

**3.4.10**
**deepest underlying type**
for an aggregation data type that contains nested aggregation data types, first encountered base-type at some level of nesting that is not an anonymous aggregation type, a defined data type whose underlying type is an aggregation type, or a SELECT type whose working select list contains a single defined data type whose underlying type is an aggregation type

**3.4.11**
**default mapping**
for a given EXPRESS Schema, XML schema that is derived as specified in this Part of ISO 10303, using the default values of all configuration directives and no additional configuration directives

**3.4.12**
**derived XML schema**
XML schema derived from the EXPRESS schema corresponding to the given EXPRESS schema and the given configuration file

**3.4.13**
**independent element**
instance element that appears as an immediate child of the `uos` element

**3.4.14**
**instance element**
XML element that represents a value of an EXPRESS data type

NOTE An instance element may be a by-value instance element, representing the value directly, or a by-reference instance element, which does not contain a representation of the value, but refers to a by-value element that does.

**3.4.15**
**list-of-values**
XML representation of an aggregate value that consists of a single character string in which the encodings of the component values appear in order, separated by SPACE characters (cf. sequence-of-elements)

**3.4.16**
**nested aggregation data type**
EXPRESS aggregation data type that appears as the base-type of another aggregation data type

**3.4.17**
**sequence-of-elements**
XML representation of an aggregate value that consists of a sequence of XML elements in which the content of each element is the encoding of a single component value (cf. list-of-values)

 

**3.4.18**
**uncharacterized entity instance**
entity instance for which no single entity data type in the context schema has all the properties needed to represent the entity instance

**3.4.19**
**unit of serialization; uos**
XML representation of a schema instance — a collection of EXPRESS entity instances, together with all of their attribute values, that are described by a single exchange schema

## 3.5 Conflicting terminology

EXPRESS and XML (Schema) use similar or identical words for related concepts. In this part of ISO 10303, the qualifiers "XML" and "EXPRESS" are used to distinguish between the different contexts. These qualifiers are used with the following terms:

— attribute;

— data type;

— element;

— entity;

— schema.

Exception: The term "element" always refers to an XML "element", except when it is preceded by the qualifier "EXPRESS". In many cases, the word "element" is part of a term denoting a specific type of XML element that is defined in this part of ISO 10303.

NOTE The XML "entity" component is not used in this edition of this part of ISO 10303. The term "entity" always refers to an EXPRESS entity data type or entity instance, and the qualifier "EXPRESS" always appears on the first use in a paragraph.

## 3.6 Abbreviations

For the purposes of this document, the following abbreviations apply.

| | |
|---|---|
| IANA | Internet Assigned Names Authority |
| IETF | Internet Engineering Task Force |
| MIME | Multi-purpose Internet Mail Extension |
| URI | Uniform Resource Identifier |
| URN | Uniform Resource Name |
| XML | Extensible Markup Language |

## 3.7 Conventions

The following conventions are used in this document.

### 3.7.1 Text conventions

— EXPRESS language keywords are represented in SMALL CAPITALS;

— Verbatim XML Schema is represented by text in a Courier font;

— XML Schema parameters are represented by ***bold italic Courier*** text;

— Configuration directives are represented by **bold** text;

— Terms are highlighted by *italic* font where they are introduced and defined in the text.

When the ellipsis character "..." appears in normative text, it represents all the other members of the list.

When the ellipsis character "..." appears in an example, it indicates that the (EXPRESS or XML) schema component is incomplete.

### 3.7.2 Namespace conventions

For the sake of clarity, the namespace prefixes in Table 1 are used throughout this part of ISO 10303 to refer to the namespaces identified by the corresponding URIs in Table 1.

NOTE There is no requirement for conforming documents to use these prefixes for these namespaces, although the use of `xs:` and `xsi:` for the XML standard namespaces is recommended.

**Table 1 — Namespace prefixes**

| Prefix | Associated namespace URI |
|---|---|
| `xs:` | `http://www.w3.org/2001/XMLSchema` |
| `xsi:` | `http://www.w3.org/2001/XMLSchema-instance` |
| `exp:` | `urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common` (as defined in Annex C) |
| `cnf:` | `urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:configuration_language` (as defined in Annex B) |
| `doc:` | `urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:document` (as defined in Annex D) |
| `Tns:` | the target namespace associated with the EXPRESS context schema (see 7.7 or 8.7) |

# 4 Conformance

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

— an XML instance document;

 

— an XML schema derived from an EXPRESS schema;

— a configuration file;

— a pre-processor that produces XML instance documents;

— a post-processor that accepts XML instance documents;

— a processor that produces an XML schema from an EXPRESS schema.

## 4.1 Conformance of an XML document

An *XML instance document* is an XML document that conveys EXPRESS-defined data. There are three classes of XML instance documents that can conform to this part of ISO 10303:

— iso-10303-28 documents, as specified in 4.1.1,

— uos documents, as specified in 4.1.2, and

— configured documents, as specified in 4.1.2.

For the purposes of this subclause, the XML information set (infoset) for a document shall be considered *valid* if the validity property for each element and for each attribute in the document's post-schema-validation infoset is "valid", as defined in *XML schema Part 1: Structures*.

### 4.1.1 Conformance of an iso-10303-28 document

An XML document shall *conform as an iso-10303-28 document* if it satisfies all of the following conditions:

— The root element for the document is an `iso_10303_28` element, as specified in Clause 5.

— The XML infoset for the `iso_10303_28` element is valid with respect to the Document Schema defined in Annex D.

— Each data set encoded in the document, if any, conforms as specified in ISO 10303-11 to a given EXPRESS schema, designated the *governing schema* for that data set.

— Each data set is encoded using a substitute unit of serialization element for the `uos` element specified in the Base XML Schema (see Annex C).

— For each substitute unit of serialization element, the XML infoset is valid with respect to the XML schema that is derived as specified in Clause 6, using the EXPRESS schema and the configuration file specified by that unit of serialization element.

NOTE 1 The governing EXPRESS schema can be different for each data set.

NOTE 2 The substitute `uos` element corresponding to a given EXPRESS schema is described in 7.8 or 8.8.

NOTE 3 Each unit of serialization element specifies the governing EXPRESS schema and the configuration file to which it corresponds. See 5.6 and 9.2.

NOTE 4 There is no requirement for the data set to be a "valid population", as defined in ISO 10303-11. The relationship between encoded data sets and populations is specified in a given `iso_10303_28` element by `schema_population` elements. See 5.3.

### 4.1.2 Conformance of a uos document

An XML document shall *conform as a uos document* if it satisfies all of the following conditions:

— The root element for the document is a `uos` element.

— The data set encoded in the `uos` element conforms as specified in ISO 10303-11 to a given EXPRESS schema, designated the *governing schema* for that data set.

— The XML infoset for the `uos` element is valid with respect to the XML schema that is derived from the governing EXPRESS schema as specified in Clause 6.

NOTE In a uos document, the `uos` element is the one that corresponds to the default mapping of the governing schema, that is, the one corresponding to an empty configuration file.

### 4.1.3 Conformance of a configured document

Properly, the conformance of a configured document is defined by the specification that defines the configured XML schema. This subclause is intended solely for reference by such specifications. Conformance of a configured document in the absence of such a referencing specification is not defined by this Part of ISO 10303.

An XML document shall *conform as a configured document* for a given EXPRESS schema and a given configuration file if it satisfies all of the following conditions:

— The root element for the document is the unit of serialization element derived as described in 8.8 from the given EXPRESS schema using the given configuration file.

— The data set encoded in the unit of serialization element conforms to the given EXPRESS schema as specified in ISO 10303-11.

— The XML infoset for the unit of serialization element is valid with respect to the XML schema that is derived from the governing EXPRESS schema as specified in Clause 6 using the given configuration file.

NOTE For specifications on the creation of an XML instance document conforming to this part of ISO 10303, see Clause 9.

## 4.2 Conformance of a derived XML schema document

An XML document shall *conform as a derived XML schema document* for a given EXPRESS schema and a given configuration file if it satisfies all of the following conditions:

— The given EXPRESS schema conforms to ISO 10303-11.

— The given configuration file, if any, conforms as a configuration file for the given EXPRESS schema as specified in 4.3.

— The content of the document satisfies the schema validity constraints described in Clause 5 of *XML Schema: Part 1: Structures*.

— The XML schema definition contains exactly the definitions and declarations that are specified in Clause 7 or Clause 8, as indicated by Clause 6, to be derived from the given EXPRESS schema using the given configuration file. The XML schema definition may contain additional XML Schema `annotation` elements.

 

There are two conformance classes for derived XML schema documents:

— The XML schema document *conforms in class 1* if it includes all definitions and declarations that are specified in Clause 7 or Clause 8, as indicated by Clause 6, *except* those specified to be included only in conformance class 2. A derived XML schema that conforms in class 1 shall not include any definitions or declarations specified in Clause 8 to be included only in conformance class 2 (see 10.2.23).

— The XML schema document *conforms in class 2* if it includes all of the definitions and declarations that are specified in Clause 8, *including* all of those specified to be included only in conformance class 2.

NOTE In order for a document to conform in class 2, Clause 6 must specify that the derived XML Schema will be the Configured XML Schema binding, specified in Clause 8, and the configuration directive **generate-keys= "true"** must apply.

## 4.3 Conformance of a configuration file

An XML document shall *conform as a configuration file* for a given EXPRESS schema if it satisfies all of the following conditions:

— The root element of the document is a `configuration` element.

— The `configuration` element is valid with respect to the Configuration Schema specified in Annex B.

— The configuration file conforms to the specifications given in Clause 10, where the context schema is the given EXPRESS schema.

NOTE If the configuration file contains no elements that are stated in Clause 10 to be schema-specific, the configuration file will conform for an arbitrary EXPRESS schema. But for the purposes of this part of ISO 10303, it is only useful to define conformance in the context of a given EXPRESS schema.

## 4.4 Conformance of a pre-processor

A software application is said to conform as a *pre-processor* if it produces XML instance documents that conform to this part of ISO 10303 as specified in 4.1. An application may conform as a pre-processor for iso-10303-28 documents only, or for uos documents only, or for both. 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.

There are three conformance classes for pre-processors:

— A pre-processor *conforms in class 1* if it can produce uos documents conforming to the class 1 derived XML schema for some EXPRESS schemas.

— A pre-processor *conforms in class 2* if it can produce iso-10303-28 documents that contain `uos` elements, and independent uos documents, conforming to the class 1 derived XML schema for some EXPRESS schemas.

— A pre-processor *conforms in class 3* if it can produce iso-10303-28 documents that contain substitute `uos` elements conforming to arbitrary EXPRESS schemas with arbitrary configuration files.

Conformance of a pre-processor in any case shall be claimed based on the EXPRESS schemas to which its XML document production is limited.

NOTE Both class 1 and class 2 define pre-processor conformance as the ability to produce XML data sets corresponding to the default XML schemas. Class 3 requires the pre-processor to support arbitrary configuration files. If an application conforms only for configured documents corresponding to specific configured XML schemas, it is not said to conform as a pre-processor to this part of ISO 10303. Rather, it conforms as a pre-processor to the specifications that define the configured XML schemas.

### 4.5 Conformance of a post-processor

A software application is said to conform as a *post-processor* if it can accept and process XML instance documents that conform to this part of ISO 10303 as specified in 4.1. An application may conform as a post-processor for iso-10303-28 documents only, or for uos documents only, or for both. 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.

There are three conformance classes for post-processors:

— A post-processor *conforms in class 1* if it can accept and process uos documents and/or iso-10303-28 documents that contain `uos` elements conforming to the class 1 derived XML schema for an arbitrary EXPRESS schema.

— A post-processor *conforms in class 2* if it can accept and process uos documents and/or iso-10303-28 documents that contain `uos` elements conforming to the class 2 derived XML schema for an arbitrary EXPRESS schema, and perform the corresponding XML validation.

— A post-processor *conforms in class 3* if it can accept and process iso-10303-28 documents that contain unit of serialization elements conforming to EXPRESS schemas with an arbitrary conforming configuration file.

In each of conformance classes 1 and 2, conformance of a post-processor shall be claimed based on the types of documents and the EXPRESS schemas to which its XML document processing is limited. In conformance class 3, conformance of a post-processor shall be claimed based on the EXPRESS schemas to which its XML document processing is limited.

NOTE If an application conforms only for configured documents corresponding to specific configured XML schemas, it is not said to conform as a post-processor to this part of ISO 10303. Rather, it conforms as a post-processor to the specifications that define the configured XML schemas.

### 4.6 Conformance of an XML schema generator

A software application *conforms as an XML schema generator* if it can process any valid EXPRESS schema and produce a conforming XML schema document as defined in 4.2.

An XML schema generator *conforms in class 1* if it can generate the default XML schema specified in Clause 7, that is, the derived schema that results from an empty configuration file.

An XML schema generator *conforms in class 2* if it can generate the derived configured XML schema corresponding to any EXPRESS schema and any conforming configuration file, as specified in Clause 8.

## 5 Document level elements

This clause specifies the XML elements that define the structure of an XML instance document conforming to this Part of ISO 10303. 5.1 defines the structure of an XML document that conforms as an iso-10303-28 document. 5.2 through 5.6 define the the content elements of the `iso-10303-28`

 

document root element. 5.7 defines the structure of an XML document that conforms as a uos document. 5.8 defines the structure of an XML document that conforms as a configured document. 5.9 defines an XML attribute contained in the the `iso-10303-28` document root element.

This Part of ISO 10303 specifies in Annex C a collection of XML Schema declarations and definitions, called the Base XML Schema, that shall be part of the base schema for every conforming XML instance document. The XML element names, attribute names and data type names declared in that collection shall constitute the namespace designated `urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common`. This Part of ISO 10303 also specifies in Annex D a collection of XML Schema declarations and definitions, called the Document Schema, that define the XML infoset for a conforming `iso-10303-28` document. The XML element names, attribute names and data type names declared in that collection shall constitute the namespace designated `urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:document`.

## 5.1 The iso-10303-28 document

The root element of an iso-10303-28 document shall be an `iso_10303_28` element. The `iso_10303_28` element contains all other elements defined in this part of ISO 10303. These elements permit the document to contain descriptive information, EXPRESS schemas, XML schema configuration files, and data sets described by EXPRESS schemas. These declarations are so specified that a conforming document can contain EXPRESS schemas, configuration files, and data sets from other XML documents by reference.

This Part of ISO 10303 specifies in Annex D a collection of XML Schema declarations and definitions, called the Document Schema, that shall be part of the document schema for every XML instance document that conforms as an iso-10303-28 document. The XML element names, attribute names and data type names declared in that collection shall constitute the namespace specified in Annex A.3.

The element type declaration in the Document Schema is:

```
<xs:element name="iso_10303_28">
 <xs:complexType>
    <xs:sequence>
       <xs:element ref="exp:header" minOccurs="0"/>
       <xs:choice minOccurs="0"  maxOccurs="unbounded">
          <xs:element ref="doc:schema_population"/>
          <xs:element ref="exp:uos"/>
         <xs:element ref="doc:express"/>
         <xs:element ref="doc:schema"/>
         <xs:element ref="cnf:configuration"/>
       </xs:choice>
    </xs:sequence>

       <xs:attribute name="version" type="xs:string" use="required"/>
       <xs:attribute name="edo"  type="xs:anyURI" use="optional"/>

 </xs:complexType>
</xs:element>
```

The `version` XML attribute shall identify the edition of this part of ISO 10303 to which the XML document conforms. For documents conforming to this edition of this part of ISO 10303, the value of `version` shall be '`2.0`'.

If present, the edo attribute should contain the fixed global identifier for the document as an enterprise data object (see 5.9).

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

## 5.2 Document and uos header information

This subclause specifies elements to contain "provenance" information that describes an element that is an information resource. The resource may be the entire iso-10303-28 document element or a unit of serialization (see 5.6) that it contains. "Provenance" information is information about the source of the information contained in the resource.

### 5.2.1 The exp:header element

The exp:header element shall contain administrative information that characterizes the content of the entire XML resource – the element that contains the exp:header element. The resource that contains the header element may be an iso_10303_28 element (representing the entire document) or a uos element instance document. The element type declaration in the Document Schema is:

```
<xs:complexType name="name_and_address">
  <xs:sequence>
    <xs:element name="name" type="xs:string" />
    <xs:element name="address">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="address_line" type="xs:string"
              minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:element name = "header">
  <xs:complexType>
    <xs:sequence>
     <xs:element name="name" type="xs:string" minOccurs="0"/>
     <xs:element name="time_stamp" type="xs:dateTime" minOccurs="0"/>
     <xs:element name="author" type="exp:name_and_address"
     minOccurs="0"/>
     <xs:element name="organization" type="exp:name_and_address"
     minOccurs="0"/>
     <xs:element name="preprocessor_version" type="xs:string"
     minOccurs="0"/>
     <xs:element name="originating_system" type="xs:string"
     minOccurs="0"/>
     <xs:element name="authorization" type="xs:string" minOccurs="0"/>
     <xs:element name="documentation" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

NOTE This definition corresponds to the FILE_NAME entity defined in ISO10303-21 [1].

 

### 5.2.2 The name element

The `name` element shall provide a human readable identifier for the XML resource.

### 5.2.3 The time_stamp element

The `time_stamp` element shall specify the date and time when the XML resource was created.

### 5.2.4 The author element

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

### 5.2.5 The organization element

The `organization` element shall identify the organization that created, or is responsible for, the XML resource.

### 5.2.6 The authorization element

The `authorization` element shall specify the release authorization for the XML resource and the signatory, where appropriate.

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

### 5.2.7 The originating_system element

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

### 5.2.8 The preprocessor_version element

The `preprocessor_version` element shall identify the software system that created the XML resource itself, including platform and version identifiers.

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

## 5.3 The `schema_population` element

The `schema_population` element identifies a collection of EXPRESS entity instances that are represented in unit of serialization elements within the document as constituting a "population", which can be validated under the rules of an EXPRESS schema, as defined in ISO 10303-11. Given the specification of a population via a `schema_population` element, a conforming post-processor may, but is not required to, validate the population against the governing schema.

The element type declaration in the Document Schema is:

```
<xs:element name="schema_population">
 <xs:complexType>
  <xs:attribute name="governing_schema" type="xs:IDREF" use="required"/>
  <xs:attribute name="governed_sections" type="xs:IDREFS"
    use="optional"/>
  <xs:attribute name="determination_method"
    type="xs:normalizedString" default="section_boundary"/>
```

```
    </xs:complexType>
  </xs:element>
```

The collection of entity instances constituting the population shall be determined by applying an algorithm identified by the XML attribute `determination_method` to the set of unit of serialization elements identified by the XML attribute `governed_sections`.

The value of the `governing_schema` XML attribute shall be the local XML identifier for the `express` element that represents the EXPRESS schema to be used to validate the population described by the `schema_population` element. The corresponding `express` element shall appear in the document.

NOTE 1 The referenced `express` element may contain a representation of the EXPRESS schema or an external reference for the EXPRESS schema.

The value of the `governed_sections` XML attribute shall be a sequence of local XML identifiers separated by spaces, each of which shall identify a unit of serialization element appearing in the document. If no value for the `governed_sections` attribute is provided, the specified list of unit of serialization elements shall comprise all unit of serialization elements appearing in the document. The specified unit of serialization elements shall be the input operands of the determination method.

NOTE 2 Possible relationships between the context schema for a unit of serialization element and the `governing_schema` for the population(s) to which its entity instances belong are discussed in Annex E.

The value of the `determination_method` attribute shall be a string of characters that identifies the algorithm to be used for selecting EXPRESS entity instances to comprise the population. Three possible values for this attribute and the corresponding selection algorithms are specified in Annex E. Other values may appear, and their interpretation is not specified in this Part of ISO 10303. If no value for this attribute is provided, the value `"section_boundary"` is assumed and the section boundary method described in Annex E shall be used.

A conforming `iso_10303_28` element may contain zero or more `schema_population` elements. Each `schema_population` element describes one complete population for purposes of validation as specified in ISO 10303-11. If no `schema_population` elements appear, the relationship of the EXPRESS entity instances represented in the document, if any, to any "valid population" is not specified, and validation of the contents of the `iso_10303_28` element against any EXPRESS schema is not possible without additional information.

A given unit of serialization element may be used explicitly, by a reference to the element in the `governed_sections` XML attribute of the `schema_population` element, in zero, one, or many populations. A given unit of serialization element may also be used implicitly in zero, one, or many populations, according to the determination method used for those populations, but such a use does not necessarily imply the use of all EXPRESS entity instances represented in that unit of serialization. If a unit of serialization element is not explicitly or implicitly referenced by any `schema_population` element, the relationship of the EXPRESS entity instances represented therein to any "valid population" is not specified.

NOTE 3 The following procedure describes an approach for checking the validity of a population that is consistent with the above:

For each `schema_population` element in the `iso_10303_28` element:

— find a set of EXPRESS entity instances by applying the determination method identified by the value of the `determination_method` attribute to the unit of serialization elements identified in the value of the

 

governed_sections attribute, or to all unit of serialization elements in the iso_10303_28 element if governed_sections is not specified;

— check this set of instances against the rules and constraints defined by the EXPRESS schema designated by the value of the governing_schema attribute.

## 5.4 The express element

The express element shall contain one EXPRESS schema, either "by-value" or "by-reference".

The element type declaration in the Document Schema is:

```
<xs:element name="express" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="schemaLocation" type="exp:Seq-anyURI"
          use="optional"/>
        <xs:attribute name="id" type="xs:ID" use="required"/>
        <xs:attribute name="schema_identifier"
          type="xs:normalizedString"  use="optional"/>
        <xs:attribute name="schema_name" type="xs:normalizedString"
          use="optional"/>
        <xs:attribute name="schema_version" type="xs:normalizedString"
          use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

The Seq-anyURI type declaration in the Base XML Schema is:

```
<xs:simpleType name="Seq-anyURI">
  <xs:list itemType="xs:anyURI"/>
</xs:simpleType>
```

In either case:

— The value of the id XML attribute shall be a local XML identifier for the EXPRESS schema. This value provides the reference to the EXPRESS schema for schema_population elements and unit of serialization elements within the document.

— The schema_name XML attribute, if present, shall contain the EXPRESS identifier for the schema, converted as specified in 7.1.2 or 8.1.2, as specified by Clause 6.

— The schema_version XML attribute, if present, shall contain the version identifier for the schema.

— For an EXPRESS schema that is defined by a Part of ISO 10303, the schema_identifier attribute, if present, shall contain the ASN.1 identifier value associated with that schema by that Part of ISO 10303. In any other case, the use of the schema_identifier attribute is not specified by this Part of ISO 10303.

### 5.4.1 By-reference representation of an EXPRESS schema

When the EXPRESS schema is represented "by-reference", the `schemaLocation` XML attribute shall be present and its value shall identify one or more resources that contain the EXPRESS schema (text).

NOTE By convention, the value of `schemaLocation` is either a single URI, or a list of URIs of which the first is a URN that identifies the configuration file globally, and all of the others are possible locations of the text.

When the EXPRESS schema is represented "by-reference", the content of the `express` element shall be nil (empty).

### 5.4.2 By-value representation of an EXPRESS schema

When the EXPRESS schema is represented "by-value", the content of the `express` element shall be a text representation of the EXPRESS schema conforming to 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 the *XML 1.0 Recommendation,* 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.

When the EXPRESS schema is represented "by-value", the `schemaLocation` XML attribute shall not be present.

EXAMPLE The EXPRESS schema:

```
SCHEMA hello_world;
END_SCHEMA;
```

would appear as:

```
<express_schema id="s_01"><![CDATA[SCHEMA hello_world; END_SCHEMA;]]>
</express_schema>
```

as an XML CDATA section.

## 5.5 The `configuration` element

The `configuration` element contains a configuration file that controls the mapping of a given EXPRESS schema to an XML schema, and the XML representation of the data that is described by that EXPRESS schema. The structure and interpretation of the `configuration` element is specified in 10.1.

## 5.6 The unit of serialization element

A *schema instance* is a data set, grouped for some purpose, that is modeled by a single EXPRESS schema. The *unit of serialization element* is the XML element that represents one schema instance. The EXPRESS schema that models that schema instance is designated the *context schema* for that unit of serialization element. For each context schema, there is a specific unit of serialization element that

 

is declared in the namespace associated with the derived XML schema, as described in 7.8 or 8.8. Each unit of serialization element shall be a member of the substitution group for the `exp:uos` element, and its data type shall be an extension of the `exp:uos` data type, defined in the Base XML Schema as follows:

```
<xs:complexType name="uos">
 <xs:sequence>
    <xs:element ref="exp:header" minOccurs="0"/>
 </xs:sequence>

 <xs:attribute name="id" type="xs:ID" use="optional"/>
 <xs:attribute name="express" type="exp:Seq-anyURI"
    use="optional"/>
 <xs:attribute name="configuration" type="exp:Seq-anyURI"
    use="optional"/>
 <xs:attribute name="schemaLocation"  type="exp:Seq-anyURI"
    use="optional"/>
 <xs:attribute name="edo"  type="xs:anyURI" use="optional"/>
 <xs:attribute name="defaultLanguage" type="xs:language"
     use="optional"/>
</xs:complexType>

<xs:element name="uos" type="exp:uos" abstract="true" />
```

The `exp:header` element, if present, shall be as specified in 5.2.1.

When the unit of serialization is contained in an ISO 10303-28 document, the `id` XML attribute shall be present. The value of the `id` XML attribute shall be a local XML identifier for the "unit of serialization". This value provides the reference to the unit of serialization element for `schema_population` elements and serves as part of the unique identifier for the unit of serialization element when it is referenced as a resource from outside the ISO 10303-28 document. When the `uos` element is the root of the document (see 5.7), the `id` XML attribute should not be present.

The value of the `express` XML attribute shall identify one or more resources that contain the context schema for the schema instance encoded in the unit of serialization element. If this attribute is not present, the EXPRESS schema associated with the configuration file (see 10.1) is presumed to be the context schema.

The value of the `configuration` XML attribute shall identify one or more resources that contain the configuration file that was used to control the encoding of the EXPRESS schema instance in this unit of serialization element. If this attribute is not present, the default configuration for the EXPRESS schema specified by the `express` attribute is assumed.

The value of the `schemaLocation` XML attribute shall identify one or more resources that contain the governing XML schema for the unit of serialization element. If this attribute is not present, the governing XML schema shall be that derived from the specified EXPRESS schema using the specified configuration file.

For any of the `express`, `configuration`, and `schemaLocation` attributes, the URI may have the form: ***uri*#*id***, where ***uri*** is a URI conforming to IETF RFC 2396 that locates the resource, and the fragment identifier ***id*** is an XML identifier local to that resource that identifies the schema element.

NOTE 1 When the URI has the form #***id*** (with no URL), it refers to an XML element with that id in the current document. For example, it may refer to a `express` or `configuration` XML element in the current document.

NOTE 2 The use of the `configuration` and `express` attributes are described in more detail in 9.2.

If present, the `edo` attribute should contain the fixed global identifier for the unit of serialization as an enterprise data object (see 5.9).

The value of the `defaultLanguage` XML attribute shall identify the language used as default in the data set. Values of this XML attribute are defined by ISO-639, "Codes for the Representation of Names of Languages". Subcategories of these languages are identified by ISO-3166, "Codes for the Representation of Names of Countries".

EXAMPLE "fr-CA" for Canadian-french. If this attribute is specified, any string in the data set can be interpreted as belonging to the identified language. If this attribute is not present no language default can be assumed.

NOTE 3 A default language may be overwritten on the basis of an EXPRESS attribute, if the underlying EXPRESS schema and the configuration applied define so.

### 5.7 The uos document

A uos document represents a single schema instance, and a single unit of serialization, as defined in 5.6. The root element of a uos document shall be a schema-specific `uos` element conforming to the default XML schema for the EXPRESS schema that governs the schema instance, as specified in 7.8.

### 5.8 The configured document

A configured document represents a single schema instance, and a single unit of serialization, as defined in 5.6. The root element of a configured document shall be the unit of serialization element declared in the XML schema derived from the EXPRESS schema that governs the schema instance, according to the given configuration file, as specified in 8.8.

### 5.9 Enterprise data objects

An *enterprise data object* is a data resource that may be referred to in business transactions and other exchange documents. An enterprise data object has a fixed global identifier by which it is known. For consistency with XML practice, we assume that the fixed global identifier is a "reference URI" that is independent of location.

An enterprise data object can be an ISO 10303-28 document, a unit of serialization, or a single EXPRESS entity instance. In each case, the XML `edo` attribute is attached to the corresponding XML element to capture this property.

## 6 Derived XML Schema

The XML schema derived from the EXPRESS schema is said to be the *derived XML schema* corresponding to the given EXPRESS schema and an optional given configuration file.

The XML schema derived from the EXPRESS schema is said to be the *default XML schema* corresponding to the given EXPRESS schema if any of the following is true:

— no configuration file is given;

— the given configuration file contains no directives;

— all directives in the given configuration file specify only the default values.

 

When any of the above apply, the XML schema derived from the EXPRESS schema shall be the default XML schema binding, specified in Clause 7.

The XML schema derived from the EXPRESS schema is said to be the *configured XML schema* in all other cases. If the optional configuration file is included and it contains at least one directive that does not specify the default value for the given EXPRESS schema then the configured XML binding shall apply as specified in Clause 8.

The following subclauses specify the fundamental concepts and assumptions of the derived XML schema bindings.

## 6.1 Preconditions

The creation of XML Schema declarations and definitions from an EXPRESS schema assumes that the schema meets the following precondition:

— syntactically correct EXPRESS schema;

— the context schema is the complete schema for some collection of data sets to be exchanged.

## 6.2 Unmapped EXPRESS concepts

The following EXPRESS features are not mapped:

— `RULE` declarations;

— `FUNCTION` declarations and `PROCEDURE` declarations;

— `CONSTANT` declarations;

— local (`WHERE`) rules within `ENTITY` declarations;

— local (`WHERE`) rules within `TYPE` declarations;

— `USE` and `REFERENCE` statements (see 6.1);

— subtype constraints (`SUPERTYPE` clauses and `SUBTYPE_CONSTRAINT`s).

NOTE The subtype constraint itself is not mapped; however, if the subtype constraint declares an entity `ABSTRACT`, it affects the mapping to XMLSchema (see 7.5.3 or 8.5.3).

## 6.3 Abstract entity data types

An EXPRESS entity data type is *abstract* if every instance of that entity data type must be an instance of some subtype of it.

An EXPRESS entity datatype can be declared abstract in one of three ways:

— as an `ABSTRACT ENTITY` in the entity declaration,

— as an `ABSTRACT SUPERTYPE` in a supertype-expression in the entity declaration, or

— as `ABSTRACT` in a `SUBTYPE_CONSTRAINT`.

# 7 Default XML Schema Binding

This clause specifies a mapping from an EXPRESS schema to the default corresponding XML Schema declarations, definitions and representation. When the derived XML schema specified in Clause 6 is said to be the *default XML schema,* this clause shall apply.

## 7.1 Naming conventions

This clause specifies uniform conventions for creating XML names based on EXPRESS identifiers.

### 7.1.1 Schema

Every EXPRESS schema shall have an associated XML namespace URI. This URI shall designate the target namespace for the XML element and data type names that are schema-specific elements of the binding of that schema, as specified in 7.7.

### 7.1.2 EXPRESS identifiers

EXPRESS language identifiers are not case-sensitive. XML names are case-sensitive.

Every EXPRESS identifier that appears in the XML schema definitions and declarations shall be mapped to an XML name, or the local part of an XML name. The XML name shall be the same as the EXPRESS identifier except that the initial letter is upper case and all other letters are lower case.

Due to the XML naming restrictions specified in the XML Recommendation, any EXPRESS identifier 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, and attribute names. Exceptions for certain situations are specified in the subclauses dealing with particular kinds of EXPRESS identifiers.

### 7.1.3 Data types

Each EXPRESS data type identifier shall be mapped to a qualified name as defined in the Namespaces in XML Recommendation.

The local part of the qualified name shall be the EXPRESS data type identifier mapped as specified in 7.1.2. For every EXPRESS named data type declared in, or explicitly interfaced into, the context schema, the associated XML Schema namespace for the qualified name shall be that of the context schema.

For every data type that is implicitly interfaced into the context schema, the local part of the qualified name shall be derived as specified in 7.1.2 from the EXPRESS identifier for that data type, as it appears in the schema in which the data type is declared. The associated namespace shall be that of the schema in which the data type is declared.

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

## 7.2 XML Schema data types corresponding to EXPRESS data types

For every EXPRESS data type appearing in the context schema, including named data types, built-in data types and anonymous constructed data types, there is a corresponding XML schema data type. This subclause specifies that correspondence.

NOTE This subclause does not specify any definitions or declarations to appear in the derived XML schema. It specifies XML schema components that may appear in definitions and declarations required by other subclauses.

### 7.2.1 EXPRESS simple data types

This subclause specifies the XML schema datatypes that correspond to the EXPRESS built-in simple data types.

#### 7.2.1.1 BINARY data type

The EXPRESS `BINARY` data type is mapped to the `exp:hexBinary` data type. Exception: If the EXPRESS `BINARY` data type specifies a maximum or `FIXED` length, it shall be mapped as specified in 7.2.1.1.1

The `exp:hexBinary` data type is defined in the Base XML Schema (see Annex C) as follows:

```
<xs:complexType name="hexBinary" >
 <xs:simpleContent>
    <xs:extension base="xs:hexBinary">
       <xs:attribute name="extraBits" type="xs:integer"
          use="optional"/>
    </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

EXAMPLE

```
ENTITY entity_with_binary_attribute;
  a_binary   : BINARY;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Entity_with_binary_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
        <xs:element type="exp:hexBinary" name="A_binary"/>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML instance document:

```
<Entity_with_binary_attribute>
  <A_binary>0FB7</A_binary>
</Entity_with_binary_attribute>
```

##### 7.2.1.1.1 Constrained `BINARY` data types

An EXPRESS `BINARY` data type for which a maximum or `FIXED` length is specified is said to be a *constrained* `BINARY` data type. It shall be mapped to a `complexType` that is a restriction of the `exp:hexBinary` data type.

When the constrained `BINARY` data type is the underlying type in an EXPRESS type declaration, the corresponding XML data type shall be as specified in 7.3.1.1.

For a constrained BINARY data type, the derived XML schema shall contain a type definition for the XML data type that corresponds to the constrained BINARY data type. The type definition shall have the form:

```
<xs:complexType name="Binary.minBits.bits">
  <xs:simpleContent>
   <xs:restriction base="exp:hexBinary">
     <xs:minLength value="minLength"/>
     <xs:maxLength value="maxLength"/>
     <xs:attribute name="extraBits" type="xs:integer"
         fixed="remainder"/>
   </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

where:

— ***minBits*** shall be the fixed length (in bits) specified for the EXPRESS BINARY data type, if it is declared FIXED. Otherwise, ***bits*** shall be "0".

— ***bits*** shall be the maximum or fixed length (in bits) specified for the EXPRESS BINARY data type.

— ***minLength*** shall be the octet value (see below) of the fixed length specified for the EXPRESS BINARY data type, if it is declared FIXED. Otherwise, the xs:minLength facet may be omitted.

— ***maxLength*** shall be the octet value of the maximum (or fixed) length specified for the EXPRESS BINARY data type.

— ***remainder*** shall be a decimal integer giving the value (***bits*** modulo 8), where ***bits*** is as specified below, if the EXPRESS data type is declared FIXED. Otherwise, the redeclaration of the extraBits attribute shall be omitted.

Given ***bits*** = the maximum (or fixed) length in bits specified for the EXPRESS data type, the *octet value* shall be calculated as follows:

— If *bits* modulo 8 is greater than 0, *bits* / 8 + 1.

— If *bits* modulo 8 is equal to 0, *bits* / 8.

NOTE The values of ***minBits*** and ***bits*** are lengths in bits, while the values of ***minLength*** and ***maxLength*** are lengths in octets.

EXAMPLE

In EXPRESS:

```
ENTITY entity_with_fixed_binary_attribute;
  a_binary : BINARY(12)FIXED;
  b_binary : BINARY(8);
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Binary.12.12">
  <xs:simpleContent>
    <xs:restriction base="exp:hexBinary">
```

 

```
      <xs:maxLength value="2"/>
      <xs:minLength value="2"/>
      <xs:attribute name="extrabits" type="xs:integer" fixed="4"/>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="Binary.0.8">
  <xs:simpleContent>
    <xs:restriction base="exp:hexBinary">
      <xs:maxLength value="1"/>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="Entity_with_fixed_binary_attribute">
    <xs:complexContent>
      <xs:extension base="exp:Entity">
        <xs:all>
          <xs:element name="A_binary" type="Tns:Binary.12.12"/>
          <xs:element name="B_binary" type="Tns:Binary.0.8"/>
        </xs:all>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

#### 7.2.1.2 BOOLEAN data type

The EXPRESS `BOOLEAN` data type is mapped to the `xs:boolean` data type.

EXAMPLE

```
ENTITY entity_with_boolean_attribute;
  a_boolean  : BOOLEAN;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Entity_with_boolean_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
        <xs:element type="xs:boolean" name="A_boolean"/>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML instance document:

```
<Entity_with_boolean_attribute>
  <A_boolean>true</A_boolean>
</Entity_with_boolean_attribute>
```

#### 7.2.1.3 INTEGER data type

The EXPRESS `INTEGER` data type is mapped to the `xs:long` data type.

EXAMPLE

```
ENTITY entity_with_integer_attribute;
  an_integer : INTEGER;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Entity_with_integer_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
        <xs:element type="xs:long" name="An_integer"/>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML instance document:

```
<Entity_with_integer_attribute>
  <An_integer>1234</An_integer>
</Entity_with_integer_attribute>
```

#### 7.2.1.4 LOGICAL data type

The EXPRESS `LOGICAL` data type is mapped to the `exp:logical` data type.

The `exp:logical` data type is defined in the Base XML Schema (see Annex C) as follows:

```
<xs:simpleType name = "logical">
 <xs:restriction base = "xs:string">
    <xs:enumeration value = "false"/>
    <xs:enumeration value = "true"/>
    <xs:enumeration value = "unknown"/>
 </xs:restriction>
</xs:simpleType>
```

EXAMPLE

```
ENTITY entity_with_logical_attribute;
  a_logical  : LOGICAL;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Entity_with_logical_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
        <xs:element type="exp:logical" name="A_logical"/>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML instance document:

```
<Entity_with_logical_attribute>
```

 

```
  <A_logical>true</A_logical>
</Entity_with_logical_attribute>
```

#### 7.2.1.5 NUMBER data type

The EXPRESS `NUMBER` data type is mapped to the `xs:decimal` data type.

EXAMPLE

```
ENTITY entity_with_number_attribute;
a_number   : NUMBER;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Entity_with_number_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
        <xs:element type="xs:decimal" name="A_number"/>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML instance document:

```
<Entity_with_number_attribute>
  <A_number>1234.5678</A_number>
</Entity_with_number_attribute>
```

#### 7.2.1.6 REAL data type

The EXPRESS `REAL` data type is mapped to the `xs:double` data type. EXPRESS `REAL` data type with precision shall be mapped without restriction of the precision.

EXAMPLE 1

```
ENTITY entity_with_real_attribute;
a_real     : REAL;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Entity_with_real_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
        <xs:element type="xs:double" name="A_real"/>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML instance document:

```
<Entity_with_real_attribute>
  <A_real>1.2E02</A_real>
```

```
</Entity_with_real_attribute>
```

EXAMPLE 2: Real with precision shall be mapped identically to example 1 above:

```
ENTITY entity_with_real_attribute;
a_real     : REAL(16);
END_ENTITY;
```

#### 7.2.1.7 STRING data type

The EXPRESS `STRING` data type is mapped to the `xs:normalizedString` data type. Exception: If the EXPRESS data type is declared to have a maximum length or a `FIXED` length, the EXPRESS data type shall be mapped as specified in 7.2.1.7.1

EXAMPLE

```
ENTITY entity_with_string_attribute;
a_string   : STRING;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Entity_with_string_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
        <xs:element type="xs:normalizedString" name="A_string"/>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML instance document:

```
<Entity_with_string_attribute>
  <A_string>Hallo</A_string>
</Entity_with_string_attribute>
```

##### 7.2.1.7.1 Constrained `STRING` data types

An EXPRESS `STRING` data type for which a maximum or `FIXED` length is specified is said to be a *constrained* `STRING` data type. It shall be mapped to a `simpleType` that is a restriction of the `xs:normalizedString` data type.

When the constrained `STRING` data type is the underlying type in an EXPRESS type declaration, the corresponding XML data type shall be as specified in 7.3.1.3.

Otherwise, the derived XML schema shall contain a type definition for the XML data type corresponding to the constrained `STRING` data type. The type definition shall have the form:

```
<xs:simpleType name="String.minLength.maxLength">
    <xs:restriction base="xs:normalizedString">
      <xs:minLength value="minLength"/>
      <xs:maxLength value="maxLength"/>
    </xs:restriction>
</xs:simpleType>
```

 

where:

— ***minLength*** shall be the fixed length specified for the EXPRESS STRING data type, if it is declared FIXED. Otherwise, it shall be "0" and the xs:minLength facet may be omitted.

— ***maxLength*** shall be the maximum (or fixed) length specified for the EXPRESS STRING data type.

NOTE If the EXPRESS schema declares a defined data type for the constrained STRING data type, there will be an XML definition corresponding to the defined data type (see 7.3). The definition above is only required for constrained STRING data types that are anonymous.

EXAMPLE

```
ENTITY entity_with_string_types;
  a_bounded_string : STRING(16);
  a_fixed_length_string   : STRING(8) FIXED;
END_ENTITY;
```

In XML Schema:

```
<xs:simpleType name="String.0.16">
   <xs:restriction base="xs:normalizedString">
      <xs:maxLength value="16"/>
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="String.8.8">
   <xs:restriction base="xs:normalizedString">
      <xs:minLength value="8"/>
      <xs:maxLength value="8"/>
   </xs:restriction>
</xs:simpleType>

<xs:complexType name="Entity_with_string_types">
   <xs:complexContent>
      <xs:extension base="exp:Entity">
         <xs:all>
            <xs:element name="A bounded string" type="Tns:String.0.16"/>
            <xs:element name="A_fixed_length_string"
               type="Tns:String.8.8"/>
         </xs:all>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

XML instance document:

```
<Entity_with_string_types>
   <A_bounded_string>Public, John Q. </A_bounded_string>
   <A_fixed_length_string>jqpublic</A_fixed_length_string>
</Entity_with_string_types>
```

### 7.2.2 Aggregation data types

For an EXPRESS aggregation data type, that is, a SET, LIST, BAG or ARRAY data type, the *base-type* of the aggregate is the data type of the component values. The XML Schema data type corresponding to the aggregation data type depends on the base-type that applies to that use.

The XML data type that corresponds to a given use of the EXPRESS aggregation data type shall be as specified in Table 2 for the combination of aggregate type and base-type that applies to that use, as follows:

— The value for Base-type shall be as specified in the EXPRESS declaration for the aggregation data type. When the base type for the aggregation is a defined data type, the fundamental type of that defined data type shall be used as the Base-type value.

— *Simple* types are those whose XML representations have simpleTypes and contain no spaces, including EXPRESS types BOOLEAN, INTEGER, LOGICAL, REAL and NUMBER, and defined data types that are specializations of any of these.

— The value for Aggregate-type shall be as specified in the EXPRESS declaration for the aggregation data type. For the purposes of this table, ARRAY OF OPTIONAL base-type is considered a distinct aggregate-type.

— The applicable data type properties and the applicable values of the configuration directives should be ordered like the columns in Table 2, and compared one-for-one with the values in each row of the table.

— The applicable values will match all the values in exactly one row of the table. The corresponding XML data type shall be as specified in the subclause identified in the Subclause column of that row.

**Table 2 — Subclause governing aggregation data type correspondence**

| Aggregate type | Base type | Subclause |
|---|---|---|
| ARRAY/LIST/SET/BAG | Simple | 7.2.2.3 |
| ARRAY/LIST/SET/BAG | STRING/BINARY | 7.2.2.2 |
| ARRAY/LIST/SET/BAG | ENUMERATION | 7.2.2.5 |
| ARRAY/LIST/SET/BAG | SELECT | 7.2.2.6 |
| ARRAY/LIST/SET/BAG | Entity | 7.2.2.7 |
| ARRAY/LIST/SET/BAG | Aggregation type | 7.2.2.4 |
| ARRAY OF OPTIONAL | Simple | 7.2.2.2 |
| ARRAY OF OPTIONAL | STRING/BINARY | 7.2.2.2 |
| ARRAY OF OPTIONAL | ENUMERATION | 7.2.2.2 |
| ARRAY OF OPTIONAL | SELECT | 7.2.2.6 |
| ARRAY OF OPTIONAL | Entity | 7.2.2.7 |
| ARRAY OF OPTIONAL | Aggregation type | 7.2.2.4 |

#### 7.2.2.1 XML attributes for aggregation data types

The XML schema data types corresponding to some EXPRESS aggregation data types are the Sequence-of-elements form having all of the following attributes:

```
<xs:attribute name="arraySize">
    <xs:simpleType>
        <xs:restriction>
         <xs:simpleType>
           <xs:list itemType="xs:integer"/>
         </xs:simpleType>
```

 

```
          <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>

<xs:attribute name="itemType"  type="xs:QName"/>

<xs:attribute name="cType">
 <xs:simpleType>
   <xs:list itemType="exp:aggregateType"/>
 </xs:simpleType>
</xs:attribute>

 <xs:simpleType name="aggregateType">
    <xs:restriction base="xs:normalizedString">
        <xs:enumeration value="array"/>
        <xs:enumeration value="list"/>
        <xs:enumeration value="set"/>
        <xs:enumeration value="bag"/>
        <xs:enumeration value="array-unique"/>
        <xs:enumeration value="array-optional"/>
        <xs:enumeration value="array-optional-unique"/>
        <xs:enumeration value="list-unique"/>
    </xs:restriction>
 </xs:simpleType>
```

The value of the `arraySize` attribute indicates the number of component elements in the aggregate value represented by a particular instance. For an `ARRAY`, the value is generally fixed and equal to `HiIndex - LoIndex + 1` (see below- some exceptions apply). For a multi-dimensional array, the value is a sequence of integer values giving the number of component elements in each dimension (see 7.2.2.4 and 9.8.5).

The value of the `itemType` attribute indicates the base-type of the aggregate value — the type of the component elements. Each actual aggregation data type specifies a fixed value for `itemType` (see below).

The value of the `cType` attribute indicates the kind of EXPRESS aggregate — `SET`, `LIST`, `BAG`, `ARRAY`, `ARRAY OF UNIQUE` or `LIST OF UNIQUE`. Each actual aggregation data type specifies the required value of `cType` (see below and also 9.8).

#### 7.2.2.2 Sequence-of-elements form of aggregates

When the *sequence-of-elements* form for representation of the EXPRESS aggregation data type is specified in Clause 7.2.2, the corresponding XML Schema data type shall be a `complexType` having the form:

```
<xs:complexType>
    <xs:sequence>
        <xs:element ref = "base-instance"
            minOccurs = "lower-bound" maxOccurs = "upper-bound"/>
    </xs:sequence>
    <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
    <xs:attribute ref="exp:arraySize" usage />
    <xs:attribute ref="exp:itemType"  fixed="base-instance"/>
    <xs:attribute ref="exp:cType"  fixed="aggregate-type" />
</xs:complexType>
```

where:

— ***base-instance*** is the qualified name of the instance element corresponding to the base-type of the aggregation data type, as specified in 7.4 or 7.5.

— ***lower-bound:*** When the lower bound on the size of the collection is given and evaluates to a constant, and the EXPRESS aggregate-type is not ARRAY OF OPTIONAL, the value of `minOccurs` shall be the value of the lower bound on the size of the EXPRESS aggregate represented as a decimal integer. Otherwise the value of `minOccurs` shall be "0". If the lower bound evaluates to '1', the `minOccurs` attribute may be omitted.

— ***upper-bound:*** When the upper bound on the size of the collection is given and evaluates to a constant, the value of `maxOccurs` shall be the value of the upper bound on the size of the EXPRESS aggregate represented as a decimal integer. Otherwise the value of `maxOccurs` shall be "unbounded".

NOTE 1 If the EXPRESS aggregate is an `ARRAY`, the value of the lower and upper bounds is: `HiIndex(a) - LoIndex(a) + 1`, where `a` is the aggregation data type.

— ***aggregate-type*** is the keyword for the EXPRESS aggregate type, one of: `"set"` for `SET`, `"bag"` for `BAG`, `"list"` for `LIST`, `"array"` for `ARRAY`, `"list-unique"` for `LIST OF UNIQUE`, `"array-unique"` for `ARRAY OF UNIQUE`, `array-optional` for `ARRAY OF OPTIONAL`, and `array-optional-unique` for `ARRAY OF OPTIONAL UNIQUE`.

— ***usage***: When the EXPRESS aggregate type is `ARRAY` (and it is not optional), and the upper and lower bounds on the array evaluate to constants, `arraySize` shall be declared `fixed="`***upper-bound***`"`, where ***upper-bound*** is as defined above. When the EXPRESS aggregate type is `ARRAY`, and the upper and lower bounds on the array do not evaluate to constants, `arraySize` shall be declared use="required" Otherwise, `arraySize` shall be declared `use="optional"`.

NOTE 2 The values of this XML schema data type represent values of the EXPRESS aggregation data type as specified in 9.8.2 and 9.8.3.

NOTE 3 The above definition defines the body of the complexType, however the complexType element itself may or may not have a name attribute, as indicated in subsequent clauses.

EXAMPLE 1

In EXPRESS:

```
TYPE address = LIST [1:?] OF STRING;
END_TYPE;
```

In XML Schema:

```
<xs:complexType name="Address">
  <xs:sequence>
    <xs:element ref="exp:string-wrapper"
      minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
  <xs:attribute ref="exp:arraySize" use="optional"/>
  <xs:attribute ref="exp:itemType" fixed="exp:string-wrapper"/>
  <xs:attribute ref="exp:cType" fixed="list"/>
</xs:complexType>
```

EXAMPLE 2

In EXPRESS:

 

```
TYPE set_of_string_type = SET [1 : ?] OF defined_type_based_on_string;
END_TYPE;

TYPE defined_type_based_on_string = STRING;
END_TYPE;
```

In XML Schema:

```
<xs:complexType name="Set_of_string_type">
   <xs:sequence>
      <xs:element ref="Tns:Defined_type_based_on_string-wrapper"
        minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
   <xs:attribute ref="exp:arraySize" use="optional"/>
   <xs:attribute ref="exp:itemType"
      fixed="Tns:Defined_type_based_on_string-wrapper"/>
   <xs:attribute ref="exp:cType" fixed="set"/>
</xs:complexType>
```

#### 7.2.2.3 List-of-values form of aggregates

When the *list-of-values* form for representation of the EXPRESS aggregation data type is specified in Clause 7.2.2, the corresponding XML Schema data type is based on a simple list data type, as specified in 7.2.2.3.1. The corresponding XML Schema data type is a restriction of the unrestricted `complexType` that supports aggregations of the EXPRESS base type, as specified in 7.2.2.3.2.

The XML data type corresponding to the EXPRESS aggregation data type shall have the form:

```
<xs:complexType>
 <xs:simpleContent>
    <xs:restriction base="unrestricted-type">
       <xs:simpleType>
          <xs:restriction base="list-type">
             <xs:minLength value="lower-bound"/>
             <xs:maxLength value="upper-bound"/>
          </xs:restriction>
       </xs:simpleType>
       <xs:attribute ref="exp:arraySize" usage />
       <xs:attribute ref="exp:cType"     fixed="aggregate-type" />
    </xs:restriction>
 </xs:simpleContent>
</xs:complexType>
```

where:

— ***unrestricted-type*** shall be the qualified name of the unrestricted `complexType` that supports aggregations of the EXPRESS base-type, as specified in 7.2.2.3.2.

— ***list-type*** shall be the qualified name of the simple list data type that corresponds to the base type, as specified in 7.2.2.3.1.

— ***lower-bound*** is the lower bound on the EXPRESS aggregate. When the lower bound is given and evaluates to a constant, the value of `minLength` shall be the value of the lower bound, represented as a decimal integer. Otherwise, the value of `minLength` shall be '0'. If the value of `minLength` is '0', the `minLength` facet may be omitted.

— ***upper-bound*** is the upper bound on the EXPRESS aggregate. When the upper bound is given and evaluates to a constant, the `maxLength` attribute shall appear and shall have the value of the upper bound, represented as a decimal integer. Otherwise, the `maxLength` facet shall be omitted.

NOTE If the EXPRESS aggregation type is an `ARRAY`, the value of the lower and upper bounds is `HiIndex(a) - LoIndex(a) + 1`, where `a` is the aggregation data type.

— if the lower bound on the EXPRESS aggregation data type is 0 or unspecified and the upper bound is indeterminate or unspecified, the restriction on the simple list type may be omitted from the declaration, that is, only the values of the XML attributes are restricted.

— ***aggregate-type*** is the keyword for the EXPRESS aggregate type, one of: `"set"` for `SET`, `"bag"` for `BAG`, `"list"` for `LIST`, `"array"` for `ARRAY`, `"list-unique"` for `LIST OF UNIQUE`, `"array-unique"` for `ARRAY OF UNIQUE`. The list-of-values form is not used for `ARRAY OF OPTIONAL`.

— ***usage***: When the EXPRESS aggregate type is `ARRAY` and the upper and lower bounds on the array evaluate to constants, `arraySize` shall be declared `fixed="`***upper-bound***`"`, where ***upper-bound*** is as defined above. When the EXPRESS aggregate type is `ARRAY` and the upper and lower bounds on the array do not evaluate to constants, `arraySize` shall be declared use="required" Otherwise, `arraySize` shall be declared `use="optional"`.

EXAMPLE 1

In EXPRESS:

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

The simple type that supports aggregations of `REAL` must be defined as `Tns:List-double`, as shown in the example in 7.2.2.3.1. The unrestricted `complexType` that supports aggregations of `REAL` must be defined as `Tns:Seq-double`, as shown in the example in 7.2.2.3.2. The XML data type corresponding to `compoundplaneanglemeasure` is:

```
<xs:complexType name="Compoundplaneanglemeasure">
  <xs:simpleContent>
    <xs:restriction base="Tns:Seq-double">
      <xs:simpleType>
        <xs:restriction base="Tns:List-double">
          <xs:minLength value="3"/>
          <xs:maxLength value="3"/>
        </xs:restriction>
      </xs:simpleType>
      <xs:attribute ref="exp:arraySize" fixed="3" />
      <xs:attribute ref="exp:cType" fixed="array" />
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

EXAMPLE 2

In EXPRESS

```
TYPE length_measure = REAL; END_TYPE;

TYPE measurements = LIST [1:?] OF length_measure;
END_TYPE;
```

 

The simple type that supports aggregations of `length_measure` must be defined as `Tns:List-Length_measure`, as shown in the example in 7.2.2.3.1. The unrestricted `complexType` that supports aggregations of `length_measure` must be defined in the derived XML schema as shown in the examples in 7.2.2.3.2.

And the XML data type corresponding to the EXPRESS type `measurements` is:

```
<xs:complexType name="Measurements">
  <xs:simpleContent>
    <xs:restriction base="Tns:Seq-Length_measure">
      <xs:simpleType>
        <xs:restriction base="Tns:List-Length_measure">
          <xs:minLength value="1"/>
        </xs:restriction>
      </xs:simpleType>
      <xs:attribute ref="exp:cType" fixed="list" />
      <xs:attribute ref="exp:arraySize" use="optional" />
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

#### 7.2.2.3.1 Simple list type supporting aggregations

When the *list-of-values* form for representation of the EXPRESS aggregation data type is specified in Clause 7.2.2, the corresponding XML Schema data type is based on a simple list data type. The simple list data type shall be defined in the derived XML schema by a `simpleType` definition of the form:

```
<xs:simpleType name="list-type" >
 <xs:list itemType="base-type" />
</xs:simpleType>
```

where:

— ***list-type*** shall be "`List-`", followed by the local part of the name of the XML schema data type that corresponds to the base-type of the aggregation data type, as specified in 7.2 or 7.3.

— ***base-type*** shall be the qualified name of the XML schema data type that corresponds to the base-type of the aggregation data type, as specified in 7.2 or 7.3.

Only one such definition is needed for all EXPRESS aggregation types having the same base-type.

EXAMPLE

In EXPRESS:

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

TYPE length_measure = REAL; END_TYPE;

TYPE measurements = LIST [1:?] OF length_measure;
END_TYPE;
```

The corresponding simple types are defined in the derived XML schema as:

```
<xs:simpleType name="List-double">
 <xs:list itemType="xs:double"/>
```

```
</xs:simpleType>

<xs:simpleType name="List-Length_measure">
 <xs:list itemType="Tns:Length_measure"/>
</xs:simpleType>
```

#### 7.2.2.3.2 Unrestricted complexType supporting aggregations

A single unrestricted `complexType` supports all aggregations (`ARRAY`, `BAG`, `LIST`, `SET`) of a given EXPRESS base-type. The XML data type that corresponds to each specific aggregation data type with that base-type is a restriction of the unrestricted `complexType`.

The derived XML schema shall contain a `complexType` definition for the unrestricted `complexType`. Only one such definition is needed for all EXPRESS aggregation types having the same base-type. The definition shall have the form:

```
<xs:complexType name="seq-type">
 <xs:simpleContent>
    <xs:extension base="Tns:list-type">
       <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
       <xs:attribute ref="exp:arraySize" use="optional"/>
       <xs:attribute ref="exp:itemType" fixed="base-type" />
       <xs:attribute ref="exp:cType" default="set"/>
    </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

where:

— ***Tns:list-type*** shall be the qualified name for the simple ***list-type*** defined in 7.2.2.3.1 above.

— ***seq-type*** shall be "`Seq-`", followed by the local part of the name of the XML schema data type that corresponds to the base-type of the aggregation data type, as specified in 7.2 or 7.3.

— ***base-type***, the `itemType` attribute value, shall be the qualified name of the XML schema data type that corresponds to the base-type of the aggregation data type, as specified in 7.2 or 7.3.

EXAMPLE 1

In EXPRESS:

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

The unrestricted `complexType` for aggregations of `REAL` must be defined in the derived XML schema, as:

```
<xs:complexType name="Seq-double">
 <xs:simpleContent>
    <xs:extension base="Tns:List-double">
       <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
       <xs:attribute ref="exp:arraySize" use="optional"/>
       <xs:attribute ref="exp:itemType" fixed="xs:double" />
       <xs:attribute ref="exp:cType" default="set"/>
    </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

 

This definition appears only once, even if the context schema contains several aggregation types whose base-type is REAL.

EXAMPLE 2

In EXPRESS:

```
TYPE length_measure = REAL; END_TYPE;

TYPE measurements = LIST [1:?] OF length_measure;
END_TYPE;
```

The unrestricted complexType for aggregations of length_measure must be defined in the derived XML schema, as:

```
<xs:complexType name="Seq-Length_measure">
 <xs:simpleContent>
    <xs:extension base="Tns:List-Length_measure">
       <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
       <xs:attribute ref="exp:arraySize" use="optional"/>
       <xs:attribute ref="exp:itemType" fixed="Tns:Length_measure " />
       <xs:attribute ref="exp:cType" default="set"/>
    </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

This definition appears only once, even if the context schema contains several aggregation types whose base-type is length_measure.

#### 7.2.2.4 Aggregates of aggregation data types

When the base-type of the EXPRESS aggregation data type (called ***type A*** below) is another aggregation data type, or a defined data type whose fundamental type is another aggregation data type, the corresponding XML schema data type shall have the multi-dimensional array structure. The structure of the corresponding XML schema data type shall be as specified in this subclause.

An EXPRESS aggregation data type (SET, LIST, BAG or ARRAY) is said to be *nested*, if it appears in the context schema as the base-type of another aggregation data type.

Let the *deepest underlying type* of an aggregation type be determined as follows: If the base-type of the aggregation is an anonymous aggregation type, a defined data type whose underlying type is an aggregation type, or a SELECT type whose working select list contains a single defined data type whose underlying type is an aggregation type, the deepest underlying type is the deepest underlying type of that aggregation. Otherwise, the deepest underlying type is the base-type of the aggregation.

Type B shall be the deepest underlying type of type A.

The complexType corresponding to ***type A*** shall have a content model that only allows elements corresponding to ***type B***. In addition, the complexType shall specify fixed values for the exp:itemType attribute, and the exp:cType attribute, as specified below.

The XML data type shall have the form:

```
<xs:complexType>
        <xs:sequence>
            <element-or-group ref="base-instance"
                minOccurs = "lower-bound" maxOccurs = "upper-bound" />
        </xs:sequence>
```

```
        <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
        <xs:attribute ref="exp:arraySize"  usage/>
        <xs:attribute ref="exp:itemType"  fixed="base-instance"/>
        <xs:attribute ref="exp:cType"     fixed="aggregate-types"/>
</xs:complexType>
```

where:

— ***base-instance:*** If ***type B*** is an entity data type, ***base-instance*** shall be the qualified name of the corresponding complexEntity group (see 7.5.6.2). If ***type B*** is a SELECT data type, ***base-instance*** shall be the qualified name of the corresponding group or instance element (see 7.3.4). Otherwise, ***base-instance*** shall be the qualified name of the corresponding instance element (see 7.4).

— ***element-or-group*** is `xs:element`, if ***base-instance*** is an element, or `xs:group`, if ***base-instance*** is a group.

NOTE 1 In general, ***base-instance*** is an element. But ***base-instance*** is a group if ***type B*** is a `SELECT` data type (see 7.3.4) or an entity data type that uses the inheritance-free mapping (see 7.5).

— ***aggregate-types*** is a list of the keywords for the EXPRESS aggregation data types that are part of the multi-dimensional structure, in order, with the keyword for ***type A*** first and the keyword for the most nested aggregate last. Each keyword shall be one of: `"set"` for `SET`, `"bag"` for `BAG`, `"list"` for `LIST`, `"array"` for `ARRAY`, `"list-unique"` for `LIST OF UNIQUE`, `"array-unique"` for `ARRAY OF UNIQUE`, `array-optional` for `ARRAY OF OPTIONAL`, and `array-optional-unique` for `ARRAY OF OPTIONAL UNIQUE`. The items in the list shall be separated by whitespace.

— ***lower-bound***: When none of the base-types are `OPTIONAL`, and the lower bound on the size of each collection is given and evaluates to a constant, the value of the `minOccurs` attribute shall be specified as the product of the lower bounds of each aggregation data type. Otherwise, the value of the `minOccurs` attribute shall be 0. If the lower bound evaluates to '1', the `minOccurs` attribute may be omitted.

— ***upper-bound***: When the upper bound on the size of each collection is given and evaluates to a constant, the value of the `maxOccurs` attribute shall be specified as the product of the upper bounds of each aggregation data type. Otherwise, the value shall be “unbounded”.

— ***usage***: When type A and all of the nested aggregation data types are `ARRAYS`, and none of them are optional, and all of the index range bounds on type A and the nested ARRAYs evaluate to constants, `arraySize` shall be declared `fixed="`$size_1$ ... $size_N$`"`, where $size_1$ ... $size_N$ are the sizes of ***type A*** and all the nested aggregates that are part of the multi-dimensional structure, in order with the size of ***type A*** first and the size of the most nested aggregate last. When type A and all of the nested aggregation data types are `ARRAYS`, and none of them are optional, and the index range bounds on type A or any of the nested ARRAYs does not evaluate to a constant, `arraySize` shall be declared `use="required"`. Otherwise, `arraySize` shall be declared `use="optional"`.

NOTE 2 If the aggregate is an `ARRAY`, what is given in the EXPRESS syntax is the `LOINDEX` and `HIINDEX` values — the subscript range. The size of the dimension is the difference between the index values plus 1.

NOTE 3 The multi-dimensional array form is a special case of the "sequence-of-elements" form specified in 7.2.2.2. It has the same structure, but the component instance elements represent values of the deepest underlying type instead of the base-type of ***type A***. The proper interpretation of this structure depends on the values of `arraySize` and possibly `cType`.



 

NOTE 4 This XML schema data type is used to represent the values of the array as specified in 9.8.5.3 or 9.8.5.4.

EXAMPLE 1

In EXPRESS:

```
TYPE Matrix = ARRAY[1:3] OF ARRAY[1:2] OF OPTIONAL REAL;
END_TYPE;

ENTITY X;
  matrix : Matrix;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Matrix">
       <xs:sequence>
         <xs:element ref = "exp:double-wrapper" minOccurs="0"
maxOccurs="6"/>
       </xs:sequence>
       <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
       <xs:attribute ref = "exp:arraySize" fixed = "3 2"/>
       <xs:attribute ref = "exp:itemType" fixed = "exp:double-
wrapper"/>
       <xs:attribute ref = "exp:cType" fixed = "array array-
optional"/>
</xs:complexType>
```

In XML instance data (as it would appear inside an accessor element):

```
<exp:double-wrapper pos = "0 0">1.0</exp:double-wrapper >
<exp:double-wrapper pos = "0 1">2.0</exp:double-wrapper >
<exp:double-wrapper pos = "1 0">0.0</exp:double-wrapper >
<exp:double-wrapper pos = "1 1">4.0</exp:double-wrapper >
<exp:double-wrapper pos = "2 0">0.0</exp:double-wrapper >
<exp:double-wrapper pos = "2 1">5.4</exp:double-wrapper >
```

EXAMPLE 2 multi-dimensional aggregation of simple types:

In EXPRESS:

```
TYPE direction_vectors = LIST[1:3] OF ARRAY[1:2] OF REAL;
END_TYPE;
```

In XML Schema:

```
<xs:complexType name="Direction_vectors">
       <xs:sequence>
         <xs:element ref = "exp:double-wrapper" minOccurs="2"
maxOccurs="6"/>
       </xs:sequence>
       <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
       <xs:attribute ref = "exp:arraySize" fixed = "3 2"/>
       <xs:attribute ref = "exp:itemType" fixed = "exp:double-
wrapper"/>
       <xs:attribute ref = "exp:cType" fixed = "list array"/>
</xs:complexType>
```

In XML instance data (as it would appear inside an accessor element)::

```
<exp:double-wrapper pos = "0 0">1.0</exp:double-wrapper>
<exp:double-wrapper pos = "0 1">2.0</exp:double-wrapper>
<exp:double-wrapper pos = "1 0">0.0</exp:double-wrapper>
<exp:double-wrapper pos = "1 1">4.0</exp:double-wrapper>
<exp:double-wrapper pos = "2 0">0.0</exp:double-wrapper>
<exp:double-wrapper pos = "2 1">5.4</exp:double-wrapper>
```

#### 7.2.2.5 Aggregates of ENUMERATION data types

When the base-type of the EXPRESS aggregation data type is a defined data type whose fundamental type is an ENUMERATION type, the corresponding XML Schema data type shall be as specified in 7.2.2.3.

EXAMPLE

In EXPRESS:

```
TYPE material_type = ENUMERATION OF (steel, copper, plastic);
END_TYPE;

TYPE list_of_materials = SET [1:?] of material_type;
END_TYPE;
```

In XML Schema:

```
<xs:simpleType name = "Material_type">
  <xs:restriction base = "xs:string">
       <xs:enumeration value = "steel"/>
       <xs:enumeration value = "copper"/>
       <xs:enumeration value = "plastic"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="List-Material_type">
  <xs:list itemType="Tns:Material_type"/>
</xs:simpleType>

<xs:complexType name="Seq-Material_type">
  <xs:simpleContent>
    <xs:extension base="Tns:List-Material_type">
      <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
      <xs:attribute ref="exp:arraySize" use="optional"/>
      <xs:attribute ref="exp:itemType" fixed="Tns:Material_type"/>
      <xs:attribute ref="exp:cType" default="set"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="List_of_materials">
  <xs:simpleContent>
    <xs:restriction base="Tns:Seq-Material_type">
      <xs:simpleType>
        <xs:restriction base="Tns:List-Material_type">
          <xs:minLength value="1"/>
        </xs:restriction>
```

 

```
          </xs:simpleType>
          <xs:attribute ref="exp:arraySize" use="optional"/>
          <xs:attribute ref="exp:cType"  fixed="set" />
        </xs:restriction>
      </xs:simpleContent>
    </xs:complexType>
```

#### 7.2.2.6 Aggregates of SELECT data types

When the base-type of the EXPRESS aggregation data type is a defined data type whose fundamental type is a `SELECT` type, and the working select-list of the select type (see 7.3.4.1) contains only one data type, the base-type is considered to be that data type, instead of the `SELECT` type, and the subclause in 7.2.2 specified for that data type shall apply. In all other cases, when the base-type of the EXPRESS aggregation data type is a defined data type whose fundamental type is a `SELECT` type, the corresponding XML schema data type shall be a `complexType`.

If the underlying type of the defined data type is not the same as the fundamental type (that is, if the defined data type is a *specialization* of a `SELECT` type), the `complexType` shall be as specified in 7.2.2.2, using the defined data type as the base-type.

In all other cases, the `complexType` shall have the form:

```
<xs:complexType>
    <xs:sequence>
        <group-or-element ref="base-group"
            minOccurs="lower-bound"
            maxOccurs="upper-bound"/>
    </xs:sequence>
    <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
    <xs:attribute ref="exp:arraySize" usage/>
    <xs:attribute ref="exp:itemType" fixed="base-type"/>
    <xs:attribute ref="exp:cType" fixed="aggregate-type"/>
</xs:complexType>
```

where:

— ***base-group*** is the qualified name of the XML group or instance element that corresponds to the defined data type, as specified in 7.3.4.

— ***group-or-element:*** shall be xs:group if ***base-group*** is a group, otherwise, xs:element.

— ***lower-bound***: When the EXPRESS aggregate-type is not `ARRAY OF OPTIONAL`, and the lower bound on the size of the EXPRESS aggregate collection is given and does not evaluate to a constant; it is the value of the lower bound, represented as a decimal integer. Otherwise, it is '`0`'. If the lower bound evaluates to '`1`', the `minOccurs` attribute may be omitted.

— ***upper-bound*** is the upper bound on the EXPRESS aggregate, represented as a decimal integer. When the EXPRESS aggregate is an `ARRAY` and the upper bound on the EXPRESS aggregate is given and evaluates to a constant, the `maxOccurs` value shall be (`HiIndex(`***a***`) - LoIndex(`***a***`) + 1`), where ***a*** is the aggregation data type. When the EXPRESS aggregate is not an `ARRAY`, and the upper bound on the EXPRESS aggregate is given and evaluates to a constant, the value shall be the value of the upper bound represented as a decimal integer. Otherwise the value shall be "unbounded".

— ***base-type*** is the qualified name of the XML data type corresponding to the base-type of the aggregate, that is, the defined data type whose fundamental type is the `SELECT` data type, as specified in 7.3.

— ***aggregate-type*** is the keyword for the EXPRESS aggregate type, one of: "set" for SET, "bag" for BAG, "list" for LIST, "array" for ARRAY, "list-unique" for LIST OF UNIQUE, "array-unique" for ARRAY OF UNIQUE, array-optional for ARRAY OF OPTIONAL, and array-optional-unique for ARRAY OF OPTIONAL UNIQUE.

— ***usage***: When the EXPRESS aggregate type is ARRAY and the upper and lower bounds on the array evaluate to constants, arraySize shall be declared fixed="***upper-bound***", where ***upper-bound*** is as defined above. When the EXPRESS aggregate type is ARRAY and the upper and lower bounds on the array do not evaluate to constants, arraySize shall be declared use="required". Otherwise, arraySize shall be declared use="optional".

NOTE 1 If the aggregate is an ARRAY, what is given in the EXPRESS syntax is the LOINDEX and HIINDEX values — the subscript range. The minimum and maximum size of the aggregate values is the same – the difference between the index values plus 1.

NOTE 2 This form is identical in all regards to the "sequence-of-elements" form specified in 7.2.2.2, except that the content model of the complexType is a repeating group instead of a repeating element. It is a "sequence of elements" representation for the purposes of any other clause.

EXAMPLE

In EXPRESS:

```
TYPE Label = STRING;
END_TYPE;

TYPE Length_measure = REAL;
END_TYPE;

TYPE Traffic_light = ENUMERATION OF (red, yellow, green);
END_TYPE;

TYPE Inner = SELECT (Label, Length_measure);
END_TYPE;

ENTITY Pipe;
   ...
END_ENTITY;

TYPE S = SELECT (Length_measure, Traffic_light, Pipe, Inner);
END_TYPE;

TYPE Select_collection = SET OF S;
END_TYPE;
```

In XML Schema, the declaration corresponding to SELECT_COLLECTION is:

```
<xs:complexType name = "Select_collection">
     <xs:sequence>
       <xs:group ref="Tns:S"
          minOccurs = "0" maxOccurs = "unbounded"/>
     </xs:sequence>   <xs:attribute name="ref" type="xs:IDREF"
       use="optional"/>
     <xs:attribute ref ="exp:cType" fixed = "set"/>
     <xs:attribute ref="exp:arraySize" use = "optional"/>
     <xs:attribute ref="exp:itemType"  fixed="Tns:S"/>
</xs:complexType>
```

#### 7.2.2.7 Aggregates of ENTITY data types

When the base-type of the EXPRESS aggregation data type is an entity data type, the corresponding XML schema data type has the the sequence of entity instances form, and the corresponding XML schema data type shall be a `complexType` having the form:

```
<xs:complexType>
        <xs:sequence>
            <xs:group ref="base-complexEntity-group"
                minOccurs="lower-bound"
                maxOccurs="upper-bound"/>
        </xs:sequence>
        <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
        <xs:attribute ref="exp:arraySize" usage/>
        <xs:attribute ref="exp:itemType" fixed="base-instance"/>
        <xs:attribute ref="exp:cType" fixed="aggregate-type"/>
</xs:complexType>
```

where:

— ***base-instance*** is the qualified name of the entity instance element corresponding to the base-type of the aggregate, as specified in 7.5.5.

— ***base-complexEntity-group*** is the qualified name of the complexEntity group corresponding to the base-type of the aggregate, as specified in 7.5.6.2.

— ***lower-bound***: When the lower bound on the size of the collection is given and evaluates to a constant, the value of `minOccurs` is the value of the lower bound on the EXPRESS aggregate, represented as a decimal integer. Otherwise, the value shall be `"0"`.

— ***upper-bound*** is the upper bound on the EXPRESS aggregate. When the EXPRESS aggregate is an `ARRAY` and the upper bound on the EXPRESS aggregate is given and evaluates to a constant, the `maxOccurs` value shall be (`HiIndex(`***a***`) - LoIndex(`***a***`) + 1`), where ***a*** is the aggregation data type. When the EXPRESS aggregate is not an `ARRAY`, and the upper bound on the EXPRESS aggregate is given and evaluates to a constant, the value shall be the value of the upper bound represented as a decimal integer. Otherwise the value shall be "unbounded".

NOTE 1 If the EXPRESS aggregate is an `ARRAY`, the value of the lower and upper bounds = `HiIndex(a) - LoIndex(a) + 1`, where `a` is the aggregation data type.

— ***aggregate-type*** is the keyword for the EXPRESS aggregate type, one of: `"set"` for `SET`, `"bag"` for `BAG`, `"list"` for `LIST`, `"array"` for `ARRAY`, `"list-unique"` for `LIST OF UNIQUE`, `"array-unique"` for `ARRAY OF UNIQUE`, `array-optional` for `ARRAY OF OPTIONAL`, and `array-optional-unique` for `ARRAY OF OPTIONAL UNIQUE`.

— ***usage***: When the EXPRESS aggregate type is `ARRAY` and the upper and lower bounds on the array evaluate to constants, `arraySize` shall be declared `fixed="`***upper-bound***`"`, where ***upper-bound*** is as defined above. If the EXPRESS aggregate type is `ARRAY` and the upper and lower bounds on the array do not evaluate to constants, `arraySize` shall be declared `use="required"`. Otherwise, `arraySize` shall be declared `use="optional"`.

NOTE 2 This form is identical in all regards to the "sequence-of-elements" form specified in 7.2.2.2, except that the content model of the `complexType` may be a repeating group instead of a repeating element. It is a "sequence of elements" representation for the purposes of any other clause.

EXAMPLE

In EXPRESS:

```
ENTITY Pt3d;
  -- additional attributes skipped
END_ENTITY;

TYPE Pt3dSet = SET OF Pt3d;
END_TYPE;

ENTITY Pt2d;
  -- additional attributes skipped
END_ENTITY;

TYPE Pt2dList = LIST OF Pt2d;
END_TYPE;
```

In XML Schema:

```
<xs:complexType name="Pt3dset">
       <xs:sequence>
         <xs:group ref="Tns:Pt3d-complexEntity-group"
           minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
      <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
       <xs:attribute ref="exp:arraySize" use="optional"/>
       <xs:attribute ref="exp:itemType" fixed="Tns:Pt3d"/>
       <xs:attribute ref="exp:cType" fixed="set"/>
</xs:complexType>

<xs:complexType name="Pt2dlist">
       <xs:sequence>
         <xs:group ref="Tns:Pt2d-complexEntity-group"
           minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
      <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
       <xs:attribute ref="exp:arraySize" use="optional"/>
       <xs:attribute ref="exp:itemType" fixed="Tns:Pt2d"/>
       <xs:attribute ref="exp:cType" fixed="list"/>
</xs:complexType>
```

### 7.2.3 Constructed data types

EXPRESS defines two kinds of data types, called "constructed data types", that can only occur in the definition of a defined data type. There are no XML Schema data types that correspond to the constructed data types as such, because in EXPRESS they cannot be the data type of anything.

#### 7.2.3.1 ENUMERATION data types

The XML schema data type that corresponds to an `ENUMERATION` data type is the one that corresponds to the EXPRESS defined data type. See 7.2.4 and 7.3.3.

#### 7.2.3.2 SELECT data types

The XML schema data type that corresponds to a `SELECT` data type is the one that corresponds to the EXPRESS defined data type. See 7.2.4 and 7.3.4.

### 7.2.4 Defined data types

For each EXPRESS defined data type, the corresponding XML schema data type is a named data type defined by an XML schema data type definition. The XML data type name and its definition shall be as specified in 7.3.

### 7.2.5 ENTITY data types

For each EXPRESS entity data type, the corresponding XML schema data type is a named data type defined by an XML schema data type definition. The XML data type name and its definition shall be as specified in 7.5.

## 7.3 XML Schema definitions and declarations for EXPRESS defined data types

An EXPRESS *defined data type* is a data type that is defined by an EXPRESS type declaration. The type declaration defines the data type in terms of an *underlying type*. The defined data type is said to be a *specialization* of the underlying type. The underlying type of an EXPRESS defined data type, however, may itself be an EXPRESS defined data type, which is defined by an EXPRESS type declaration to be based on another underlying type, and so on. Ultimately, there is a *fundamental type* that is not itself an EXPRESS defined data type, but rather a data type defined by the EXPRESS language.

NOTE 1 The terms *defined data type*, *underlying type*, *specialization*, and *fundamental type* are taken from ISO 10303-11:2003 and are used extensively in this Part of ISO 10303 in defining the mapping of defined data types and the encoding of their values.

For each EXPRESS defined data type that is declared in, or explicitly interfaced into the context schema, the derived XML schema shall contain an XML schema data type definition.

The XML schema data type definition shall depend on the underlying type of the EXPRESS defined data type, as follows:

— If the underlying type of the defined data type is a simple data type (`BINARY`, `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER`, `REAL`, or `STRING`), the XML schema data type definition shall be as specified in 7.3.1.

— If the underlying type of the defined data type is an aggregation data type (`ARRAY`, `BAG`, `LIST`, or `SET`), the XML schema data type definition shall be as specified in 7.3.2.

— If the underlying type of the defined data type is an `ENUMERATION` data type, the XML schema data type definition shall be as specified in 7.3.3.

— If the underlying type of the defined data type is a `SELECT` data type, the XML schema data type definition shall be as specified in 7.3.4.

— If the underlying type of the defined data type is another defined data type, the XML schema data type definition shall be as specified in 7.3.5.

The derived XML schema may also contain a corresponding instance element declaration, as specified in 7.4.3. The XML data type definition shall be independent of, and appear outside of, the instance element declaration.

NOTE 2 The underlying type of an EXPRESS defined data type cannot be an entity data type.

### 7.3.1 Simple underlying types

When the underlying type is a simple data type (BINARY, BOOLEAN, INTEGER, LOGICAL, NUMBER, REAL, or STRING), the corresponding XML schema data type definition shall be as specified in this subclause.

When the underlying type is a BINARY data type, the XML data type definition shall be as specified in 7.3.1.1.

When the underlying type is BOOLEAN, INTEGER, LOGICAL, NUMBER or REAL, the XML data type definition shall be as specified in 7.3.1.2.

When the underlying type is a STRING data type, the XML data type definition shall be as specified in 7.3.1.3.

#### 7.3.1.1 Definitions for BINARY data types

When the underlying type is a BINARY data type, the XML data type definition shall have the form:

```
<xs:complexType name = "identifier">
 <xs:simpleContent>
  <xs:restriction base = "exp:hexBinary">
  </xs:restriction>
 </xs:simpleContent>
</xs:complexType>
```

where:

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 7.1.2.

When the underlying type is a constrained BINARY data type (see 7.2.1.1.1), the XML data type definition shall have the form:

```
<xs:complexType name = "identifier">
  <xs:restriction base="exp:hexBinary">
    <xs:minLength value="minLength"/>
    <xs:maxLength value="maxLength"/>
    <xs:attribute name="extraBits" fixed="remainder" />
  </xs:restriction>
</xs:complexType>
```

where:

— ***identifier*** and ***binary-type*** are as above, and

— ***minLength***, ***maxLength*** and ***remainder*** are as specified in 7.2.1.1.1.

#### 7.3.1.2 Definitions for BOOLEAN, INTEGER, LOGICAL, NUMBER or REAL

When the underlying type is BOOLEAN, INTEGER, LOGICAL, NUMBER or REAL, the corresponding XML data type definition shall have the form:

```
<xs:simpleType name = "identifier">
 <xs:restriction base = "underlying-type">
 </xs:restriction>
```

```
</xs:simpleType>
```

where

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 7.1.2, and

— ***underlying-type*** is the XML schema data type corresponding to the underlying type, as specified in 7.2.1.

#### 7.3.1.3 Definitions for STRING data types

When the underlying type is an unconstrained STRING data type, the corresponding XML data type definition shall have the form:

```
<xs:simpleType name = "identifier">
 <xs:restriction base = "xs:normalizedString">
 </xs:restriction>
</xs:simpleType>
```

where

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 7.1.2.

When the underlying type is a constrained STRING data type (see 7.2.1.7.1), the XML data type definition shall have the form:

```
<xs:simpleType name = "identifier">
  <xs:restriction base="xs:normalizedString">
    <xs:minLength value="minLength"/>
    <xs:maxLength value="maxLength"/>
  </xs:restriction>
</xs:simpleType>
```

where:

— ***identifier*** is as above.

— ***minLength*** shall be the fixed length specified for the EXPRESS STRING data type, if it is declared FIXED. Otherwise, the xs:minLength facet shall be omitted.

— ***maxLength*** shall be the maximum or fixed length specified for the EXPRESS STRING data type.

### 7.3.2 Aggregate underlying types

When the underlying type of the EXPRESS defined data type is an aggregation data type — a SET, LIST, BAG or ARRAY type — the XML Schema data type that corresponds to the underlying type depends on several factors, as specified in 7.2.2. The form of the XML Schema data type definition for the defined data type depends on the form of the XML Schema data type corresponding to the underlying, but all of them are complexTypes.

Therefore, the corresponding XML data type definition shall have the general form:

```
<xs:complexType name="identifier">
 (body)
```

```
</xs:complexType>
```

where:

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 7.1.2, and

— ***(body)*** is the model of the `complexType` that corresponds to the underlying aggregation data type, as specified in 7.2.2.

NOTE For all practical purposes, the data type definition is identical to the corresponding data type, except that `name="`***identifier***`"` is added to the initial `<xs:complexType>` element.

### 7.3.3 ENUMERATION underlying types

When the underlying type of the EXPRESS defined data type is an `ENUMERATION` data type, the corresponding XML schema data type shall be a `simpleType` that is defined as a restriction of the data type `xs:string` using enumeration facets.

For each enumeration item in the EXPRESS type declaration, there shall be a corresponding XML schema `enumeration` facet whose value shall be the string value comprising the characters of the EXPRESS identifier for the enumeration item, all in lower case.

That is, the XML schema data type definition shall have the form:

```
<xs:simpleType name = "identifier">
 <xs:restriction base = "xs:string">
        <xs:enumeration value = "enum-item-1"/>
        ...
        <xs:enumeration value = "enum-item-n"/>
 </xs:restriction>
</xs:simpleType>
```

where:

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 7.1.2, and

— ***enum-item-1... enum-item-n*** are the identifiers for the EXPRESS enumeration items, in lower case.

EXAMPLE

In EXPRESS:

```
TYPE source = ENUMERATION OF (made,bought,not_known);
END_TYPE;
```

In XML Schema:

```
<xs:simpleType name="Source">
   <xs:restriction base="xs:string">
     <xs:enumeration value="made"/>
     <xs:enumeration value="bought"/>
     <xs:enumeration value="not_known"/>
  </xs:restriction>
</xs:simpleType>
```

 

### 7.3.4 SELECT underlying types

When the underlying type of the EXPRESS defined data type is a SELECT data type, the corresponding XML schema data type shall depend on the number of data types in the working select list as specified in 7.3.4.1.

If the working select list contains no data types, the SELECT data type is not mapped to the derived XML schema.

If the working select list contains only one data type:

— the corresponding XML Schema data type shall be the XML Schema data type that corresponds to that data type, and

— the corresponding instance element shall be the instance element that corresponds to that data type.

Otherwise, the derived XML schema shall contain an XML group declaration, as specified in 7.3.4.2, and a complexType declaration, as specified in 7.3.4.3. The corresponding XML Schema data type shall be the complexType specified in 7.3.4.3.

NOTE For a SELECT data type that has more than one type in the select list, there is no corresponding instance element (see 7.4). Instead, the group specified in 7.3.4.2 is the equivalent of the instance element for the SELECT data type.

#### 7.3.4.1 Working select list

The working select list for the SELECT data type shall be developed as follows:

— The working select list shall be initialized to the set of EXPRESS data types in the select-list of the SELECT data type.

— For each EXPRESS data type in the working select list that is a defined data type whose (immediate) underlying type is a SELECT type, the defined data type shall be deleted from the working select list and replaced by all of the data types in the select-list of its underlying SELECT type.

— The previous step shall be repeated until there are no defined data types in the working select list whose (immediate) underlying type is a SELECT type.

— For each EXPRESS entity data type that appears in the working select list, all of the subtypes that appear in the corresponding subtypes group (if any, see 7.5.6.1) shall be added to the working select list. Exception: no data type shall be added to the working select list that is already on the working select list.

NOTE 1 Using the type graph associated with the entity data type (see 7.5.1), every leaf node that has a path to the node corresponding to the entity data type represents a possible subtype, and every node that lies on such a path represents a possible subtype.

— No data type that is already on the working select list shall be added to the working select list.

— All abstract entity data types (see 6.3) shall be deleted from the working select list.

NOTE 2 The working select list is the set of all EXPRESS data types whose instances are admissible as instances of the original SELECT data type. Nested SELECT data types are expanded into the list.

NOTE 3 A defined data type whose underlying type is another defined data type is *not* replaced/expanded in the working select list, even when its fundamental type is a SELECT data type. A defined data type whose underlying type is another defined data type is always treated as a specialization and that information is retained in the XML schema type model.

EXAMPLE 1

In EXPRESS:

```
TYPE an_int = INTEGER;
END_TYPE;

TYPE an_odd_int= an_int;
WHERE odd : NOT (SELF = 2*(SELF/2));
END_TYPE;

TYPE my_select = SELECT (an_int, an_odd_int);
END_TYPE;
```

Working select list for MY_SELECT: { an_int , an_odd_int }.

EXAMPLE 2 (continue from above)

In EXPRESS:

```
TYPE my_select2 = SELECT (an_odd_int);
END_TYPE;
```

Working select list for MY_SELECT2: { an_odd_int }. See 7.6.3.2 for an example of an attribute of type my_select2.

### 7.3.4.2 XML group declaration for SELECT data types

When the underlying type of the EXPRESS defined data type is a SELECT data type that has a working select list containing more than one data type, the derived XML schema shall contain a group declaration that corresponds to the EXPRESS defined data type.

The name of the group shall be derived from the EXPRESS identifier for the defined data type as specified in 7.1.3.

The contents of the group is derived from the working select list of the SELECT data type as specified in 7.3.4.1

The group shall contain a single choice element. For each EXPRESS data type in the working select list, the choice element shall contain an element declaration for the instance element corresponding to that EXPRESS data type (as specified in 7.4), or for each of the instance elements found in its subtype group (see 7.5.6.1). The element declaration shall have a ref attribute whose value is the qualified name of the instance element.

That is, the group declaration shall have the form:

```
<xs:group name = "identifier" >
 <xs:choice>
     <xs:element ref="select-list-item-1"/>
     ...
     <xs:element ref="select-list-item-n"/>
     complexEntity-usage
```

 

```
    </xs:choice>
  </xs:group>
```

where

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 7.1.2;

— ***select-list-item-1*** … ***select-list-item-n:*** the qualified name of each instance element in the SELECT group contents list, determined as follows:

- The SELECT group contents list shall be initialized to contain:
  - an instance element corresponding to each EXPRESS data type in the working select list that is not an entity, and
  - the subtype group definition for each data type in the working select list that is an entity.
- For each subtype group definitions, that subtype group shall be deleted and replaced with each of the instance elements and subtype groups contained within.
- The previous step shall be repeated until there are no subtype groups in the SELECT group contents list, leaving only a set of instance elements.
- Any duplicate instance element in the list shall be deleted.
- Any ABSTRACT instance elements in the list shall be deleted.

— ***complexEntity-usage***: If any item of the working select list is an instance of an entity data type that may be uncharacterized, the choice group shall contain <xs:element ref="exp:complexEntity"/> Otherwise, the complexEntity element shall be omitted.

NOTE Since every data type in the working select list is a named data type, the qualified names of the instance elements will be derived from the EXPRESS identifiers for the data types as specified in 7.1.3.

EXAMPLE

In EXPRESS:

```
TYPE label = STRING;
END_TYPE;

TYPE length_measure = REAL;
END_TYPE;

TYPE traffic_light = ENUMERATION OF (red, yellow, green);
END_TYPE;

TYPE inner = SELECT (label, length_measure);
END_TYPE;

TYPE s = SELECT (length_measure, traffic_light, pipe_entity, inner);
END_TYPE;

ENTITY pipe_entity;
  name : STRING;
END_ENTITY;
```

```
The SELECT group contents list shall be initialized to:
       Tns:Label-wrapper
       Tns:Length_measure-wrapper
       Tns:Pipe_entity-group
       Tns:Traffic_light-wrapper
```

In XML Schema:

```
<xs:group name="S">
  <xs:choice>
    <xs:element ref="Tns:Label-wrapper"/>
    <xs:element ref="Tns:Length_measure-wrapper"/>
    <xs:element ref="Tns:Pipe_entity"/>
    <xs:element ref="Tns:Traffic_light-wrapper"/>
  </xs:choice>
</xs:group>
```

#### 7.3.4.3 XML data type definition for SELECT data types

When the underlying type of the EXPRESS defined data type is a `SELECT` data type that has a working select list containing more than one data type, the corresponding XML schema data type shall be a `complexType` whose content model is the `group` corresponding to the `SELECT` data type, as specified in 7.3.4.2.

The `complexType` definition shall have the form:

```
<xs:complexType name = "identifier" >
    <xs:group ref="group-name" minOccurs="0" maxOccurs="1" />
    <xs:attribute name="ref" type="xs:IDREF" use="optional" />
    exp:arraySize-usage
</xs:complexType>
```

where

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 7.1.2.

— ***group-name*** is the qualified name of the group corresponding to the `SELECT` data type, as specified in 7.3.4.2.

— ***exp:arraySize-usage***: If any of the instance elements contained in the `SELECT` group (see 7.3.4.2) represent anonymous aggregates, <xs:element ref="`exp:arraySize`" use="optional"/>. Otherwise, the ***exp:arraySize*** element shall be omitted.

NOTE 1 Each instance element in the group corresponding to the `SELECT` data type contains the optional XML attribute `path`, as specified in 7.4.5. The `path` attribute specifies the types in the select list that are instantiated in that value.

NOTE 2 In order to allow accessor elements (see 7.6.3) to refer to a value of the `SELECT` data type (instead of containing it), the `complexType` contains the `ref` attribute, and allows the `choice` group to be omitted. 9.4.3 requires either the ref attribute to be present or the content to contain an instance of the `choice` group.

EXAMPLE Continuing the example in 7.3.4.2 above, the XML `complexType` declaration would be:

```
<xs:complexType name="S">
  <xs:group ref="Tns:S" minOccurs="0" maxOccurs="1"/>
```

 

```
    <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
  </xs:complexType>
```

### 7.3.5 Defined data type underlying type

When the underlying type of the EXPRESS defined data type is another defined data type, the corresponding XML schema data type model shall depend on the XML schema data type corresponding to the fundamental type.

If the XML schema data type corresponding to the fundamental type is a `simpleType`, the XML schema data type model shall be a `simpleType` derived by restriction from the XML schema data type correspoinding to the underlying type. The XML schema data type definition shall have the form:

```
<xs:simpleType name="identifier">
  <xs:restriction base="underlying-type"/>
</xs:simpleType>
```

If the XML schema data type corresponding to the fundamental type is a `complexType`, the XML schema data type model shall be a `complexType` derived by restriction from the XML schema data type corresponding to the underlying type. The XML schema data type definition shall have the form:

```
<xs:complexType name="identifier">
 <contentUsage>
  <xs:restriction base="underlying-type">
     (content model for the fundamental type)
  </xs:restriction>
 </contentUsage>
</xs:complexType>
```

where, in either case:

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 7.1.2.

— ***underlying-type*** is the qualified name of the XML schema data type that corresponds to the underlying-type.

— ***contentUsage:*** When the `complexType` corresponding to the underlying type has complex content, ***contentUsage*** shall be `xs:complexContent`. Otherwise, it shall be `xs:simpleContent`.

— ***(content model for the fundamental type):*** When the `complexType` corresponding to the underlying type has complex content, this is the XML particle that describes the content of the XML data type that corresponds to the fundamental type of the defined data type: the `sequence` particle specified in 7.2.2, if the fundamental type is an aggregation data type, or the `group` specified in 7.3.4.2, if the fundamental type is a `SELECT` type.

NOTE The content model need not include the XML attributes of the fundamental type.

EXAMPLE 1

In EXPRESS:

```
TYPE RatioMeasure = REAL;
END_TYPE;
```

```
TYPE NormalisedRatioMeasure = RatioMeasure;
  WHERE
    WR1 : {0.0 <= SELF <= 1.0};
END_TYPE;
```

In XML Schema:

```
<xs:simpleType name="Ratiomeasure">
  <xs:restriction base="xs:double"/>
</xs:simpleType>

<xs:simpleType name="Normalisedratiomeasure">
  <xs:restriction base="Tns:Ratiomeasure"/>
</xs:simpleType>
```

EXAMPLE 2

Using the example in 7.3.4.2 above:

In EXPRESS:

```
TYPE s = SELECT (length_measure, traffic_light, pipe_entity, inner);
END_TYPE;

TYPE s2 = s;
END_TYPE;
```

In XML Schema:

```
<xs:complexType name="S">
  <xs:group ref="Tns:S" minOccurs="0" maxOccurs="1"/>
  <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
</xs:complexType>

<xs:complexType name="S2">
  <xs:complexContent>
    <xs:restriction base="Tns:S">
      <xs:group ref="Tns:S" minOccurs="0" maxOccurs="1"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

## 7.4 Instance elements corresponding to EXPRESS data types

An *instance element* is an XML element that is named for an EXPRESS data type and represents a value of that EXPRESS data type.

A *by-value instance element* is an instance element whose content is the data value.

A *by-reference instance element* is an instance element that has no content but contains a reference to a by-value instance element whose content is the data value.

NOTE 1 Most instance elements for data types that are not entity data types will be by-value instance elements.

The instance element corresponding to an EXPRESS entity data type shall be declared as specified in 7.5.

 

The instance element corresponding to an EXPRESS data type that is neither an entity data type nor a select data type shall be declared when any of the following applies:

— the EXPRESS data type appears as the base-type of an aggregation data type that uses the sequence-of-elements representation (see 7.2.2);

— the EXPRESS data type appears in the working select list of a SELECT data type (see 7.3.4);

— the value of the EXPRESS data type can appear by-reference (see 7.4.6).

Otherwise, the instance element may be declared in the derived XML schema, but it is not required.

Exceptions: The instance element shall not be declared in the derived XML schema when it is declared in the Base XML Schema, as indicated in 7.4.1. No instance element shall be declared for a defined data type whose underlying type is a SELECT data type.

NOTE 2 For a defined data type whose underlying type is a SELECT data type, the XML group declared as specified in 7.3.4.2 is the equivalent of the instance element. But a defined data type whose *underlying* type is a defined data type and whose *fundamental* type is a SELECT data type (that is, a specialization of a SELECT data type), *does* have an instance element. The above exception does not apply to such data types.

The instance element declaration shall be as specified according to the EXPRESS data type in 7.4.1 through 7.4.4.

NOTE 3 The typical usage for a value of an EXPRESS data type that is not an entity data type is as the value of an entity attribute. In such cases, the EXPRESS data type value is represented as the content of an XML element or an XML attribute that represents the EXPRESS entity attribute (see 7.6). However, a value of an EXPRESS data type may also appear in an instance document as a component of an aggregation value, or as a value of a SELECT data type. In these contexts, the value may be represented by an instance element.

### 7.4.1 Instance elements for simple data types

The Base XML Schema (see Annex C) declares an instance element for each EXPRESS simple data type. The data type of the instance element is the XML Schema data type that corresponds to the EXPRESS data type, and the instance element adds the instanceAttributes attribute group (see 7.4.5).

In some cases, additional instance elements for simple types may be declared in the derived XML schema. See below.

#### 7.4.1.1 BINARY data type

The instance element corresponding to the EXPRESS BINARY data type shall be the exp:hexBinary-wrapper element. Exception: If the EXPRESS BINARY data type specifies a maximum or FIXED length, the instance element shall be mapped as specified in 7.4.1.1.1.

The exp:hexBinary-wrapper instance element is declared in the Base XML Schema as follows:

```
<xs:element name="hexBinary-wrapper" nillable="true" >
 <xs:complexType>
    <xs:simpleContent>
       <xs:extension base="exp:hexBinary">
          <xs:attributeGroup ref="exp:instanceAttributes"/>
       </xs:extension>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

#### 7.4.1.1.1 Constrained BINARY instance element

When required, the instance element corresponding to an EXPRESS BINARY data type for which a maximum or FIXED length is specified (see 7.2.1.1.1) shall be declared in the derived XML schema as follows:

```
<xs:element name="Binary.minBits.bits-wrapper"
         nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="binary-type">
        <xs:attributeGroup ref="exp:instanceAttributes"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

where:

— ***binary-type*** is the qualified name of the XML schema data type declared for the constrained BINARY data type as specified in 7.2.1.1.1;

— ***bits*** is the fixed or maximum length in bits specified for the EXPRESS data type;

— ***minBits*** shall be set to ***bits*** if the length specified for the EXPRESS BINARY data type is declared FIXED; otherwise, it shall be 0.

EXAMPLE

In EXPRESS:

```
ENTITY access_authorization;
  title : STRING;
  codes : LIST OF BINARY(12)FIXED;
END_ENTITY;
```

That form requires the instance element corresponding to the LIST base-type BINARY(12)FIXED to be declared:

```
<xs:element name="Binary.12.12-wrapper" nillable="true" >
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="Tns:Binary.12.12">
        <xs:attributeGroup ref="exp:instanceAttributes"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

### 7.4.1.2 BOOLEAN data type

The instance element corresponding to the EXPRESS BOOLEAN data type is the exp:boolean-wrapper element. The exp:boolean-wrapper element is declared in the Base XML Schema as follows:

```
<xs:element name="boolean-wrapper" nillable="true">
 <xs:complexType>
    <xs:simpleContent>
       <xs:extension base="xs:boolean">
          <xs:attributeGroup ref="exp:instanceAttributes"/>
       </xs:extension>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

#### 7.4.1.3 INTEGER data type

The instance element corresponding to the EXPRESS `INTEGER` data type is the `exp:long-wrapper` element. The instance element is declared in the Base XML Schema, as follows:

```
<xs:element name="long-wrapper" nillable="true" >
 <xs:complexType>
    <xs:simpleContent>
     <xs:extension base="xs:long">
           <xs:attributeGroup ref="exp:instanceAttributes"/>
     </xs:extension>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

#### 7.4.1.4 LOGICAL data type

The instance element corresponding to the EXPRESS `LOGICAL` data type is the `exp:logical-wrapper` element. The `exp:logical-wrapper` element is declared in the Base XML Schema as follows:

```
<xs:element name="logical-wrapper" nillable="true">
 <xs:complexType>
    <xs:simpleContent>
       <xs:extension base="exp:logical">
          <xs:attributeGroup ref="exp:instanceAttributes"/>
       </xs:extension>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

#### 7.4.1.5 NUMBER data type

The instance element corresponding to the EXPRESS `NUMBER` data type is the `exp:decimal-wrapper` element. The instance element is declared in the Base XML Schema as follows:

```
<xs:element name="decimal-wrapper" nillable="true" >
 <xs:complexType>
    <xs:simpleContent>
       <xs:extension base="xs:decimal">
          <xs:attributeGroup ref="exp:instanceAttributes"/>
       </xs:extension>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

#### 7.4.1.6 REAL data types

The instance element corresponding to the EXPRESS `REAL` data type is the `exp:double-wrapper` element. The instance element is declared in the Base XML Schema as follows:

```
<xs:element name="double-wrapper" nillable="true" >
 <xs:complexType>
    <xs:simpleContent>
       <xs:extension base="xs:double">
          <xs:attributeGroup ref="exp:instanceAttributes"/>
       </xs:extension>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

#### 7.4.1.7 STRING data type

The instance element corresponding to the EXPRESS `STRING` data type is the `exp:string-wrapper` element. Exception: If the EXPRESS `STRING` data type specifies a maximum or `FIXED` length, the corresponding instance element shall be as specified in 7.4.1.7.1

The `exp:string-wrapper` element is declared in the Base XML Schema as follows:

```
<xs:element name="string-wrapper" nillable="true" >
 <xs:complexType>
    <xs:simpleContent>
       <xs:extension base="xs:normalizedString">
          <xs:attributeGroup ref="exp:instanceAttributes"/>
       </xs:extension>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

##### 7.4.1.7.1 Constrained `STRING` instance element

When required, the instance element corresponding to an EXPRESS `STRING` data for which a maximum or `FIXED` length is specified (see 7.2.1.7.1), shall be declared in the derived XML schema as follows:

```
<xs:element name="stringtype-wrapper" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="Tns:stringtype">
        <xs:attributeGroup ref="exp:instanceAttributes"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

where:

— ***stringtype*** is the local part of the name of the XML schema datatype that corresponds to the constrained `STRING` data type as specified in 7.2.1.7.1;

 

— ***Tns:stringtype*** is the qualified name of the XML schema datatype that corresponds to the constrained `STRING` data type.

EXAMPLE

In EXPRESS:

```
ENTITY user;
  title : STRING;
  userids : SET OF STRING (8);
END_ENTITY;
```

That form requires the instance element corresponding to the `SET` base-type `STRING (8)` to be declared:

```
<xs:element name="String.0.8-wrapper" nillable="true" >
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="Tns:String.0.8-wrapper">
        <xs:attributeGroup ref="exp:instanceAttributes"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

### 7.4.2 Instance elements for anonymous aggregation data types

When an instance element representing an anonymous aggregation data type is required, as specified in 7.4 above, it shall be declared as specified in this subclause.

The instance element for an anonymous aggregation type shall correspond to *all* anonymous aggregation data types in the context schema that have the same base-type, regardless of the kind of aggregate (`SET`, `LIST`, `BAG` or `ARRAY`).

Because the instance element represents a value of any aggregation data type with the same base-types, it shall include all of the `ref`, `arraySize`, `itemType` and `cType` attributes defined in 7.2.2.1.

Because the instance element may be referenceable (see 7.4.6) or represent a "row" in a higher-level aggregate value, the instance element shall include the `instanceAttributes` group (see 7.4.5).

The XML schema data type in the instance element declaration shall depend on the XML schema data type corresponding to the aggregation data type, as specified in 7.2.2, but modified as follows:

— If the XML data type corresponding to the aggregation data type has the list-of-values form, as specified in 7.2.2, the instance element declaration shall be as specified in 7.4.2.1 below.

— If the XML data type corresponding to the aggregation data type has the sequence of elements form, as specified in 7.2.2, the instance element declaration shall be as specified in 7.4.2.2 below.

— If the XML data type corresponding to the aggregation data type uses the multi-dimensional-array form, as specified in 7.2.2.4, the instance element declaration shall be as specified in 7.4.2.2 for the sequence-of-elements form, except that the base-type shall be considered to be the deepest underlying type of the multi-dimensional array structure, for all purposes.

NOTE 1 As a consequence the sequence-of-elements instance element represents all aggregations of the (deepest) base-type instance element, regardless of the number of dimensions.

NOTE 2 As a consequence the list-of-values instance element represents all serialized aggregations of values of the (deepest) base-type, regardless of the number of dimensions.

#### 7.4.2.1 Instance elements for list-of-values forms

If the XML schema data type corresponding to the aggregation data type uses the list-of-values form (see 7.2.2.3), the content model for the instance element shall be a simple list, extended by the XML instance attributes and the XML aggregate attributes.

The instance element declaration shall have the form:

```
<xs:element name="element-name" nillable="true" >
 <xs:complexType>
    <xs:simpleContent>
       <xs:extension base="unrestricted-type">
          <xs:attributeGroup ref="exp:instanceAttributes" />
       </xs:extension>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

where:

— ***element-name***, the XML name of the instance element, shall have the form: `Seq-`***base***, where ***base*** is the local part of the name of the XML Schema data type that corresponds, as specified in 7.2, to the base-type, of the aggregation data type.

— ***unrestricted-type*** shall be the qualified name of the unrestricted XML data type that supports all aggregations of the base-type, of the aggregation data type, as specified in 7.2.2.3.2.

NOTE 1 The instance element corresponding to the base-type is never used in this representation.

NOTE 2 There are many required behaviours of the XML attributes of this instance element that are declared to be `use="optional"`. These behaviours are specified in 9.8.

EXAMPLE

For the EXPRESS anonymous data type:

```
ARRAY [1:3] OF REAL
```

The instance element declaration will be:

```
<xs:element name="Seq-double" nillable="true" >
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="Tns:Seq-double">
        <xs:attributeGroup ref="exp:instanceAttributes" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

#### 7.4.2.2 Instance elements for all sequence-of-elements forms

If the XML schema data type corresponding to the aggregation data type uses the sequence-of-elements form (see 7.2.2.2), or the multi-dimensional array form (see 7.2.2.4), the instance element declaration shall have the form:

```
<xs:element name="element-name" nillable="true" >
 <xs:complexType>
        <xs:sequence>
            <element-or-group ref="base-instance"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
        <xs:attribute ref="exp:itemType"   fixed="base-instance"/>
        <xs:attribute ref="exp:arraySize"  use="optional"/>
        <xs:attribute ref="exp:cType" default="set"/>
        <xs:attributeGroup ref="exp:instanceAttributes"/>
 </xs:complexType>
</xs:element>
```

where:

— ***element-name***, the XML name of the instance element, shall have the form: `Seq-`***base***, where ***base*** is the local part of the ***base-instance*** name (see below).

— ***base-instance*** : Let the term "base-type" refer to the EXPRESS base-type of the aggregation data type for the sequence-of-elements form, or to the deepest underlying type for the multi-dimensional array form. Then if base-type is an entity data type, ***base-instance*** shall be the qualified name of the corresponding complexEntity group (see 7.5.6.2). If base-type is a SELECT data type, ***base-instance*** shall be the qualified name of the corresponding group (see 7.3.4.2). Otherwise, ***base-instance*** shall be the qualified name of the corresponding instance element (see 7.4).

— ***element-or-group*** shall be `xs:element`, if ***base-instance*** is an element, or `xs:group`, if ***base-instance*** is a group.

NOTE 1 The XML data type that corresponds to the aggregation data type, as specified in 7.2.2, is not actually used in this declaration.

NOTE 2 Although the `ref`, `cType` and `arraySize` attributes are declared `use="optional"`, clause 9.8 requires them to appear in certain circumstances, in particular when the instance element represents a multi-dimensional aggregate value.

EXAMPLE

The EXPRESS anonymous data type:

```
LIST [1:?] OF ARRAY [1:3] OF REAL
```

The instance element declaration will be:

```
<xs:element name="Seq-double-wrapper" nillable="true">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="exp:double-wrapper" minOccurs="0"
           maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute use="optional" type="xs:IDREF" name="ref"/>
```

```
      <xs:attribute fixed="exp:double-wrapper" ref="exp:itemType"/>
      <xs:attribute ref="exp:arraySize" use="optional"/>
      <xs:attribute ref="exp:cType" default="set"/>
      <xs:attributeGroup ref="exp:instanceAttributes"/>
    </xs:complexType>
  </xs:element>
```

### 7.4.3 Instance elements for defined data types

For each EXPRESS defined data type that appears as the base-type of an aggregation data type or in the select-list of a SELECT data type, the derived XML schema shall contain the declaration for an instance element. Exception: There shall be no instance element that corresponds to a defined data type whose underlying type is a SELECT data type; the corresponding XML schema object is a group (see 7.3.4.2).

Note 1 When the underlying type is a SELECT type and the working select list contains more than one data type, the XML schema object corresponding to the defined data type is a group (see 7.3.4.2). When the working select list contains only one data type, the instance element for the sole choice is used for the defined data type (see 7.3.4).

Note 2 For a defined data type whose fundamental type is a SELECT data type but whose *underlying type* is not, there is an instance element as specified in the subclause.

If the XML schema data type corresponding to the defined data type is a simpleType, the instance element declaration shall specify the data type of the content to be that data type, and the instance element declaration shall also include the instanceAttributes group by reference. That is, the element declaration shall have the form:

```
<xs:element name="element-name" nillable="true" >
 <xs:complexType>
    <xs:simpleContent>
       <xs:extension base="type-name">
          <xs:attributeGroup ref="exp:instanceAttributes" />
       </xs:extension>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

where

— ***type-name*** is the qualified name of the XML schema data type corresponding to the EXPRESS defined data type (see 7.1.3).

— ***local-type-name*** is the local part of ***type-name***.

— ***element-name***: If the fundamental type of the EXPRESS defined data type is BOOLEAN, INTEGER, LOGICAL, REAL, NUMBER, STRING, BINARY, or an ENUMERATION type, ***element-name*** has the form ***local-type-name***-wrapper. Otherwise, ***element-name*** is identical to ***local-type-name***.

If the XML schema data type corresponding to the defined data type is a complexType, the instance element declaration shall specify the data type of the instance element to be an extension of that data type that includes the instanceAttributes group. That is, the element declaration shall have the form:

```
<xs:element name="local-type-name" nillable="true">
```

 

```
 <xs:complexType>
  <xs:complexContent>
    <xs:extension base="type-name">
        <xs:attributeGroup ref="exp:instanceAttributes" />
    </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
```

where ***type-name*** and ***local-type-name*** are as defined above.

EXAMPLE

In EXPRESS:

```
TYPE material_type = ENUMERATION OF (steel, copper, plastic);
END_TYPE;

TYPE list_of_materials = SET [1:?] of material_type;
END_TYPE;
```

In XML Schema:

```
<xs:simpleType name = "Material_type" >
  <xs:restriction base = "xs:string">
       <xs:enumeration value = "steel"/>
       <xs:enumeration value = "copper"/>
       <xs:enumeration value = "plastic"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="Material_type-wrapper" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension type="Tns:Material_type">
        <xs:attributeGroup ref="exp:instanceAttributes"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

### 7.4.4 Instance elements for entity data types

For an EXPRESS entity data type, the corresponding instance element, if any (see 7.5), shall be as specified in 7.5.5.

### 7.4.5 Instance element attributes

All instance elements shall have a `complexType` that includes the `instanceAttributes` XML attribute group. The attribute group is defined in the Base XML Schema as:

```
<xs:attributeGroup name="instanceAttributes">
 <xs:attribute name="id" type="xs:ID" use="optional"/>
 <xs:attribute name="path" type="xs:NMTOKENS" use="optional"/>
 <xs:attribute name="pos" use="optional">
    <xs:simpleType>
       <xs:restriction>
          <xs:simpleType>
                 <xs:list itemType="xs:integer"/>
```

```
            </xs:simpleType>
            <xs:minLength value="1"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:attributeGroup>
```

The `id` attribute specifies a local XML identifier for the instance element. The `id` attribute shall appear only in a by-value instance element (an element that supplies the information content of the EXPRESS value, see 7.4). It enables the instance element to be referenced in a by-reference accessor or instance element elsewhere in the document.

The `path` attribute specifies the set of EXPRESS named data types that are instantiated in the value of the instance element. The `path` attribute shall appear only in an instance element that represents a value of an EXPRESS `SELECT` data type. Its value shall be a space-separated list of XML names, each of which is the local part of the name of the XML data type corresponding to an EXPRESS named data type that is instantiated in the value of the instance element.

The `pos` attribute specifies the position or indices of a given array component value, when certain aggregate representations are used (see 9.8.3 and 9.8.5). The value of the `pos` attribute shall be a space-separated list of the indices of the component in the array value, each expressed as a decimal integer.

### 7.4.6 Referenceable instances

An EXPRESS data type is said to be *referenceable* if any element in the unit of serialization can contain a reference to a value of the EXPRESS data type in lieu of the value itself.

Because there are different defaults for the mapping of different EXPRESS data types, the practical determination that a data type is referenceable varies with the data type.

In the default XML schema, every EXPRESS entity data type is referenceable unless none of its instances can be (part of) the value of any attribute in a unit of serialization conforming to the context schema. In the default XML schema, every non-entity EXPRESS data type is referenceable.

## 7.5 XML Schema definitions and declarations for EXPRESS entity data types

For each EXPRESS entity data type that is declared in or interfaced into the context schema, the corresponding XML schema definitions and declarations shall depend on the EXPRESS entity declaration and on the structure of the associated type graph, as specified in 7.5.1. The derived XML schema shall contain:

— a `complexType` definition for the corresponding XML data type, as specified in 7.5.4,

— a corresponding instance element declaration, as specified in 7.5.5,

— corresponding XML group declarations, as specified in 7.5.6,

— a corresponding single entity value declaration, if required, as specified in 7.5.7,

— corresponding proxy element declarations, as specified in 7.5.8, and

— corresponding XML uniqueness constraints, as specified in 7.5.9.

 

Exception: If the entity data type is declared to be abstract (see 6.3), there shall be corresponding XML group declarations, as specified in 7.5.6, XML uniqueness constraints, as specified in 7.5.9, and possibly single entity value declarations, as specified in 7.5.7, but no corresponding XML data type, instance element, or proxy element shall be declared.

### 7.5.1 Type graph associated with the EXPRESS entity data type

The collection of EXPRESS entity data types that are declared in or interfaced into, the context schema forms a set of "type graphs". Each type graph consists of a set of nodes that represent the entity data types and a set of directed edges that represent the `SUBTYPE OF` relationships. The set of type graphs is formed by the following algorithm:

— (1) If all EXPRESS entity data types have been assigned to some type graph, stop.

— (2) Choose any entity data type not yet assigned to a type graph, and create a new type graph consisting of one node designating that entity data type.

— (3) For the given entity data type, add all its immediate supertypes to the type graph as nodes, eliminating any duplicates. Link the given entity data type to each supertype by a directed edge designated `SUBTYPE OF`.

— (4) For the given entity data type, add all its immediate subtypes to the type graph as nodes, eliminating any duplicates. Link each subtype to the given entity data type by a directed edge designated `SUBTYPE OF`.

— (5) Repeat steps (3) and (4) for each new entity data type added to the type graph (by step (3) or step (4)), until all entity data types in the type graph have been so processed. Then return to step (1).

A node in the type graph is said to be a *root node* if it represents an entity data type that has no supertypes. A node in the type graph is said to be a *leaf node* if it represents an entity data type that has no subtypes in the context schema, and no synthetic subtypes. A type graph is said to be a *tree structure*, if it has exactly one root node, and for each leaf node, it has exactly one path along directed edges from the leaf node to the root node.

A type graph is said to be *associated with* each of the EXPRESS entity data types represented in it. Using the construction algorithm defined above, every EXPRESS entity data type in the context schema is associated with exactly one type graph.

NOTE 1 This subclause does not state any requirements. Rather it defines a concept and a set of terms that are used in stating requirements in other subclauses.

NOTE 2 The "type graph" is conceptual – it represents a collection of all the entity data types in the context schema that are related to each other by subtype/supertype relationships in any way. The set of all "paths" through the type graph that follow the directed edges from a given entity data type to any root includes exactly all of the entity data types from which an instance of the given entity data type inherits attributes. The "type graph" is introduced here in order to simplify the specification of entity data type mappings. A conforming pre-processor or post-processor need not instantiate the type graphs per se.

NOTE 3 A type graph may consist of a single entity data type. This will occur when the entity data type has no modeled subtypes and is not itself a subtype.

NOTE 4 Every type graph will have one or more root nodes and one or more leaf nodes. Not every type graph is a tree structure. A type graph may be a "lattice", having a single root but at least one leaf node with more than one path to the root. A type graph may have multiple roots.

### 7.5.2 Complex entity instances

Every EXPRESS entity instance instantiates one or more entity data types, all of which can be associated with nodes in a single type graph, as defined in 7.5.1. The type graph associated with the instance is the graph consisting of exactly those nodes.

In ISO 10303-11, a *complex entity instance* is an entity instance that is described by more than one EXPRESS entity declaration. That is, a *complex entity instance* is an instance that instantiates more than one entity data type, or equivalently, an instance whose associated subgraph has more than one node.

Each of the entity declarations involved is said to define a *single entity value*, that is, a part of the complex entity instance that contains exactly the set of explicit attributes defined in that entity declaration. And for each single entity value there is one corresponding node in the subgraph associated with the entity instance.

In this Part of ISO 10303, complex entity instances are further classified as either characterized or uncharacterized, as follows:

— The entity instance is said to be *characterized* by an entity data type if that entity data type is instantiated in the instance and it is a subtype of every other entity data type that is instantiated in the instance. A characterized entity instance is called a *single-leaf* instance in ISO 10303-11 – the subgraph has exactly one leaf node.

— The entity instance is said to be *uncharacterized* if there is no entity data type that characterizes it. An uncharacterized entity instance is called a *multi-leaf* instance in ISO 10303-11 – the subgraph has more than one leaf node.

NOTE 1 The leaf nodes of the subgraph corresponding to an entity instance may not be leaf nodes of the type graph.

Each characterized entity instance shall be represented in the unit of serialization as an instance of the characterizing entity data type, as specified in 9.3.

NOTE 2 If the entity instance is characterized, the EXPRESS entity data type corresponding to the leaf node has all the properties needed to represent the entity instance, and the corresponding XML schema element, as defined in this clause, can be used to represent the instance. If the entity instance is uncharacterized, there is no single entity data type in the EXPRESS schema that has all the properties needed to represent the entity instance. This situation can arise when there are ANDOR relationships among subtypes.

Uncharacterized entity instance can be represented using the `exp:complexEntity` instance element (see 7.5.3.3), as specified in 9.3.

### 7.5.3 Base XML data types and elements for EXPRESS entity data types

The derived XML schema shall include by reference an XML data type representing the EXPRESS `GENERIC_ENTITY` data type. The generic entity data type represents the supertype of all EXPRESS entity data types in the context schema. The derived XML schema shall also include by reference the corresponding XML element, whose purpose is to be the root of a substitution group that includes all entity instance elements.

The generic entity data type shall be `exp:Entity` (see 7.5.3.1), and no additional definition is needed in the derived XML schema. The generic entity instance element shall be `exp:Entity`, which is declared in the Base XML Schema (see Annex C) as:

```
<xs:element name="Entity" type="exp:Entity" abstract="true" />
```

 

The derived XML schema may also declare a *root entity instance element*, representing the supertype of all independently instantiable EXPRESS entity data types in the context schema. The root entity data type shall be `exp:Entity` (see 7.5.3.1), and the root entity instance element shall be the same as the generic entity instance element shown above.

NOTE In EXPRESS, independently instantiable entity data types are those interfaced via `USE` instead of `REFERENCE`.

These definitions and declarations are specified in this subclause.

### 7.5.3.1 Base XML data type for entity data types – `exp:Entity`

The XML schema data types corresponding to EXPRESS entity data types are extensions of a `complexType` named `exp:Entity`, which is defined in the Base XML Schema (see Annex C) as follows:

```
<xs:complexType name = "Entity" abstract="true">
 <xs:attribute name = "href" type = "xs:anyURI" use = "optional"/>
 <xs:attribute name = "ref" type = "xs:IDREF" use = "optional"/>
 <xs:attribute name = "proxy" type = "xs:IDREF" use = "optional"/>
 <xs:attribute name = "edo" type = "xs:anyURI" use = "optional"/>
 <xs:attributeGroup ref="exp:instanceAttributes"/>
</xs:complexType>
```

The `id` attribute (from `exp:instanceAttributes)` specifies a local XML identifier for the entity instance. The `id` attribute shall appear only in a by-value instance element (that is,. an element that supplies the information content of the EXPRESS entity instance, see 9.3.1). It enables the instance element to be referenced in a by-reference instance element or a by-reference accessor element elsewhere in the document.

The `ref` attribute specifies a local XML identifier that identifies a by-value instance element somewhere else in the document. The `ref` attribute shall appear only in a by-reference instance element (that is, in an element that contains a reference to a by-value instance element, see 9.3.3), rather than by supplying the information content.

The `proxy` attribute specifies the local identifier for a proxy element, which can be used to obtain a by-value instance element from some external resource. The proxy attribute shall appear only in a by-proxy instance element (that is, an element that refers to an external by-value instance element by proxy, see 9.3.2).

The `href` attribute specifies an access path for a by-value instance element in some external resource. The value of the `href` attribute shall have the form: ***uri*#*id***, where ***uri*** is a URI conforming to IETF RFC 2396 that locates the resource, and the fragment identifier ***id*** is an XML identifier local to that resource that identifies a by-value instance element of the same type (see *XML Pointer Framework (Xpointer) Version 1, Section 3.2*). The `href` attribute shall appear only in by-path instance element (that is, an element that refers to an external by-value instance element directly, see 9.3.2).

The `edo` attribute specifies a universal identifier for the entity instance as a resource. The purpose of the `edo` attribute is to uniquely identify the entity instance without regard to the location and form of encodings of it. This attribute may appear in a by-value instance element or in any form of external reference; it shall not appear in a by-reference instance element.

Because an instance element may represent an element of an aggregate value, the `complexType` also includes the `pos` attribute(see 7.4.5).

NOTE The detailed rules for the use of by-value, by-reference, and external instance elements appear in Clause 9.3.

#### 7.5.3.2 Base XML data type for proxy elements – `exp:edokey`

An *external instance element* is an entity instance element that is contained in an external resource. A *proxy element* is an element that appears in the unit of serialization and contains the globally-unique, persistent identifiers for the external instance element and, thus, the information needed to access the external instance.

The derived XML schema includes declarations for proxy elements that correspond to external entity instance elements (see 7.5.8). The XML data types for the proxy elements are extensions of a complexType named `exp:edokey`. The purpose of the `exp:edokey` data type is to provide the basic data structure for proxy elements. The data type of a proxy element is derived from `exp:edokey` and depends on the data type of the external entity instance.

For convenience, the proxy elements are declared to belong to a common substitution group based on an abstract element also called `exp:edokey`. The `exp:edokey` data type and element are defined in the Base XML Schema (see Annex C) as follows:

```
<xs:complexType name = "edokey" abstract="true">
    <xs:attribute name="id" type="xs:ID" use="required" />
    <xs:attribute name="edo"  type="xs:anyURI" use="optional"/>
    <xs:attribute ref="exp:authority"  use = "required"/>
    <!--  key attributes appear in subtypes  -->
</xs:complexType>

<xs:attribute name="authority" type="xs:anyURI"/>

<xs:attribute name="edokeyType" type="xs:QName" />

<xs:element name="edokey" type="exp:edokey" abstract="true" />
```

The interpretation of the XML attributes is as follows:

The `id` attribute provides the local XML identifier for the proxy element.

The `edo` attribute, if present, provides a global URI, conforming to IETF RFC 2396, by which the external entity instance is known, as distinct from the location of a resource that contains a corresponding entity instance element. Its primary purpose is to allow the post-processor to recognize the referenced entity instance "by name" and to use any available copy.

The `exp:authority` attribute specifies the owner of the external instance element. The value of the `exp:authority` attribute shall be a URI that identifies the resource that contains the referenced external instance element. The URI shall conform to IETF RFC 2396.

The key attributes appear only in the subtypes of `exp:edokey` that are defined for individual entity data types (see 7.5.8).

The `edokeyType` attribute specifies the `complexType` of the proxy element that corresponds to the `exp:edokey` data type.

#### 7.5.3.3 Instance element for complex entity instances – `exp:complexEntity`

The `exp:complexEntity` instance element is used to represent uncharacterized complex entity instances (see 7.5.2 above) in a unit of serialization. The `exp:complexEntity` element is declared in the Base XML Schema as follows:

```
<xs:element name="complexEntity" substitutionGroup="exp:Entity"
 nillable="true">
 <xs:complexType>
    <xs:complexContent>
       <xs:extension base="exp:Entity">
          <xs:sequence>
             <xs:any namespace="##any"
                    processContents="skip" maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="entities" type="xs:NMTOKENS"
             use="optional"/>
       </xs:extension>
    </xs:complexContent>
 </xs:complexType>
</xs:element>
```

The `exp:complexEntity` element shall contain the single entity value elements for each of the entity data types instantiated in the complex entity instance, as specified in 7.5.7.

The value of the `entities` XML attribute shall be a list of the EXPRESS identifiers for the entity data types corresponding to the single entity values. (See 7.5.7).

#### 7.5.3.4 Base XML data type and element for single entity values – `exp:Single-Entity`

The `exp:Single-Entity` element is the base XML element for the substitution group that represents single entity value elements (see 7.5.7) in a unit of serialization. The `exp:Single-Entity` data type is the base XML data type for single entity value elements. They are declared in the Base XML Schema as follows:

```
<xs:element name="Single-Entity" type="exp:Single-Entity"
       abstract="true"/>

<xs:complexType name="Single-Entity">
</xs:complexType>
```

The XML single entity data type corresponding to a given EXPRESS data type extends this data type to include the EXPRESS attributes declared in the corresponding EXPRESS entity declaration. See 7.5.7.

### 7.5.4 XML data type definitions for entity data types

The `complexType` definition for the EXPRESS entity data type shall be independent of, and appear outside the content of, all XML `complexType` definitions and `element` declarations. That is, the `complexType` shall belong to the schema namespace.

The `complexType` corresponding to the EXPRESS entity data type shall be an extension of the `exp:Entity` data type. The extension shall contain declarations for the accessor elements and accessor attributes corresponding to the EXPRESS attributes of the entity data type, as specified below.

NOTE 1 The accessor elements are declared within an `<all>` group to maximize the order independence and extensibility of the EXPRESS attributes within any instance document.

That is, the `complexType` declaration shall have the form:

```
<xs:complexType name="identifier" abstract="abstract">
 <xs:complexContent>
    <xs:extension base="exp:Entity">
       <xs:all>
          (accessor element declarations)
       </xs:all>
    </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

where

— ***identifier*** is derived from the EXPRESS identifier for the entity data type as specified in 7.1.2.

— ***abstract*** shall be `"true"` if the entity data type is declared to be "abstract" in EXPRESS (see 6.3), otherwise it shall be `"false"` and may be omitted.

— ***accessor element declarations*** are as specified below.

The `extension` group shall contain one accessor declaration as specified in 7.6 for each EXPRESS attribute that appears in the entity type declaration. The accessor element declarations, if any, shall be contained within an `<all>` particle.

In the type graph associated with the entity data type (see 7.5.1), there will be one or more paths between the node corresponding to the entity data type and one or more root nodes. For every node that appears on *any* such path, the `extension` group shall also contain one accessor declaration as specified in 7.6 for each EXPRESS attribute that appears in the entity type declaration for the entity data type that corresponds to that node. The set of attributes corresponding to a given node shall be mapped only once, regardless of the number of paths that node appears on.

NOTE 2 The above provision implies that every "inherited" attribute, using the EXPRESS notion of inheritance, will have one corresponding accessor declaration in the content model of the `complexType`.

NOTE 3 Per 7.6.2, some attributes of the EXPRESS entity data type may not be mapped to accessors.

If an EXPRESS attribute is redeclared on any path to any root node, the accessor declaration shall correspond to the attribute as redeclared, not to the attribute as originally declared. If an EXPRESS attribute is redeclared more than once on a given path to the entity data type whose declaration contained the original EXPRESS attribute, only the first redeclaration encountered on that path shall be used. If an EXPRESS attribute is not redeclared identically on every path to the entity data type whose declaration contained the original EXPRESS attribute, the declaration to be used shall be implementation defined. In this last case, the preprocessor shall either synthesize a type declaration for the attribute that combines the constraint in the multiple redeclarations, or use the original attribute declaration.

EXAMPLE

In EXPRESS:

```
ENTITY named_unit;
    dimensions : dimensional_exponents;
    unit_type  : unit_enum;
```

 

```
END_ENTITY;

ENTITY si_unit
  SUBTYPE OF(named_unit);
    prefix     : OPTIONAL si_prefix;
    name       : si_unit_name;
  DERIVE
    SELF\named_unit.dimensions : dimensional_exponents :=
       dimensions_for_si_unit(name);
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Named_unit">
   <xs:complexContent>
      <xs:extension base="exp:Entity">
        <xs:all>
          <xs:element name="Dimensions">
            <xs:complexType>
              <xs:sequence>
                <xs:group ref =
                     "Tns:Dimensional_exponents-complexEntity-group"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="Unit_type" type="Tns:Unit_enum"/>
        </xs:all>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

<xs:complexType name="Si_unit">
   <xs:complexContent>
      <xs:extension base="exp:Entity">
        <xs:all>
          <xs:element name="Dimensions">
            <xs:complexType>
              <xs:sequence>
                <xs:group ref =
                     "Tns:Dimensional_exponents-complexEntity-group"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="Unit_type" type="Tns:Unit_enum" />
          <xs:element name="Prefix" type="Tns:Si_prefix"
            minOccurs="0" nillable="true"/>
          <xs:element name="Name" type="Tns:Si_unit_name" />
        </xs:all>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

### 7.5.5 Instance elements corresponding to entity data types

An EXPRESS entity instance may be represented in the exchange document in several forms (see 9.3). A single `element` declaration shall be used to declare the XML element type used for all forms of entity instance element that correspond to the EXPRESS entity data type.

NOTE As stated in 7.5 above, there will be no instance element for an abstract entity data type

The instance element declaration shall be independent of, and appear outside the content of, all XML `complexType` definitions and `element` declarations. That is, the instance element type shall belong to the schema namespace.

The `element` declaration for the entity instance element shall have the form:

```
<xs:element name="identifier"
     type="qualified-type"
     nillable="true"
     block="blockattribute"
     substitutionGroup="exp:Entity"/>
```

where:

— ***identifier*** is derived from the EXPRESS identifier for the entity data type as specified in 7.1.2.

— ***qualified-type*** is the qualified name for the XML schema data type that corresponds to the entity data type as specified in 7.5.

— ***blockattribute*** is `'extension restriction'` for entity instance elements and `'#all'` for non-entity instance elements.

EXAMPLE

In EXPRESS:

```
ENTITY named_unit;
-- all attributes omitted
END_ENTITY;
```

In XML Schema:

```
<xs:element name="Named_unit" type="Tns:Named_unit" nillable="true"
  block="extension restriction" substitutionGroup="exp:Entity"/>
```

### 7.5.6 XML groups corresponding to entity data types

For every entity data type in the context schema, the XML schema shall contain XML declarations for two groups that correspond to the entity data type: a subtypes `group` as specified in 7.5.6.1, and a complexEntity group as specified in 7.5.6.2.

#### 7.5.6.1 subtypes group

The subtypes group shall represent a `choice` among the XML elements that may be used to represent instances of the entity data type.

The group shall contain one element declaration referring to the corresponding entity instance element (see 7.5.5) for the entity data type itself. Exception: If the entity data type is declared abstract (see 6.3), the (possibly non-existent) instance element for the entity data type itself shall not appear in the group.

If external representation of instances of the entity data type is permitted, the group shall contain one element declaration referring to the corresponding proxy element (see 7.5.8) for the entity data type.

 

The <choice> group shall also contain one subtype group declaration, referring to the corresponding choice group, for each immediate subtype of the entity data type that appears in the associated type graph.

Figure 1 - illustrates a choice group. A's group consists of A, B and C.

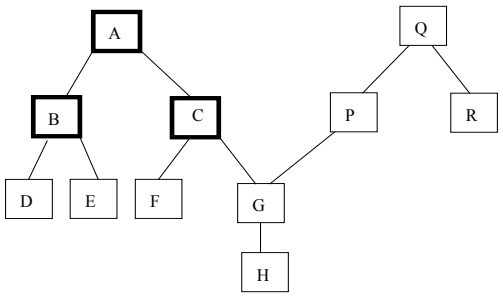


**Figure 1 - Choice group**

NOTE The specified nodes represent all subtypes of the entity data type that are declared in, or interfaced into, the context schema.

The group declaration shall have the form:

```
<xs:group name="identifier-group">
 <xs:choice>
    <xs:element ref="Tns:identifier" />
    <xs:element ref="Tns:identifier-proxy" />
    <xs:group ref="Tns:subtype1-group" />
        ...
    <xs:group ref="Tns:subtypen-group " />
 </xs:choice>
</xs:group>
```

where:

— ***Identifier***-group: ***Identifier*** is the local part of the XML name for the instance element corresponding to the entity data type as specified in 7.5.5. '-group' is a constant literal suffix.

— ***Tns:identifier*** is the qualified name for the instance element corresponding to the entity data type as specified in 7.5.5. '-proxy' is a constant literal suffix.

— ***Tns:subtype***$_1$-group ... ***Tns:subtype***$_n$-group are the qualified names for the choice groups that correspond to subtypes of the entity data type, as specified above.

EXAMPLE

In EXPRESS:

```
ENTITY named_unit;
END_ENTITY;

ENTITY si_unit
  SUBTYPE OF(named_unit);
END_ENTITY;
```

In XML Schema, extending the Example in 7.5.4:

```
<xs:complexType name="Named_unit" >
</xs:complexType>

<xs:complexType name="Si_unit" >
</xs:complexType>
```

The group declarations are:

```
<xs:group name="Named_unit-group">
  <xs:choice>
    <xs:element ref="Tns:Named_unit" />
    <xs:element ref="Tns:Named_unit-proxy" />
    <xs:group ref="Tns:Si_unit-group " />
  </xs:choice>
</xs:group>

<xs:group name="Si_unit-group">
  <xs:choice>
    <xs:element ref="Tns:Si_unit" />
    <xs:element ref="Tns:Si_unit-proxy" />
  </xs:choice>
</xs:group>
```

If external references are not permitted, the proxy element reference is omitted from both groups.

#### 7.5.6.2 complexEntity group

For every entity data type in the context schema, the XML schema shall contain an XML declaration for a complexEntity `group` that corresponds to the entity data type. The complexEntity group shall have a choice group that consists of the subtypes group specified in 7.5.6.1 above. Additionally, if some instances of the entity data type may be uncharacterized, the complexEntity group shall contain one element declaration referring to the `exp:complexEntity` element.

The complexEntity group declaration shall have the form:

```
<xs:group name="identifier-complexEntity-group">
 <xs:choice>
    <xs:group ref="Tns:identifier-group"/>
    complexEntity-usage
 </xs:choice>
</xs:group>
```

where:

— ***Identifier***`-complexEntity-group`: ***Identifier*** is the local part of the XML name for the instance element corresponding to the entity data type as specified in 7.5.5. '`-complexEntity-group`' is a constant literal suffix.

— ***Tns:identifier*** is as defined in 7.5.6.1.

— ***complexEntity-usage***: If any instance of the entity data type may be uncharacterized, the choice group shall contain <xs:element ref="`exp:complexEntity`"/> Otherwise, the complexEntity element shall be omitted.

EXAMPLE (continuing the example from 7.5.6.1 above):

 

In EXPRESS:

```
ENTITY si_unit_2
  SUBTYPE OF(named_unit);
END_ENTITY;
```

```
<xs:group name="Si_unit_2-group">
   <xs:choice>
      <xs:element ref="Tns:Si_unit" />
      <xs:element ref="Tns:Si_unit-proxy" />
   </xs:choice>
</xs:group>


<xs:group name="Si_unit_2-complexEntity-group">
   <xs:choice>
      <xs:group ref="Tns:Si_unit_2-group" />
      <xs:element ref="exp:complexEntity" />
   </xs:choice>
</xs:group>

<xs:group name="Si_unit-complexEntity-group">
   <xs:choice>
      <xs:group ref="Tns:Si_unit-group" />
      <xs:element ref="exp:complexEntity" />
   </xs:choice>
</xs:group>
```

If `si_unit` is the only declared subtype of `named_unit`, and it has no declared subtypes, neither `si_unit` nor `named_unit` could have an uncharacterized instance, and the reference to exp:`complexEntity` would not be present.

### 7.5.7 Single entity value elements corresponding to entity data types

For each EXPRESS entity data type that is a subtype of another EXPRESS entity data type and may appear in the type graph of an uncharacterized entity instance in the unit of serialization (see 7.5.2), the derived EXPRESS schema shall contain a data type definition and an element declaration for the corresponding single entity value.

Exceptions:

— The XML declarations specified in this subclause shall not appear for entity data types that are *roots* of their associated type graphs (see 7.5.1).

— The XML declarations specified in this subclause should not appear if all valid instances of the entity data type in any corresponding unit of serialization are characterized either by entity data types in the context schema or by synthetic entity data types (see 7.5).

— The XML declarations specified in this subclause are permitted, but not required, for entity data types that have no uncharacterized `ANDOR` relationships with other entity data types in the type graph. In making this determination, the schema generator may consider the entire set of declarations in the context schema, including entity declarations, supertype clauses, subtype constraints, and other local and global rules.

NOTE This last determination requires a complex algorithm and perhaps additional knowledge. The schema generator is not required to perform such an algorithm or to apply additional knowledge. When in doubt, it should generate the declarations specified in the subclause.

The XML data type definition shall have the form:

```
<xs:complexType name="identifier-value">
    <xs:complexContent>
    <xs:extension base="exp:Single-Entity">
        <xs:all>
            (accessor element declarations)
        </xs:all>
    </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

and the XML element declaration shall have the form:

```
<xs:element name="identifier-value" type="identifier-value"
    substitutionGroup="exp:Single-Entity"/>
```

where

— ***identifier*** is the XML name derived from the EXPRESS identifier for the entity data type as specified in 7.1.2.

— ***(accessor element declarations)***: For each EXPRESS attribute that appears in the entity declaration, the `extension` shall contain one accessor declaration as specified in 7.6. The accessor element declarations, if any, shall be contained within an `<all>` particle. Exception: There shall be no accessor corresponding to an EXPRESS attribute that is redeclared in the entity declaration.

EXAMPLE

In EXPRESS:

```
ENTITY named_unit;
    dimensions : dimensional_exponents;
    unit_type  : unit_enum;
END_ENTITY;

ENTITY si_unit
  SUBTYPE OF(named_unit);
    prefix     : OPTIONAL si_prefix;
    name       : si_unit_name;
  DERIVE
    SELF\named_unit.dimensions : dimensional_exponents :=
       dimensions_for_si_unit(name);
END_ENTITY;
```

Entity `named_unit` is a root of its type graph and has no corresponding single entity declarations.

Assuming entity `si_unit` may have an `ANDOR` relationship with other subtypes of `named_unit`, the XML Schema declarations would be:

```
<xs:complexType name="Si_unit-value">
  <xs:complexContent>
    <xs:extension base="exp:Single-Entity">
      <xs:all>
        <xs:element name="Prefix" type="Tns:Si_prefix"
          minOccurs="0" nillable="true"/>
        <xs:element name="Name" type="Tns:Si_unit_name" />
      </xs:all>
    </xs:extension>
```

 

```
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="Si_unit-value" type="Si_unit-value"
       substitutionGroup="exp:Single-Entity" />
```

### 7.5.8 Proxy elements corresponding to entity data types

The XML data type of the proxy element shall be an extension of the `exp:edokey` data type (see 7.5.3.2), consisting of a <sequence> group that contains the accessor elements corresponding to those EXPRESS attributes of the entity data type that form a unique key for the entity instance within the referenced resource.

The `complexType` and element declarations shall have the form:

```
<xs:complexType name="identifier-proxy">
 <xs:complexContent>
  <xs:extension base="exp:edokey">
     <xs:sequence>
         ( key attribute declarations )
     </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="identifier-proxy" type="Tns:identifier-proxy "
substitutionGroup="exp:edokey"/>
```

where:

— ***identifier*** the XML name for the corresponding entity instance element, as specified in 7.5.5. The name of the proxy element and data type shall be the concatenation of this XML name and the characters '-proxy.

— ***Tns:identifier***-proxy is the fully qualified name of the data type of the proxy element.

— ***(key attribute declarations)***: declarations for the accessor elements, as specified in 7.6, that correspond to those EXPRESS attributes of the entity data type that form a unique key for the corresponding entity instances. Within the repository designated by the value of the `exp:authority` attribute, the values of the key attributes for each instance of the entity data type shall identify a unique entity instance.

EXAMPLE

In EXPRESS:

```
ENTITY pipe;
   id : STRING;
   Description : STRING;
UNIQUE
   key_attribute : id;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Pipe-proxy">
 <xs:complexContent>
```

```
      <xs:extension base="exp:edokey">
       <xs:sequence>
        <xs:element name="ID" type="xs:normalizedString"/>
       </xs:sequence>
      </xs:extension>
     </xs:complexContent>
    </xs:complexType>

    <xs:element name="Pipe-proxy " type="Tns:Pipe-proxy"
        substitutionGroup="exp:edokey"/>
```

### 7.5.9 XML Uniqueness constraints for entity data types

If the entity declaration for the EXPRESS entity data type contains one or more UNIQUE rules, corresponding uniqueness constraints shall be declared in the derived XML schema, as specified in this subclause.

For each UNIQUE rule contained in the entity declaration, an xs:unique schema component shall be defined in the derived XML schema.The xs:unique component shall have the form:

```
<xs:unique name="entity-rule_rulename">
 <xs:selector xpath="instance-name | subtype1 | … | subtypeN |
     exp:complexEntity/instance-name | exp:complexEntity/subtype1 | … |
     exp:complexEntity/subtypeN"/>
 <xs:field xpath="attribute1" />
 ...
 <xs:field xpath="attributeN" />
 <xs:field xpath="entity-attribute1/*/@ref" />
 ...
 <xs:field xpath="entity-attributeN/*/@ref " />

</xs:unique>
```

where:

— ***entity*** is the identifier for the EXPRESS entity data type, mapped as specified in 7.1.2.

— ***instance-name*** is the qualified name of the instance element corresponding to the entity data type, as specified in 7.5.5. Exception: Any abstract entity types shall be omitted.

— ***subtype1***, ..., ***subtypeN*** are the qualified names of the instance elements corresponding to the instantiable subtypes of the entity data type. The list includes all of the subtypes that appear anywhere in the type graph. If the entity data type is a leaf node of the type graph, there are none.

NOTE Listing the subtype instance elements requires the uniqueness constraint to hold over all instances of the entity data type, including those labeled as instances of its subtypes.

— ***rulename*** is the EXPRESS identifier for the UNIQUE rule, converted as specified in 7.1.2. If the rule has no identifier, the ***rulename*** for the first UNIQUE rule shall be 1, the second 2, and so on.

— ***attribute1***, ..., ***attributeN*** are the XML names for the accessor elements, whose data types are not entity data types, corresponding to the EXPRESS attributes specified in the UNIQUE rule.

— ***entity-attribute1, ..., entity-attributeN*** are the XML names for the accessor elements, whose data types are entity data types, corresponding to the EXPRESS attributes specified in the UNIQUE rule.

EXAMPLE

In EXPRESS:

```
ENTITY product_definition;
   for_version: product_definition_formation;
   id: STRING;
UNIQUE
   ur1: for_version, id;
END_ENTITY;

ENTITY product_definition_with_attached_document
  SUBTYPE OF (product_definition);
   ...
END_ENTITY

ENTITY product_definition_with_attached_document_2
  SUBTYPE OF (product_definition);
   ...
END_ENTITY
```

In XML Schema:

```
<xs:unique name="Product_definition-rule_Ur1">
   <xs:selector xpath="Tns:Product_definition |
      Tns:Product_definition_with_attached_document |
      Tns:Product_definition_with_attached_document_2_ |
   exp:complexEntity/Tns:Product_definition |
   exp:complexEntity/Tns:Product_definition_with_attached_document |
   exp:complexEntity/Tns:Product_definition_with_attached_document_2"/>
   <xs:field xpath="For_version/*/@ref"/>
   <xs:field xpath="Id"/>
</xs:unique>
```

## 7.6 XML Schema declarations for EXPRESS attributes

This subclause specifies the mapping of EXPRESS attributes to XML schema.

7.6.2 specifies which EXPRESS attributes of the owning entity shall be mapped to the XML schema.

For each EXPRESS attribute that is specified in 7.6.2 to be mapped to the XML schema, the EXPRESS attribute shall be mapped to an XML element, referred to as the *accessor element* corresponding to that EXPRESS attribute, as specified in 7.6.3.

### 7.6.1 Accessor element and attribute naming

The name of the accessor element corresponding to the EXPRESS attribute shall be derived from the EXPRESS identifier for the attribute as specified in 7.1.2.

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

— the XML name derived as specified in 7.1.2 from the identifier for the EXPRESS entity data type whose entity-declaration declares the attribute, followed by

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

— the XML name derived from the EXPRESS attribute identifier as specified in 7.1.2.

NOTE 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. As specified in 7.5.3.3, the XML Schema data type corresponding to the entity data type contains only one accessor element. The above exception does not apply.

### 7.6.2 EXPRESS attributes mapped to XML schema

This subclause specifies the kinds of EXPRESS attributes that are mapped to XML schema and the circumstances under which they are mapped.

#### 7.6.2.1 Explicit attributes

For each explicit EXPRESS attribute appearing in the entity declaration, there shall be a corresponding accessor element declaration, as specified in 7.6.3,

#### 7.6.2.2 INVERSE and DERIVEd attributes

EXPRESS `DERIVE`d attributes and EXPRESS `INVERSE` attributes shall not be mapped to the derived XML schema.

#### 7.6.2.3 Redeclared attributes

If an EXPRESS attribute is redeclared in a subtype, the data type of the redeclared attribute affects the declaration of the corresponding accessor elements, as specified in 7.5.4. Except as specified in 7.5.4, a redeclared attribute is not mapped to the derived XML schema.

If an explicit attribute is redeclared as `DERIVE`d or `INVERSE` in a subtype, the original attribute in the supertype shall be mapped as if it were declared `OPTIONAL`.

#### 7.6.2.4 Generic attributes

An EXPRESS abstract entity data type (see 6.3) may have attributes whose EXPRESS data types are "generalized types" that are not instantiable. Such attributes are said to be *generic attributes*. They are required by ISO 10303-11 to be redeclared in every instantiable subtype of the abstract entity data type.

When the abstract entity data type is mapped to XML schema as specified in 7.5, no generic attribute shall be mapped to XML schema as an attribute of the abstract entity data type. Only the redeclarations of the generic attribute in the instantiable subtypes shall be mapped.

NOTE In the XML schema, the redeclared attributes will appear first in the extensions that correspond to the subtypes, and they will appear to be new attributes of the subtypes.

EXAMPLE

In EXPRESS:

```
ENTITY approval_relationship ABSTRACT SUPERTYPE;
  approval_data : approval;
  approval_item : GENERIC_ENTITY;
END_ENTITY;
```

 

In XML schema :

```
<xs:complexType name="Approval_relationship">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
        <xs:element name="Approval_data">
          <xs:complexType>
            <xs:all>
              <xs:element ref="Tns:Approval" />
            </xs:all>
          </xs:complexType>
        </xs:element>
        <!-- there is no declaration for Approval_item -->
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

### 7.6.3 Accessor elements

For each EXPRESS attribute that is mapped to an accessor element (see 7.6 above), the `<all>` particle in the `complexType` corresponding to the entity data type shall contain one corresponding declaration for the attribute element.

The accessor element shall be declared as follows:

— The name of the accessor element shall as specified in 7.6.1.

— The `maxOccurs` attribute of the accessor element need not be specified in the element declaration. If it is specified, it shall have the value `"1"`.

— If the EXPRESS attribute is mandatory, the `minOccurs` attribute of the accessor element need not be specified in the element declaration. If it is specified, it shall have the value `"1"`.

— If the EXPRESS attribute is declared to be `OPTIONAL`, or specified in 7.6.2 to be treated as `OPTIONAL`, the XML `element` declaration shall contain the XML attribute `minOccurs="0"`, and the XML attribute `nillable="true"`.

NOTE 1 An unset EXPRESS attribute is indicated in the instance data either by the absence of the attribute element or by an empty accessor element with XML attribute `xsi:nil="true"`. (See 9.3.1)

NOTE 2 If the EXPRESS attribute is redeclared as `DERIVE`d or `INVERSE` in any subtype of this entity data type, the EXPRESS attribute is treated as being `OPTIONAL`, as specified in 7.6.2.3.

— The accessor element may be declared to have the XML attribute `exp:attributeType`, defined in the Base XML Schema, with a `fixed` value. If present, the value shall be `"explicit"`.

— If the data type of the EXPRESS attribute is not an entity data type, the XML data type of the accessor element shall be as specified in 7.6.3.1.

— If the data type of the EXPRESS attribute is an entity data type, the XML data type of the accessor element shall be as specified in 7.6.3.2.

#### 7.6.3.1 Attribute whose data type is not an entity data type

If the data type of the EXPRESS attribute is not an entity data type, the data type of the accessor element shall be declared to be the XML schema data type that corresponds to the data type of the EXPRESS attribute, as specified in 7.2.

EXAMPLE 1 Example of attributes whose data types are simple:

In EXPRESS:

```
TYPE Identifier = STRING;
END_TYPE;

TYPE Weight_measure = REAL;
END_TYPE;

ENTITY Pipe;
    Nominal_size : NUMBER;
    Diameter : REAL;
    Bends : INTEGER;
    Available : BOOLEAN;
    Valid : LOGICAL;
    Description : OPTIONAL STRING;
    Image : BINARY;
    Id : Identifier;
    Weight : Weight_measure;
END_ENTITY;
```

In XML Schema:

```
<xs:simpleType name = "Identifier">
  <xs:restriction base = "xs:normalizedString"/>
</xs:simpleType>

<xs:simpleType name = "Weight_measure">
  <xs:restriction base = "xs:double"/>
</xs:simpleType>

<xs:complexType name = "Tns:Pipe">
  <xs:complexContent>
    <xs:extension base = "exp:Entity">
      <xs:all>
        <xs:element name = "Nominal_size" type = "xs:decimal"/>
        <xs:element name = "Diameter" type = "xs:double"/>
        <xs:element name = "Bends" type = "xs:long"/>
        <xs:element name = "Available" type = "xs:boolean"/>
        <xs:element name = "Valid" type = "exp:logical"/>
        <xs:element name = "Description" type = "xs:normalizedString"
           minOccurs="0" nillable="true"/>
        <xs:element name = "Image" type = "exp:hexBinary"/>
        <xs:element name = "Id" type = "Tns:Identifier"/>
        <xs:element name = "Weight" type = "Tns:Weight_measure"/>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML instance data:

```
<Tns:Pipe id = "i10">
```

 

```
   <Nominal_size>3.0</Nominal_size>
   <Diameter>2.75</Diameter>
   <Bends>2</Bends>
   <Available>true</Available>
   <Valid>unknown</Valid>
   <Description>
Standard rigid double-elbow</Description>
   <Image>8AF</Image>
   <Id>P1-1</Id>
   <Weight>34.8</Weight>
</Tns:Pipe>
```

EXAMPLE 2 Example of an attribute whose data type is an aggregate of a simple data type:

In EXPRESS:

```
ENTITY Pt3d;
    c : ARRAY [1:3] OF REAL;
END_ENTITY;
```

In XML schema:

```
<xs:complexType name="Pt3d">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
        <xs:element name="C">
          <xs:complexType>
            <xs:simpleContent>
              <xs:restriction base="Tns:Seq-double">
                <xs:simpleType>
                  <xs:restriction>
                    <xs:simpleType>
                      <xs:list itemType="xs:double"/>
                    </xs:simpleType>
                    <xs:minLength value="3"/>
                    <xs:maxLength value="3"/>
                  </xs:restriction>
                </xs:simpleType>
                <xs:attribute ref="exp:arraySize" fixed="3" />
                <xs:attribute ref="exp:cType" fixed="array" />
              </xs:restriction>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="Pt3d" type="Tns:Pt3d"
  nillable="true" block="extension restriction"
  substitutionGroup="exp:Entity"/>
```

XML instance data:

```
<Tns:Pt3d id = "i4">
  <C>1.0 0.0 0.0</C>
</Tns:Pt3d>
```

EXAMPLE 3 Example of an attribute whose data type is an aggregate of an entity data type:

In EXPRESS:

```
ENTITY Pt3d;
    c : ARRAY[1:3] OF REAL;
INVERSE
    in_curve: SET [0:1] OF Polyline FOR points;
END_ENTITY;

ENTITY Polyline;
    points : LIST OF Pt3d;
END_ENTITY;
```

In XML Schema, the declarations for `Pt3d` are exactly the same as in Example 2 above. The `INVERSE` attribute `Pt3d.in_curve` is not mapped to XML schema.

The XML schema declarations for `Polyline` are:

```
<xs:complexType name = "Polyline">
  <xs:complexContent>
    <xs:extension base = "exp:Entity">
      <xs:all>
        <xs:element name = "Points">
          <xs:complexType>
                <xs:sequence>
                  <xs:group ref = "Tns:Pt3d-complexEntity-group"
                    minOccurs = "0" maxOccurs = "unbounded"/>
                </xs:sequence>
                <xs:attribute name="ref" type="xs:IDREF"
                  use="optional"/>
                <xs:attribute ref = "exp:itemType"
                  fixed = "Tns:Pt3d"/>
                <xs:attribute ref = "exp:cType" fixed = "list"/>
                <xs:attribute ref="exp:arraySize" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML instance data:

```
<Tns:Polyline id = "i1">
  <Points>
    <Tns:Pt3d id = "i4">
      <C>1.0 0.0 0.0</C>
    </Tns:Pt3d>
    <Tns:Pt3d id = "i5">
      <C>0.0 1.0 0.0</C>
    </Tns:Pt3d>
    <Tns:Pt3d ref = "i6" xsi:nil = "true"/>
  </Points>
</Tns:Polyline>

<Tns:Pt3d id = "i6">
  <C>0.0 0.0 1.0</C>
</Tns:Pt3d>
```

EXAMPLE 4

Continuing the EXPRESS in 7.3.4.1:

```
ENTITY my_select_entity;
  one_select_type : my_select2;
END_TYPE;
```

In XML Schema:

```
<xs:complexType name="My_select_entity">
 <xs:complexContent>
    <xs:extension base="exp:Entity">
       <xs:all>
          <xs:element name="One_select_type" type="Tns:An_odd_int"/>
       </xs:all>
    </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

### 7.6.3.2 Attribute whose data type is an entity data type

This subclause specifies the mapping of EXPRESS attributes whose data type is an entity data type.

The XML data type of the accessor element shall be an anonymous `complexType`, of the form:

```
<xs:complexType>
 <xs:sequence>
    <xs:group ref="instance-complexEntity-group" />
 </xs:sequence>
</xs:complexType>
```

where:

— ***instance-complexEntity-group***: shall be the qualified name of the corresponding complexEntity group (see 7.5.6.2).

EXAMPLE

In EXPRESS:

```
ENTITY measure_with_unit;
  value_component : measure_value;
  unit_component : named_unit;
END_ENTITY;

TYPE measure_value = REAL;
END_TYPE;

ENTITY named_unit;
  dimensions : dimensional_exponents;
END_ENTITY;

ENTITY si_unit
  SUBTYPE OF(named_unit);
    prefix     : OPTIONAL si_prefix;
    name       : si_unit_name;
  DERIVE
    SELF\named_unit.dimensions : dimensional_exponents :=
       dimensions_for_si_unit(name);
```

```
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name = "Measure_with_unit">
  <xs:complexContent>
    <xs:extension base = "exp:Entity">
      <xs:all>
        <xs:element name = "Value_component"
          type="Tns:Measure_value" />
        <xs:element name = "Unit_component" >
          <xs:complexType>
            <xs:sequence>
              <xs:group ref = "Tns:Named_unit-complexEntity-group"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

## 7.7 XML Schema and namespaces for EXPRESS Schema

The EXPRESS `SCHEMA` statement for the context schema shall map to the XML Schema component `xs:schema`. That is, there shall be one default XML schema corresponding to the context EXPRESS schema.

An XML namespace (the "target namespace") shall be associated with the derived XML schema. The namespace URI associated with the target namespace shall be as specified in 7.7.2. The `xs:schema` element shall have a `targetNamespace` attribute, whose value shall be the namespace URI associated with the target namespace.

The name and version of the EXPRESS schema should be provided in `documentation` elements for the XML schema.

NOTE The name and version of the EXPRESS schema are not formally mapped to any elements of the derived XML schema. They may influence the formation of the `target_namespace` URI. See 7.7.2.

Additional namespace declarations for the derived XML Schema shall be as specified in 7.7.3.

The derived XML schema shall comprise exactly the definitions and declarations specified in 7.2 through 7.6 and 7.8.

The Base XML Schema specified in Annex C shall be explicitly included in the derived XML schema by means of an `xs:import` declaration. In this case, the preprocessor shall choose a namespace prefix (see 7.7.3) to correspond to the prefix "`exp:`" that appears in 7.2 through 7.6 and 7.8, and the chosen prefix shall correspond to the namespace identified by the URN: `urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common`.

### 7.7.1 Namespace prefixes

For each namespace, the namespace prefix shall be specified by the preprocessor, with the following restrictions:

— The XML document shall map each single namespace prefix to one and only one namespace URI for the entire document.

 

— Each namespace URI shall be represented by one and only one namespace prefix for the entire document.

The namespace prefixes used in this Part of ISO 10303 – `xs:`, `exp:`, `doc:`, `cnf:` and `Tns:` – are used only to separate and identify the (potentially) distinct namespaces required. The preprocessor is free to substitute any valid prefix for any of them, within the constraints above.

### 7.7.2 URI for the target namespace of the derived XML schema

The namespace URI associated with the target namespace for the derived XML schema may be chosen by the preprocessor, with the following guidelines.

If the derived XML schema is the default mapping of an EXPRESS schema that has been registered as an International Standard, the URI shall be a URN having the form specified in Annex A.

The URI that designates the target namespace for any other mapping of an EXPRESS schema that has been registered as an International Standard is not specified by this part of ISO 10303. It may be specified by the corresponding International Standard, or by other means conforming to IETF standards.

The URI that designates the target namespace for any mapping of an EXPRESS schema that has not been registered as an International Standard shall be defined by the owning body, in conformance with IETF RFC 2396 and the appropriate IETF registration procedures, and is out of the scope of this International Standard.

NOTE As specified in IETF 2396, Uniform Resource Locator (URL) and Uniform Resource Name (URN) are subtypes of Uniform Resource Identifier (URI). A URI may or may not represent a means by which the identified resource can be accessed. URLs always represent an access path, URNs never do. Each class of URI is prefixed by a code for the class of URI — uri:, urn: url: or http: — and an initial identifier that is registered with the Internet Assigned Names Authority (IANA) as the unique identifier for an organization that owns the URI, although possibly not the resource to which it refers. The registration procedures for different classes are specified by different IETF standards. Notably, the syntax of URNs is specified in IETF RFC 2141 [3] and the corresponding registration procedures are specified in IETF RFC 3406 [4].

EXAMPLE

The XML namespace for the default mapping of the `config_control_design` EXPRESS schema contained in ISO 10303-203 [2] would be.

In XML Schema:

```
<schema targetNamespace=
  "urn:iso.org:standard:10303:part(203):version(2):
    defaultXML:Config_control_design"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:exp=
   "urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"
  xmlns:a203="urn:iso.org:standard:10303:part(203):version(2):
    defaultXML:Config_control_design">…
```

In XML:

```
<uos
  xmlns:a203="urn: iso.org:standard:10303:part(203):version(2):
    defaultXML:Config_control_design" …>
```

### 7.7.3 Namespace declarations for the derived XML schema

The `xs:schema` element shall contain (xmlns) namespace declarations for:

— the targetNamespace. The value of the namespace URI used in this namespace declaration shall be the same as that assigned to the `targetNamespace` attribute.

— the current version of the XML Schema definition.

— the Base XML Schema (see Annex C).

For any `xmlns` declaration required above, the preprocessor may choose the prefix, as specified in 7.7.1, and the appropriate URI.

If the `xs:schema` element contains the `elementFormDefault` or `attributeFormDefault` attribute, their respective values shall be set to `'unqualified'`, which is the default value specified in *XML Schema Part 1: Structures*.

EXAMPLE The following example illustrates the mapping of an EXPRESS `SCHEMA` statement to XML Schema:

In EXPRESS:

```
SCHEMA  my_schema;
 …
END_SCHEMA;
```

In XML Schema:

```
<xs:schema
  targetNamespace="urn:xyz.org:xs/My_schema"
  xmlns:my="urn:xyz.org:xs/My_schema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:exp=
   "urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

  <!-- import statements and schema components mapped from the type
and entity definitions in the EXPRESS schema would appear here -->

</xs:schema>
```

### 7.7.4 Import declarations for the derived XML schema

For EXPRESS schemas represented in the short form, the `xs:schema` element corresponding to the context EXPRESS schema shall contain an immediate child `xs:import` element for each namespace corresponding to an EXPRESS schema interfaced with the EXPRESS context schema. The value of the namespace attribute of the import element shall be the URN associated with the interfaced schema. The value of the `schemaLocation` attribute of the import element shall be the Uniform Resource Identifier (URI) of the XML schema representing the interfaced EXPRESS schema, interpreted as a locator by which the XML schema can be accessed over the Internet. The `xs:schema` element shall contain namespace declarations for each of the EXPRESS schema being interfaced with the context schema.

EXAMPLE The following example illustrates the mapping of interfaced EXPRESS schemas to XML Schema:

 

In EXPRESS:

```
SCHEMA  my_schema;

USE FROM other_schema (some_entity);
REFERENCE FROM another_schema (another_entity);
 …
END_SCHEMA;
```

In XML Schema:

```
<xs:schema
  targetNamespace="urn:xyz.org:xs/My_schema"
  xmlns:my="urn:xyz.org:xs/My_schema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:exp=
   "urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"
  xmlns:oth="urn:xyz.org:xs/Other_schema"
  xmlns:ano="urn:xyz.org:xs/Another_schema" >

  <xs:import namespace="urn:xyz.org:xs/Other_schema"
    schemaLocation="other_schema.xs"/>
  <xs:import namespace="urn:xyz.org:xs/Another_schema"
    schemaLocation="another_schema.xs"/>
  <xs:import schemaLocation="exp.xsd"
    namespace="urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"/>
<!-- other entities omitted for simplicity -->

</xs:schema>
```

## 7.8 Context-schema specific unit of serialization

For the context schema there shall be a specific unit of serialization element and a corresponding `complexType` in the namespace corresponding to the derived XML schema

The unit of serialization element declaration shall have the form:

```
<xs:element type="Tns:uos" name="uos" substitutionGroup="exp:uos">
 uniqueness_constraints
</xs:element>
```

where:

— ***Tns*** is the namespace prefix that corresponds to the namespace associated with the derived XML schema for the unit of serialization, as specified in 7.7.

— ***uniqueness_constraints:*** `any xs:unique` schema components required by 7.5.9.

The context specific `complexType` shall be an extension of the `exp:uos complexType`, described in 5.6 (which has no content model).

The content model of the context-specific `complexType` shall be a repeating `choice` group consisting of:

— exp:Entity (see Annex C),

— the `exp:edokey` instance element, and

— every referenceable instance element that corresponds to a non-entity data type (see 7.4.6).

That is, the `complexType` for the unit of serialization shall have the form:

```
<xs:complexType name="uos">
  <xs:complexContent>
        <xs:extension base="exp:uos">
            <xs:choice maxOccurs="unbounded" minOccurs="0">
                <xs:element ref="exp:Entity" />
                <xs:element ref="exp:edokey" />
                <xs:element ref="referenceable-instance-name-1"/>
                        ...
                <xs:element ref="referenceable-instance-name-n"/>
            </xs:choice>
        </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

where:

— ***referenceable-instance-name-i*** are the fully qualified names for the referenceable non-entity instance elements specified above.

EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="urn:xyz.com/Abc"
   xmlns:Tns="urn:xyz.com/Abc"
   xmlns:exp=
  "urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"
   elementFormDefault="unqualified"
   attributeFormDefault="unqualified">
 <xs:import schemaLocation="exp.xsd"
namespace="urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"/>

 <xs:element name="uos" type="Tns:uos" substitutionGroup="exp:uos"/>

 <xs:element name="A" …> ... </xs:element>
 <xs:element name="B" …> ... </xs:element>
       ...

 <xs:complexType name="uos">
   <xs:complexContent>
     <xs:extension base="exp:uos">
       <xs:choice maxOccurs="unbounded" minOccurs="0">
          <xs:element ref="exp:Entity" />
          <xs:element ref="exp:edokey" />
          <xs:element ref="Tns:A"/>
          <xs:element ref="Tns:B"/>
       </xs:choice>
     </xs:extension>
   </xs:complexContent>
 </xs:complexType>

</xs:schema>
```

 

# 8 Configured XML Schema Binding

This clause specifies a mapping from an EXPRESS schema to its corresponding configured XML Schema declarations, definitions and representation. The derived XML schema as specified in this clause is said to be a *configured XML schema* for the given EXPRESS schema. This clause applies if a configuration file is given, and the configuration file contains at least one directive that does not specify the default value. In all other cases, Clause 7 shall apply. This clause is organized such that the XML schema derived from an EXPRESS schema is specified in 8.1 through 8.7.

## 8.1 Naming conventions

This clause specifies uniform conventions for creating XML names based on EXPRESS identifiers.

### 8.1.1 Schema

Every EXPRESS schema shall have an associated XML namespace URI. This URI shall designate the target namespace for the XML element and data type names that are schema-specific elements of the binding of that schema, as specified in 8.7.

### 8.1.2 EXPRESS identifiers

EXPRESS language identifiers are not case-sensitive. XML names are case-sensitive. Unless otherwise specified in a particular case, an EXPRESS identifier shall be mapped to an XML name, or the local part of an XML name, as follows:

— When the **naming-convention** option (see 10.2.14) has the value **initial-upper**, the XML name shall be the same as the EXPRESS identifier except that the initial letter is upper case and all other letters are lower case. This is the default.

— When the **naming-convention** option has the value **camel-case**, the XML name shall be derived from the EXPRESS identifier as follows: the first letter is upper case, the first character following each underscore is upper case, all other characters in the identifier shall be lower case, and all underscores shall be deleted.

— When the **naming-convention** option has the value **preserve-case**, the XML name corresponding to an EXPRESS identifier shall be identical in letter case to the representation of the identifier in the text of the EXPRESS schema. A conforming pre-processor is not required to support this value for **naming convention**.

Due to the XML naming restrictions specified in the XML Recommendation, any EXPRESS identifier 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, and attribute names. Exceptions for certain situations are specified in the subclauses dealing with particular kinds of EXPRESS identifiers.

### 8.1.3 Data types

Each EXPRESS data type identifier shall be mapped to a qualified name as defined in the Namespaces in XML Recommendation.

The local part of the qualified name shall be the EXPRESS data type identifier mapped as specified in 8.1.2. Exception: If an **entity** or **type** configuration directive (see 10.3.2 and 10.3.3) selects the

EXPRESS data type and provides a **name** attribute, the local part of the qualified name shall be as specified by that directive.

For every EXPRESS named data type declared in, or explicitly interfaced into, the context schema, the associated XML Schema namespace for the qualified name shall be that of the context schema.

For every data type that is implicitly interfaced into the context schema, the local part of the qualified name shall be derived as specified in 8.1.2 from the EXPRESS identifier for that data type, as it appears in the schema in which the data type is declared. The associated namespace shall be that of the schema in which the data type is declared.

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

## 8.2 XML Schema data types corresponding to EXPRESS data types

For every EXPRESS data type appearing in the context schema, including named data types, built-in data types and anonymous constructed data types, there is a corresponding XML schema data type. This subclause specifies that correspondence.

NOTE This subclause does not specify any definitions or declarations to appear in the derived XML schema. It specifies XML schema components that may appear in definitions and declarations required by other subclauses.

### 8.2.1 EXPRESS simple data types

This subclause specifies the XML schema datatypes that correspond to the EXPRESS built-in simple data types.

#### 8.2.1.1 BINARY data type

The EXPRESS `BINARY` data type is mapped to the `exp:hexBinary` data type, or to `exp:base64Binary` if the configuration directive **map="xs:base64Binary"** applies to the `BINARY` data type (see 10.2.13.1). Exceptions:

— When an EXPRESS attribute whose data type is `BINARY` is mapped to an accessor attribute, the corresponding XML data type shall be `xs:hexBinary`, or `xs:base64Binary` if the configuration directive **map="xs:base64Binary"** applies.

— If the EXPRESS `BINARY` data type specifies a maximum or `FIXED` length, it shall be mapped as specified in 8.2.1.1.1.

The `exp:hexBinary` and `exp:base64Binary` data types are defined in the Base XML Schema (see Annex C).

EXAMPLE extending the example in 7.2.1.1:

Given the configuration directive

```
<attribute select="a_binary" map="xs:base64Binary"/>
```

In XML Schema (inheritance-free mapping):

```
<xs:complexType name="Entity_with_binary_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
```

 

```
            <xs:element type="exp:base64Binary" name="A_binary"/>
          </xs:all>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
```

#### 8.2.1.1.1 Constrained BINARY data types

An EXPRESS BINARY data type for which a maximum or FIXED length is specified is said to be a *constrained* BINARY data type. It shall be mapped to a complexType that is a restriction of the exp:hexBinary data type, or of the exp:base64Binary data type if the configuration directive **map="xs:base64Binary"** applies.

When the constrained BINARY data type is the underlying type in an EXPRESS type declaration, the corresponding XML data type shall be as specified in 8.3.1.1.

When a constrained BINARY data type is the data type of an EXPRESS attribute that is mapped to an accessor attribute (see 8.6.3), the data type of the accessor attribute shall be a simpleType of the form:

```
<xs:simpleType>
  <xs:restriction base="binary-type">
    <xs:minLength value="minLength"/>
    <xs:maxLength value="maxLength"/>
  </xs:restriction>
</xs:simpleType>
```

where:

— ***binary-type***, ***minLength*** and ***maxLength*** are as specified below.

NOTE 1 It is assumed in this case that the length is a multiple of 8 bits, and therefore, the value of extraBits is always zero.

For any other use of a constrained BINARY data type, the derived XML schema shall contain a type definition for the XML data type that corresponds to the constrained BINARY data type. The type definition shall have the form:

```
<xs:complexType name="typename.minBits.bits">
  <xs:simpleContent>
   <xs:restriction base="binary-type">
     <xs:minLength value="minLength"/>
     <xs:maxLength value="maxLength"/>
     <xs:attribute name="extraBits" type="xs:integer"
        fixed="remainder"/>
   </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

where:

— ***typename***: If a **map** configuration directive applies to the constrained BINARY data type, ***typename*** shall be the local part of the name of the XML data type specified by the **map** directive; otherwise, ***typename*** shall be Binary.

— ***minBits*** shall be the fixed length (in bits) specified for the EXPRESS BINARY data type, if it is declared FIXED. Otherwise, ***bits*** shall be "0".

— ***bits*** shall be the maximum or fixed length (in bits) specified for the EXPRESS `BINARY` data type.

— ***binary-type*** is `exp:hexBinary`, or `exp:base64Binary` if **map="xs:base64Binary"** applies to the `BINARY` data type.

— ***minLength*** shall be the octet value (see below) of the fixed length specified for the EXPRESS `BINARY` data type, if it is declared `FIXED`. Otherwise, the `xs:minLength` facet may be omitted.

— ***maxLength*** shall be the octet value of the maximum (or fixed) length specified for the EXPRESS `BINARY` data type.

— ***remainder*** shall be a decimal integer giving the value (***bits*** modulo 8), where ***bits*** is as specified below, if the EXPRESS data type is declared `FIXED`. Otherwise, the redeclaration of the `extraBits` attribute shall be omitted.

Given ***bits*** = the maximum (or fixed) length in bits specified for the EXPRESS data type, the *octet value* shall be calculated as follows:

— If *bits* modulo 8 is greater than 0, *bits* / 8 + 1.

— If *bits* modulo 8 is equal to 0, *bits* / 8.

NOTE 2 When the configuration directive (see 10.2.13) **map="xs:base64Binary"** applies to the EXPRESS data type, the *octet value* is the same as above.

NOTE 3 The values of ***minBits*** and ***bits*** are lengths in bits, while the values of ***minLength*** and ***maxLength*** are lengths in octets.

EXAMPLE

In EXPRESS:

```
ENTITY entity_with_fixed_binary_attribute;
  a_binary : BINARY(12)FIXED;
  b_binary : BINARY(8);
END_ENTITY;
```

In XML Schema:

If **exp-attribute="attribute-content"** does not apply, see the example XML Schema in 7.2.1.1.1:

If **exp-attribute="attribute-content"** applies:

```
<xs:complexType name="Entity_with_fixed_binary_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:attribute name="A_binary">
        <xs:simpleType>
          <xs:restriction base="xs:hexBinary">
            <xs:maxLength value="2"/>
            <xs:minLength value="2"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="B_binary">
        <xs:simpleType>
          <xs:restriction base="xs:hexBinary">
```

 

```
            <xs:maxLength value="1"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

#### 8.2.1.2 BOOLEAN data type

The EXPRESS `BOOLEAN` data type is mapped to the `xs:boolean` data type.

See EXAMPLE in 7.2.1.2.

#### 8.2.1.3 INTEGER data type

The EXPRESS `INTEGER` data type is mapped to `xs:long`. Exception: When the **map** configuration directive applies to the data type (see 10.5.4), the EXPRESS `INTEGER` data type shall be mapped to the value specified by that directive (see 10.2.13.2).

EXAMPLE extending the example in 7.2.1.3:

Given the configuration directive

```
<attribute select="an_integer" map="xs:year"/>
```

In XML Schema (inheritance-free mapping):

```
<xs:complexType name="Entity_with_integer_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
        <xs:element type="xs:year" name="An_integer"/>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

#### 8.2.1.4 LOGICAL data type

The EXPRESS `LOGICAL` data type is mapped as specified in 7.2.1.4.

#### 8.2.1.5 NUMBER data type

The EXPRESS `NUMBER` data type is mapped to the `xs:decimal` data type. Exception: When the **map** configuration directive applies to the data type (see 10.5.4), the EXPRESS `NUMBER` data type shall be mapped to the value specified by that directive (see 10.2.13.3).

EXAMPLE extending the example in 7.2.1.5:

Given the configuration directive

```
<attribute select="a_number" map="xs:double"/>
```

In XML Schema:

```
<xs:complexType name="Entity_with_number_attribute">
  <xs:complexContent>
```

```
      <xs:extension base="exp:Entity">
        <xs:all>
          <xs:element type="xs:double" name="A_number"/>
        </xs:all>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

#### 8.2.1.6 REAL data type

The EXPRESS `REAL` data type is mapped to the `xs:double` data type. Exception: When the **map** configuration directive applies to the data type (see 10.5.4), the EXPRESS `REAL` data type shall be mapped to the value specified by that directive (see 10.2.13.3).

EXAMPLE extending either example in 7.2.1.6:

Given the configuration directive

```
<attribute select="a_real" map="xs:float"/>
```

In XML Schema (inheritance mapping):

```
<xs:complexType name="Entity_with_real_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:sequence>
        <xs:element type="xs:float" name="A_real"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

#### 8.2.1.7 STRING data type

If the EXPRESS data type is declared to have a maximum length or a `FIXED` length, the EXPRESS data type shall be mapped as specified in 8.2.1.7.1. Otherwise, when the **map** configuration directive applies to the data type (see 10.5.4), the EXPRESS `STRING` data type shall be mapped to the value specified by that directive (see 10.2.13.4). In all other cases, the EXPRESS `STRING` data type is mapped to the `xs:normalizedString` data type.

EXAMPLE extending the example in 7.2.1.7

Given the configuration directive

```
<attribute select="a_string" map="xs:QName"/>
```

In XML Schema (inheritance mapping):

```
<xs:complexType name="Entity_with_string_attribute">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:sequence>
        <xs:element type="xs:QName" name="A_string"/>
      </xs:sequence>
    </xs:extension>
```

```
    </xs:complexContent>
  </xs:complexType>
```

#### 8.2.1.7.1 Constrained STRING data types

An EXPRESS STRING data type for which a maximum or FIXED length is specified is said to be a *constrained* STRING data type. It shall be mapped to a simpleType that is a restriction of the xs:normalizedString data type.

When the constrained STRING data type is the underlying type in an EXPRESS type declaration, the corresponding XML data type shall be as specified in 8.3.1.3.

Otherwise, the derived XML schema shall contain a type definition for the XML data type corresponding to the constrained STRING data type. The type definition shall have the form:

```
<xs:simpleType name="typename.minLength.maxLength">
    <xs:restriction base="string-type">
      <xs:minLength value="minLength"/>
      <xs:maxLength value="maxLength"/>
    </xs:restriction>
</xs:simpleType>
```

where:

— ***typename***: If a **map** configuration directive applies to the constrained STRING data type, ***typename*** shall be the local part of the name of the XML data type specified by the **map** directive; otherwise, ***typename*** shall be String.

— ***minLength*** shall be the fixed length specified for the EXPRESS STRING data type, if it is declared FIXED. Otherwise, it shall be "0" and the xs:minLength facet may be omitted.

— ***maxLength*** shall be the maximum (or fixed) length specified for the EXPRESS STRING data type.

— ***string-type***: If a **map** configuration directive applies to the constrained STRING data type, ***string-type*** shall be the qualified name of the XML data type specified by the map directive; otherwise, ***string-type*** shall be xs:normalizedString.

NOTE If the EXPRESS schema declares a defined data type for the constrained STRING data type, there will be an XML definition corresponding to the defined data type (see 8.3). The definition above is only required for constrained STRING data types that are anonymous.

EXAMPLE See example in 7.2.1.7.1.

### 8.2.2 Aggregation data types

For an EXPRESS aggregation data type, that is, a SET, LIST, BAG or ARRAY data type, the *base-type* of the aggregate is the data type of the component values. The XML Schema data type corresponding to the aggregation data type depends on the base-type, and on the value of the configuration directive **tagless** that applies.

NOTE 1 The XML Schema data type corresponding to the aggregation data type in an EXPRESS attribute declaration may depend on the value of the directive **exp-attribute** that applies, because **exp-attribute** can affect the default value of **tagless**.

NOTE 2 Because the configuration directives can be applied to all uses of the data type, or only to specific EXPRESS attributes, the corresponding XML schema data type may be different in different uses.

The XML data type that corresponds to a given use of the EXPRESS aggregation data type shall be as specified in Table 3 for the combination of aggregate type, base-type and **tagless** that applies to that use, as follows:

— The value for Base-type shall be as follows: If no **map** directive applies to the base-type of the aggregate (see 10.5.4), the base-type shall be as specified in the EXPRESS declaration for the aggregation data type. If a **map** directive applies to the base-type of the aggregate, and the values of the named XML data type contain no whitespace, the base-type shall be considered to be a *simple* data type, and the default value of **tagless** shall be **true**. If a **map** directive applies to the base-type of the aggregate, and the values of the named XML data type may contain whitespace, the base-type shall be considered to be a `STRING` data type, and the default value of **tagless** shall be **false**.

— *Simple* types are those whose XML representations have simpleTypes and contain no spaces, including EXPRESS types `BOOLEAN`, `INTEGER`, `LOGICAL`, `REAL` and `NUMBER`, named data types that are mapped to certain XML data types by the **map** configuration directive as specified above, and defined data types that are specializations of any of these.

— The value for Aggregate-type shall be as specified in the EXPRESS declaration for the aggregation data type. For the purposes of this table, `ARRAY OF OPTIONAL` base-type is considered a distinct aggregate-type.

— The value for the directive **tagless** that applies to the aggregation data type shall be as specified in 10.5.5.

— The applicable data type properties and the applicable values of the configuration directives should be ordered like the columns in Table 3, and compared one-for-one with the values in each row of the table.

— The value of the applicable property or directive named in the column matches the value in that column in a given row only if they are identical. Exception: Any applicable value of a directive matches a blank cell in the row.

— The applicable values will match all the values in exactly one row of the table. The corresponding XML data type shall be as specified in the subclause identified in the Subclause column of that row.

NOTE 3 Table 3 only determines the corresponding XML schema declarations. Many combinations that use the same XML schema representation have different encoding rules. See 9.8.

 

**Table 3 — Subclause governing aggregation data type correspondence**

| Aggregate type | Base type | tagless | Subclause |
|---|---|---|---|
| ARRAY/LIST/SET/BAG | Simple | true | 8.2.2.3 |
| ARRAY/LIST/SET/BAG | Simple | false | 8.2.2.2 |
| ARRAY/LIST/SET/BAG | STRING/BINARY | true | 8.2.2.3 |
| ARRAY/LIST/SET/BAG | STRING/BINARY | false | 8.2.2.2 |
| ARRAY/LIST/SET/BAG | ENUMERATION | true | 8.2.2.5 |
| ARRAY/LIST/SET/BAG | ENUMERATION | false | 8.2.2.2 |
| ARRAY/LIST/SET/BAG | SELECT | | 8.2.2.6 |
| ARRAY/LIST/SET/BAG | Entity | true | 8.2.2.8.1 |
| ARRAY/LIST/SET/BAG | Entity | false | 8.2.2.8.2 |
| ARRAY/LIST/SET/BAG | Aggregation type | | 8.2.2.4 |
| ARRAY/LIST/SET/BAG | Defined data type | | 8.2.2.7 |
| ARRAY OF OPTIONAL | Simple | | 8.2.2.2 |
| ARRAY OF OPTIONAL | STRING/BINARY | | 8.2.2.2 |
| ARRAY OF OPTIONAL | ENUMERATION | | 8.2.2.2 |
| ARRAY OF OPTIONAL | SELECT | | 8.2.2.6 |
| ARRAY OF OPTIONAL | Entity | | 8.2.2.8.2 |
| ARRAY OF OPTIONAL | Aggregation type | | 8.2.2.4 |
| ARRAY OF OPTIONAL | Defined data type | | 8.2.2.7 |

NOTE 4 As specified in 10.2.8, **tagless="true"** shall not be specified unless the base-type is one of the following:

— a simple type as defined above,

— a STRING data type whose values will not contain whitespace,

— a BINARY data type whose values will be octet-strings (multiples of 8 bits),

— a defined data type whose fundamental type is an ENUMERATION, or

— a defined data type whose fundamental type is any of the above,

— an entity data type.

NOTE 5 When the aggregation type is ARRAY OF OPTIONAL, **tagless="false"** always applies (see 10.2.8).

### 8.2.2.1 XML attributes for aggregation data types

The XML schema data types corresponding to some EXPRESS aggregation data types are the Sequence-of-elements form having all of the attributes in 7.2.2.1.

The value of the arraySize attribute indicates the number of component elements in the aggregate value represented by a particular instance. For an ARRAY, the value is always fixed and equal to HiIndex – LoIndex + 1 (see below). For a multi-dimensional array, the value is a sequence of integer values giving the number of component elements in each dimension (see 8.2.2.4 and 9.8.5).

The value of the itemType attribute and the value of the cType are as specified in 7.2.2.1.

#### 8.2.2.2 Sequence-of-elements form of aggregates

When the *sequence-of-elements* form for representation of the EXPRESS aggregation data type is specified by Table 3, the corresponding XML Schema data type shall be a `complexType` having the form specified in 7.2.2.2.

NOTE For the resulting XML Schema using the configuration directive **exp-attribute = "type-tag"** see 8.4.2.2.

#### 8.2.2.3 List-of-values form of aggregates

When the *list-of-values* form for representation of the EXPRESS aggregation data type is specified by Table 3, the corresponding XML Schema data type is based on a simple list data type, as specified in 8.2.2.3.1. The corresponding XML Schema data type shall be as follows:

— when the XML data type is the data type of an accessor attribute (see 8.6.3), it shall be as specified in 8.2.2.3.4;

— otherwise (when it is the data type of an XML element), it shall be as specified in 8.2.2.3.2.

NOTE 1 In general, when the base-type of the EXPRESS aggregation data type is `INTEGER`, `LOGICAL`, `NUMBER`, or `REAL`, or a defined data type whose fundamental type is any of these, this subclause applies. By definition, all representations of the data types `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER`, and `REAL` do not contain whitespace, and the XML `list` form – tokens separated by whitespace – is an unambiguous representation.

NOTE 2 When the base-type of the EXPRESS aggregation data type is `STRING` or `BINARY`, or a defined data type whose fundamental type is `STRING` or `BINARY`, 8.2.2.2 generally applies, but this subclause applies when specified by Table 3.

NOTE 3 When this clause applies to aggregation data types whose base-type is `STRING`, by implication, the corresponding values contain no spaces. If the values may contain spaces, this representation may be misinterpreted.

NOTE 4 When this clause applies to aggregation data types whose base-type is `BINARY`, the `xs:hexBinary` data type is used as the `itemType` of the simple list. It is not possible to specify the `extra_bits` XML attribute (see 8.2.1.1), and its value is taken to be zero.

NOTE 5 The two different XML data types are required, because the XML data type of an accessor attribute must be a `simpleType`, but for other uses, the `simpleType` loses information that distinguishes it from similar aggregate representations (see 8.2.2.8.1).

##### 8.2.2.3.1 Simple list type supporting aggregations

When the *list-of-values* form for representation of the EXPRESS aggregation data type is specified by Table 3, the corresponding XML Schema data type is based on a simple list data type. The simple list data type shall be defined in the derived XML schema by a `simpleType` definition of the form shown in 7.2.2.3.1 where:

— ***`list-type`*** shall be "`List-`", followed by the local part of the name of the XML schema data type that corresponds to the base-type of the aggregation data type, as specified in 8.2 or 8.3.

— ***`base-type`*** shall be the qualified name of the XML schema data type that corresponds to the base-type of the aggregation data type, as specified in 8.2 or 8.3.

Only one such definition is needed for all EXPRESS aggregation types having the same base-type.

 

See EXAMPLE in 7.2.2.3.1.

#### 8.2.2.3.2 List of values form for XML elements

When the list-of-values form is used as the data type of an XML element, the corresponding XML data type is a restriction of the unrestricted `complexType` that supports aggregations of the EXPRESS base type, as specified in 8.2.2.3.3.

The XML data type corresponding to the EXPRESS aggregation data type shall have the form specified in 7.2.2.3 where:

— ***unrestricted-type*** shall be the qualified name of the unrestricted `complexType` that supports aggregations of the EXPRESS base-type, as specified in 8.2.2.3.3.

— ***list-type*** shall be the qualified name of the simple list data type that corresponds to the base type, as specified in 8.2.2.3.1.

— ***lower-bound, upper-bound,*** a***ggregate-type*** and ***usage*** are as defined in 7.2.2.3.

#### 8.2.2.3.3 Unrestricted complexType supporting aggregations

The derived XML schema shall contain a `complexType` definition for the unrestricted `complexType` as shown in 7.2.2.3.2 where:

— ***Tns:list-type*** shall be the qualified name for the simple ***list-type*** defined in 8.2.2.3.1 above.

— ***seq-type*** shall be "`Seq-`", followed by the local part of the name of the XML schema data type that corresponds to the base-type of the aggregation data type, as specified in 8.2 or 8.3.

— ***base-type***, the `itemType` attribute value, shall be the qualified name of the XML schema data type that corresponds to the base-type of the aggregation data type, as specified in 8.2 or 8.3.

#### 8.2.2.3.4 List of values form for accessor attributes

When the XML data type is the data type of an accessor attribute (see 8.6.3), it shall have the form:

```
<xs:simpleType>
  <xs:restriction>
        <xs:simpleType>
          <xs:list itemType="base-type"/>
        </xs:simpleType>
        <xs:minLength value="lower-bound"/>
        <xs:maxLength value="upper-bound"/>
  </xs:restriction>
</xs:simpleType>
```

where:

— ***base-type***, the `itemType` attribute value, shall be the qualified name of the XML schema data type that corresponds to the base-type. If the base-type is a defined data type, the corresponding XML schema data type shall be as specified in 8.3. If the base-type is `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER`, `REAL`, or `STRING`, the corresponding XML schema data type shall be as specified in 8.2.1. If the base-type is `BINARY`, ***base-type*** shall be `xs:hexBinary`.

— ***lower-bound*** is the lower bound on the EXPRESS aggregate. When the lower bound is given and evaluates to a constant, the value of `minLength` shall be the value of the lower bound, represented as a decimal integer. Otherwise, the value of `minLength` shall be '`0`'. If the value of `minLength` is '`0`', the `minLength` facet may be omitted.

— ***upper-bound*** is the upper bound on the EXPRESS aggregate. When the upper bound is given and evaluates to a constant, the `maxLength` attribute shall appear and shall have the value of the upper bound, represented as a decimal integer. Otherwise, the `maxLength` facet shall be omitted.

NOTE If the EXPRESS aggregation type is an `ARRAY`, the value of the lower and upper bounds is `HiIndex(a) - LoIndex(a) + 1`, where `a` is the aggregation data type.

EXAMPLE (Compare with the example in 8.2.2.3.2 above):

In EXPRESS:

```
ENTITY Solid;
  orientation: ARRAY [1:3] OF REAL;
END_ENTITY;
```

If orientation is represented by an accessor attribute, the corresponding XML data type is:

```
<xs:simpleType>
   <xs:restriction>
       <xs:simpleType>
          <xs:list itemType="xs:double"/>
       </xs:simpleType>
       <xs:minLength value="3"/>
       <xs:maxLength value="3"/>
   </xs:restriction>
</xs:simpleType>
```

### 8.2.2.4 Aggregates of aggregation data types

When the base-type of the EXPRESS aggregation data type is another aggregation data type, or a defined data type whose fundamental type is another aggregation data type, the corresponding XML schema data type shall be as specified in this subclause.

An EXPRESS aggregation data type (`SET`, `LIST`, `BAG` or `ARRAY`) is said to be *nested*, if it appears in the context schema as the base-type of another aggregation data type.

The XML schema data type corresponding to the overall aggregation data type applies as follows:

— For all structures that are not `ARRAY OF ARRAY` structures, if the configuration directive flatten="false" applies (see 10.2.9), the mapping shall be as specified in 8.2.2.4.1.

— In all other cases, the mapping specified in 8.2.2.4.2 shall apply.

#### 8.2.2.4.1 The sequence-of-rows structure

When the sequence-of-rows structure is chosen for an EXPRESS aggregation data type (called ***type A*** below), whose base-type is another aggregation data type (called ***type B*** below), the structure of the XML schema data type corresponding to ***type A*** shall be as specified in this subclause.

 

The complexType corresponding to ***type A*** shall be a complexType whose content is a sequence of elements corresponding to ***type B***. The XML data type shall have the form:

```
<xs:complexType>
    <xs:sequence>
           <xs:element ref="base-instance"
               minOccurs = "lower-bound" maxOccurs = "upper-bound"/>
    </xs:sequence>
    <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
    <xs:attribute ref="exp:itemType" fixed="base-instance"/>
    <xs:attribute ref="exp:arraySize" usage/>
    <xs:attribute ref="exp:cType" fixed="aggregate-type"/>
</xs:complexType>
```

where:

— ***base-instance*** is the qualified name of the XML instance element corresponding to ***type B***. If ***type B*** is a defined data type, the instance element name shall be as specified in 8.4.3. If ***type B*** is an anonymous aggregation data type, the instance element name shall be as specified in 8.4.2.

— ***aggregate-type*** is the keyword for the EXPRESS aggregate type of ***type A***, one of: "set" for SET, "bag" for BAG, "list" for LIST, "array" for ARRAY, "list-unique" for LIST OF UNIQUE, "array-unique" for ARRAY OF UNIQUE, array-optional for ARRAY OF OPTIONAL, and array-optional-unique for ARRAY OF OPTIONAL UNIQUE.

NOTE 1 Although the overall structure is an aggregate of aggregates, each nested aggregate (***type B***) will be represented as a ***type B*** instance element as a component of the ***type A*** aggregate, so the ctype attribute only specifies the nature of the ***type A*** aggregate (cf. 8.2.2.4.2).

— ***lower-bound:*** When ***type A*** is not ARRAY OF OPTIONAL, and the value of the lower bound on the size of ***type A*** is given and evaluates to a constant, minOccurs shall be that value, represented as a decimal integer. Otherwise, minOccurs is '0'. If the lower bound evaluates to '1', the minOccurs attribute may be omitted.

— ***upper-bound:*** When the upper bound on ***type A*** is given and evaluates to a constant, maxOccurs is that value, represented as a decimal integer. Otherwise, the maxOccurs value shall be "unbounded".

NOTE 2 If ***type A*** is an ARRAY, the value of the lower bound and the value of the upper bound = HiIndex(***typeA***) – LoIndex(***typeA***) + 1.

— ***usage***: When ***type A*** is an ARRAY and the lower and upper bounds on the index range of type A evaluate to constants, arraySize shall be declared fixed="***upper-bound***", where ***upper-bound*** is as defined above. When ***type A*** is an ARRAY, and the lower and upper bounds on the index range of type A do not evaluate to constants arraySize shall be declared use="required". Otherwise, arraySize shall be declared use="optional".

NOTE 3 The value of arraySize in the XML encoding should be the actual size of the array in each case. See 9.8.5.

NOTE 4 This form is identical in all regards to the "sequence-of-elements" form specified in 8.2.2.2, except that the component instance elements represent ***type B*** aggregate values. It is a "sequence of elements" representation for the purposes of any other clause.

NOTE 5 This XML schema data type is used to represent values of the EXPRESS aggregation data type as specified in 9.8.5.1 and 9.8.5.2.

EXAMPLE Sequence-of-rows representation of a multi-dimensional aggregation of simple types

In EXPRESS:

```
TYPE direction_vectors = LIST[1:3] OF ARRAY[1:2] OF REAL;
END_TYPE;

ENTITY X;
  matrix : direction_vectors;
END_ENTITY;
```

In the derived XML Schema, `Tns:Seq-double` will be declared as specified in 8.4.2.1 to support `ARRAY[1:2] OF REAL`. The XML type definition corresponding to `direction_vectors` will be:

```
<xs:complexType name="Direction_vectors">
       <xs:sequence>
          <xs:element ref = "Tns:Seq-double"
            minOccurs = "1" maxOccurs = "3"/>
       </xs:sequence>
      <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
       <xs:attribute ref = "exp:arraySize" use = "optional"/>
       <xs:attribute ref = "exp:itemType" fixed="Tns:Seq-double"/>
       <xs:attribute ref = "exp:cType" fixed = "list"/>
</xs:complexType>
```

XML instance data:

```
<Tns:Matrix>
     <Tns:Seq-double >1.0 2.0</Tns:Seq-double>
     <Tns:Seq-double >0.0 4.0</Tns:Seq-double>
     <Tns:Seq-double >0.0 5.4</Tns:Seq-double>
</Tns:Matrix>
```

#### 8.2.2.4.2 The multi-dimensional array structure

When the multi-dimensional array structure is chosen for an EXPRESS aggregation data type (called ***type A***) whose base-type is another aggregation data type, the structure of the XML schema data type corresponding to ***type A*** shall be the structure specified in clause 7.2.2.4, where:

— ***base-instance*** If ***type B*** is an entity data type, ***base-instance*** shall be the qualified name of the corresponding complexEntity group (see 8.5.6.2). If ***type B*** is a SELECT data type, ***base-instance*** shall be the qualified name of the corresponding group (see 8.3.4.2). Otherwise, ***base-instance*** shall be the qualified name of the corresponding instance element (see 8.4).

All other definitions, notes and examples are as specified in clause 7.2.2.4.

### 8.2.2.5 Aggregates of ENUMERATION data types

Unless otherwise specified by Table 3, when the base-type of the EXPRESS aggregation data type is a defined data type whose fundamental type is an `ENUMERATION` type, the corresponding XML Schema data type shall be as specified in 8.2.2.3.

NOTE By definition, the representations of `ENUMERATION` data types do not contain whitespace, and the simple list form – names separated by whitespace – is an unambiguous representation.

See EXAMPLE in 7.2.2.5.

 

#### 8.2.2.6 Aggregates of SELECT data types

When the base-type of the EXPRESS aggregation data type is a defined data type whose fundamental type is a `SELECT` type, and the working select-list of the select type (see 8.3.4.1) contains only one data type, the base-type is considered to be that data type, instead of the `SELECT` type, and the subclause in Table 3 specified for that data type shall apply. In all other cases, when the base-type of the EXPRESS aggregation data type is a defined data type whose fundamental type is a `SELECT` type, the corresponding XML schema data type shall be a `complexType`.

If the underlying type of the defined data type is not the same as the fundamental type (that is, if the defined data type is a *specialization* of a `SELECT` type), the `complexType` shall be as specified in 8.2.2.2, using the defined data type as the base-type.

Otherwise, the `complexType` shall have the form specified in 7.2.2.6, where:

— ***base-group*** is the qualified name of the XML group or instance element that corresponds to the defined data type, as specified in 8.3.4.2.

— ***base-type*** is the qualified name of the XML data type corresponding to the base-type of the aggregate, that is, the defined data type whose fundamental type is the `SELECT` data type, as specified in 8.3.

— ***aggregate-type***, .***usage, group-or-element, lower-bound,*** and ***upper-bound*** are as defined in 7.2.2.6.

See EXAMPLE in 7.2.2.6.

#### 8.2.2.7 Aggregates of defined data types

When the base-type of the EXPRESS aggregation data type is a defined data type, that is, a data type defined by an EXPRESS type-declaration, the corresponding XML schema data type shall depend on the fundamental type of the defined data type. The subclause specified by Table 3, using the fundamental type as the base-type, shall apply. Within the appropriate subclause, the defined data type shall be considered to be the base-type.

Exception: If the defined data type, or any type of which it is a specialization, is explicitly mapped to an XML schema data type by a **map** configuration directive (see 10.2.13), the XML schema data type model for the EXPRESS aggregation data type shall depend on the XML schema data type specified by that **map** directive, as follows:

— If the XML schema data type specified by that **map** directive is a data type whose value representations contain no whitespace, the base-type shall be considered to be a simple type when consulting Table 3.

— Otherwise, the aggregation data type shall be mapped to the general form specified in 8.2.2.2.

#### 8.2.2.8 Aggregates of ENTITY data types

When the base-type of the EXPRESS aggregation data type is an entity data type, the corresponding XML schema data type has the list-of-references form (see 8.2.2.8.1) or the sequence of entity instances form (see 8.2.2.8.2), as specified in Table 3.

#### 8.2.2.8.1 List of references form

When the "list of references" form is chosen, the corresponding XML Schema data type shall be based on a simple list of XML `IDREF` values.

If the XML data type is the data type of an accessor attribute (see 8.6.3), it shall have the form:

```
<xs:simpleType>
 <xs:restriction base="xs:IDREFS">
     <xs:minLength value="lower-bound"/>
     <xs:maxLength value="upper-bound"/>
 </xs:restriction>
</xs:simpleType>
```

where:

— ***lower-bound*** is the lower bound on the EXPRESS aggregate. When the lower bound is given and does not evaluate to a constant, the value of the `minLength` facet shall be the value of the lower bound, represented as a decimal integer. Otherwise, the value of the `minLength` facet shall be '`0`'. If the value of `minLength` is '`0`', the `minLength` facet may be omitted.

— ***upper-bound*** is the upper bound on the EXPRESS aggregate. When the upper bound is given and evaluates to a constant, the `maxLength` facet shall appear and shall have the value of the upper bound, represented as a decimal integer. Otherwise, the `maxLength` facet shall be omitted.

NOTE If the EXPRESS aggregation type is an `ARRAY`, the value of the lower and upper bounds is `HiIndex(a) - LoIndex(a) + 1`, where `a` is the aggregation data type.

In any other case, the corresponding XML data type shall have the form:

```
<xs:complexType>
 <xs:simpleContent>
     <xs:restriction base="exp:Seq-IDREF">
         <xs:simpleType>
             <xs:restriction base="xs:IDREFS">
                 <xs:minLength value="lower-bound"/>
                 <xs:maxLength value="upper-bound"/>
             </xs:restriction>
         </xs:simpleType>
         <xs:attribute ref="exp:arraySize" usage />
         <xs:attribute ref="exp:itemType"  fixed="base-type"/>
         <xs:attribute ref="exp:cType"     fixed="aggregate-type" />
     </xs:restriction>
 </xs:simpleContent>
</xs:complexType>
```

where:

— ***lower-bound*** and ***upper-bound*** are as defined above.

— ***base-type*** shall be the qualified name of the XML schema data type that corresponds to the entity data type, as specified in 8.3.

— ***aggregate-type*** is the keyword for the EXPRESS aggregate type, one of: `"set"` for `SET`, `"bag"` for `BAG`, `"list"` for `LIST`, `"array"` for `ARRAY`, `"list-unique"` for `LIST OF UNIQUE`, `"array-unique"` for `ARRAY OF UNIQUE`. The list-of-references form is not used for `ARRAY OF OPTIONAL`.

 

— ***usage***: When the EXPRESS aggregate type is `ARRAY` and the upper and lower bounds on the array evaluate to constants, `arraySize` shall be declared `fixed="`***upper-bound***`"`, where ***upper-bound*** is as defined above. When the EXPRESS aggregate type is `ARRAY` and the upper and lower bounds on the array do not evaluate to constants, `arraySize` shall be declared `use="required"`. Otherwise, `arraySize` shall be declared `use="optional"`.

NOTE 1 This representation is the "list of values" form in 8.2.2.3, in all cases, except that the `itemType` for the `xs:list` is `xs:IDREF`, instead of ***base-type***. (The `itemType` attribute value is ***base-type***.)

NOTE 2 The accessor attribute form above, and the `simpleContent` model for the more general form, are equivalent to:

```
<xs:simpleType>
  <xs:restriction>
     <xs:simpleType>
       <xs:list itemType="xs:IDREF"/>
     </xs:simpleType>
     <xs:minLength value="lower-bound"/>
     <xs:maxLength value="upper-bound"/>
  </xs:restriction>
</xs:simpleType>
```

which is consistent with the forms in 8.2.2.3.

EXAMPLE

In EXPRESS:

```
ENTITY Pt3d;
  -- additional attributes skipped
END_ENTITY;

TYPE Pt3dSet = SET OF Pt3d;
END_TYPE;
```

In XML Schema:

```
<xs:complexType name="Pt3dSet">
   <xs:simpleContent>
     <xs:restriction base="exp:Seq-IDREF">
       <xs:simpleType>
         <xs:restriction base="xs:IDREFS"/>
       </xs:simpleType>
       <xs:attribute ref="exp:arraySize" use="optional"/>
       <xs:attribute ref="exp:itemType" fixed="Tns:Pt3d"/>
       <xs:attribute ref="exp:cType" fixed="set" />
     </xs:restriction>
   </xs:simpleContent>
</xs:complexType>
```

or (when the XML data type is the data type of an accessor attribute):

```
<xs:simpleType name="Pt3dSet">
  <xs:restriction base="xs:IDREFS"/>
</xs:simpleType>
```

#### 8.2.2.8.2 Sequence of entity instance elements form

When the "sequence of entity instance elements" form is chosen, the corresponding XML schema data type shall be a `complexType` having the form specified in 7.2.2.7 where:

— ***base-instance*** is the qualified name of the entity instance element corresponding to the base-type of the aggregate, as specified in 8.5.5.

— ***base-complexEntity-group*** is the qualified name of the complexEntity group corresponding to the base-type of the aggregate, as specified in 8.5.6.2.

— ***lower-bound, upper-bound, aggregate-type*** and ***usage*** are as defined in 7.2.2.7.

See EXAMPLE in 7.2.2.7.

### 8.2.3 Constructed data types

EXPRESS defines two kinds of data types, called "constructed data types", that can only occur in the definition of a defined data type. There are no XML Schema data types that correspond to the constructed data types as such, because in EXPRESS they cannot be the data type of anything.

#### 8.2.3.1 ENUMERATION data types

The XML schema data type that corresponds to an `ENUMERATION` data type is the one that corresponds to the EXPRESS defined data type. See 8.2.4 and 8.3.3.

#### 8.2.3.2 SELECT data types

The XML schema data type that corresponds to a `SELECT` data type is the one that corresponds to the EXPRESS defined data type. See 8.2.4 and 8.3.4.

### 8.2.4 Defined data types

For each EXPRESS defined data type, the corresponding XML schema data type is a named data type defined by an XML schema data type definition. The XML data type name and its definition shall be as specified in 8.3.

### 8.2.5 ENTITY data types

For each EXPRESS entity data type, the corresponding XML schema data type is a named data type defined by an XML schema data type definition. The XML data type name and its definition shall be as specified in 8.5.

## 8.3 XML Schema definitions and declarations for EXPRESS defined data types

An EXPRESS *defined data type* is a data type that is defined by an EXPRESS type declaration. The type declaration defines the data type in terms of an *underlying type*. The defined data type is said to be a *specialization* of the underlying type. The underlying type of an EXPRESS defined data type, however, may itself be an EXPRESS defined data type, which is defined by an EXPRESS type declaration to be based on another underlying type, and so on. Ultimately, there is a *fundamental type* that is not itself an EXPRESS defined data type, but rather a data type defined by the EXPRESS language.

NOTE 1 The terms *defined data type*, *underlying type*, *specialization*, and *fundamental type* are taken from ISO 10303-11:2003 and are used extensively in this Part of ISO 10303 in defining the mapping of defined data types and the encoding of their values.

For each EXPRESS defined data type that is declared in, or explicitly interfaced into the context schema, the derived XML schema shall contain an XML schema data type definition.

The XML schema data type definition shall depend on the underlying type of the EXPRESS defined data type, as follows:

— If the defined data type is explicitly mapped to an XML schema data type by the **map** configuration directive (see 10.2.13), the XML schema data type definition shall be as specified in 8.3.6.

— Otherwise, if the underlying type of the defined data type is a simple data type (`BINARY`, `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER`, `REAL`, or `STRING`), the XML schema data type definition shall be as specified in 8.3.1.

— If the underlying type of the defined data type is an aggregation data type (`ARRAY`, `BAG`, `LIST`, or `SET`), the XML schema data type definition shall be as specified in 8.3.2.

— If the underlying type of the defined data type is an `ENUMERATION` data type, the XML schema data type definition shall be as specified in 8.3.3.

— If the underlying type of the defined data type is a `SELECT` data type, the XML schema data type definition shall be as specified in 8.3.4.

— If the underlying type of the defined data type is another defined data type, the XML schema data type definition shall be as specified in 8.3.5.

The derived XML schema may also contain a corresponding instance element declaration, as specified in 8.4.3. The XML data type definition shall be independent of, and appear outside of, the instance element declaration.

NOTE 2 The underlying type of an EXPRESS defined data type cannot be an entity data type.

### 8.3.1 Simple underlying types

When the underlying type is a simple data type (`BINARY`, `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER`, `REAL`, or `STRING`), and no map directive applies to the defined data type, the corresponding XML schema data type definition shall be as specified in this subclause.

When the underlying type is a `BINARY` data type, the XML data type definition shall be as specified in 8.3.1.1.

When the underlying type is `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER` or `REAL`, the XML data type definition shall be as specified in 8.3.1.2.

When the underlying type is a `STRING` data type, the XML data type definition shall be as specified in 8.3.1.3.

#### 8.3.1.1 Definitions for `BINARY` data types

When the underlying type is an unconstrained `BINARY` data type, the XML data type definition shall have the form:

```
<xs:complexType name = "identifier">
 <xs:simpleContent>
  <xs:restriction base = "binary-type">
  </xs:restriction>
 </xs:simpleContent>
</xs:complexType>
```

where:

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 8.1.2, and

— ***binary-type*** is `exp:hexBinary`, or `exp:base64Binary` if **map="xs:base64Binary"** applies to the `BINARY` data type.

When the underlying type is a constrained `BINARY` data type (see 8.2.1.1.1), the XML data type definition shall have the form:

```
<xs:complexType name = "identifier">
  <xs:restriction base="binary-type">
    <xs:minLength value="minLength"/>
    <xs:maxLength value="maxLength"/>
    <xs:attribute name="extraBits" fixed="remainder" />
  </xs:restriction>
</xs:complexType>
```

where:

— ***identifier*** and ***binary-type*** are as above, and

— ***minLength***, ***maxLength*** and ***remainder*** are as specified in 8.2.1.1.1.

#### 8.3.1.2 Definitions for `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER` or `REAL`

When the underlying type is `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER` or `REAL`, the corresponding XML data type definition shall have the form specified in 7.3.1.2 where:

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 8.1.2, and

— ***underlying-type*** is the XML schema data type corresponding to the underlying type, as specified in 8.2.1.

#### 8.3.1.3 Definitions for `STRING` data types

When the underlying type is a `STRING` data type, the corresponding XML data type definition shall have one of the forms specifed in 7.3.1.3 where:

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 8.1.2, and

— ***minLength*** and ***maxLength*** (for a constrained `STRING` data type) are as specified in 7.3.1.3.

Exception: When the underlying type is an unconstrained `STRING` data type and the **notation** configuration directive applies to the defined data type, the XML data type definition shall have the form:

```
<xs:complexType name = "identifier">
 <xs:simpleContent>
    <xs:extension base = "xs:normalizedString">
   <xs:attribute ref="xml:notation" fixed="configured-notation"/>
    </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

where:

— ***identifier*** is as above, and

— ***configured-notation*** is the value given for the **notation** directive.

EXAMPLE For the EXPRESS declarations:

```
TYPE label = STRING;
END_TYPE;

TYPE clear_text = STRING;
END_TYPE;

TYPE userid = STRING (8) FIXED;
END_TYPE;
```

and the configuration directive:

```
<type select="Clear_text" notation="MIME:application/iso10303-21" />
```

The definitions in the derived XML schema are

```
<xs:simpleType name = "Label">
   <xs:restriction base = "xs:normalizedString">
   </xs:restriction>
</xs:simpleType>

<xs:complexType name = "Clear_text">
 <xs:simpleContent>
  <xs:extension base = "xs:normalizedString">
   <xs:attribute ref="xml:notation"
      fixed="MIME:application/iso10303-21"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>

<xs:simpleType name = "Userid">
  <xs:restriction base="xs:normalizedString">
    <xs:minLength value="8"/>
    <xs:maxLength value="8"/>
  </xs:restriction>
</xs:simpleType>
```

### 8.3.2 Aggregate underlying types

When the underlying type of the EXPRESS defined data type is an aggregation data type — a `SET`, `LIST`, `BAG` or `ARRAY` type — the XML Schema data type that corresponds to the underlying type depends on several factors, as specified in 8.2.2. The form of the XML Schema data type definition

for the defined data type depends on the form of the XML Schema data type corresponding to the underlying, but all of them are complexTypes.

Therefore, the corresponding XML data type definition shall have the general form:

```
<xs:complexType name="identifier">
 (body)
</xs:complexType>
```

where:

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 8.1.2, and

— ***(body)*** is the model of the complexType that corresponds to the underlying aggregation data type, as specified in 8.2.2.

NOTE For all practical purposes, the data type definition is identical to the corresponding data type, except that name="***identifier***" is added to the initial <xs:complexType> element.

### 8.3.3 ENUMERATION underlying types

When the underlying type of the EXPRESS defined data type is an ENUMERATION data type, the corresponding XML schema data type shall have the form specifed in 7.3.3, where:

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 8.1.2, and

— ***enum-item-1... enum-item-n*** are as specified in 7.3.3.

### 8.3.4 SELECT underlying types

When the underlying type of the EXPRESS defined data type is a SELECT data type, the corresponding XML schema data type shall depend on the number of data types in the working select list as specified in 8.3.4.1.

If the working select list contains no data types, the SELECT data type is not mapped to the derived XML schema.

If the working select list contains only one data type:

— the corresponding XML Schema data type shall be the XML Schema data type that corresponds to that data type, and

— the corresponding instance element shall be the instance element that corresponds to that data type.

Otherwise, the derived XML schema shall contain an XML group declaration, as specified in 8.3.4.2, and a complexType declaration, as specified in 8.3.4.3. The corresponding XML Schema data type shall be the complexType specified in 8.3.4.3. In addition, the derived XML schema may be required to contain an XML key definition corresponding to the SELECT data type, as specified in 8.3.4.4.

NOTE For a SELECT data type that has more than one type in the select list, there is no corresponding instance element (see 8.4). Instead, the group specified in 8.3.4.2 is the equivalent of the instance element for the SELECT data type.

 

#### 8.3.4.1 Working select list

The working select list for the SELECT data type shall be developed as follows:

— The working select list shall be initialized to the set of EXPRESS data types in the select-list of the SELECT data type.

— For each EXPRESS data type in the working select list that is a defined data type whose (immediate) underlying type is a SELECT type, the defined data type shall be deleted from the working select list and replaced by all of the data types in the select-list of its underlying SELECT type.

— The previous step shall be repeated until there are no defined data types in the working select list whose (immediate) underlying type is a SELECT type.

— For each EXPRESS entity data type that appears in the working select list and uses the inheritance-free mapping (see 8.5), all of the subtypes that appear in the corresponding subtypes group (if any, see 8.5.6.1) shall be added to the working select list. Exception: no data type shall be added to the working select list that is already on the working select list.

NOTE 1 Using the type graph associated with the entity data type (see 8.5.1), every leaf node that has a path to the node corresponding to the entity data type represents a possible subtype, and every node that lies on such a path represents a possible subtype.

— No data type that is already on the working select list shall be added to the working select list.

— All abstract entity data types (see 6.3) to which the inheritance-free mapping applies shall be deleted from the working select list.

NOTE 2 The working select list is the set of all EXPRESS data types whose instances are admissible as instances of the original SELECT data type. Nested SELECT data types are expanded into the list.

NOTE 3 A defined data type whose underlying type is another defined data type is *not* replaced/expanded in the working select list, even when its fundamental type is a SELECT data type. A defined data type whose underlying type is another defined data type is always treated as a specialization and that information is retained in the XML schema type model.

See Examples in 7.3.4.1.

#### 8.3.4.2 XML group declaration for SELECT data types

When the underlying type of the EXPRESS defined data type is a SELECT data type that has a working select list containing more than one data type, the derived XML schema shall contain a group declaration that corresponds to the EXPRESS defined data type.

The name of the group shall be derived from the EXPRESS identifier for the defined data type as specified in 8.1.3.

The contents of the group is derived from the working select list of the SELECT data type as specified in 8.3.4.1.

The group shall contain a single choice element. For each EXPRESS data type in the working select list, the choice element shall contain an element declaration for the instance element corresponding to that EXPRESS data type (as specified in 8.4), or for each of the instance elements found in its subtype group (see 8.5.6.1). The element declaration shall have a ref attribute whose value is the qualified name of the instance element.

That is, the group declaration shall have the form specified in 7.3.4.2 where:

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 8.1.2.

— ***select-list-item-1*** … ***select-list-item-n*** and ***complexEntity-usage*** are as specified in 7.3.4.2.

NOTE Since every data type in the working select list is a named data type, the qualified names of the instance elements will be derived from the EXPRESS identifiers for the data types as specified in 8.1.3.

See EXAMPLE in 7.3.4.2.

### 8.3.4.3 XML data type definition for SELECT data types

When the underlying type of the EXPRESS defined data type is a `SELECT` data type that has a working select list containing more than one data type, the corresponding XML schema data type shall be a `complexType` whose content model is the `group` corresponding to the `SELECT` data type, as specified in 8.3.4.2.

The `complexType` definition shall have the form specified in 7.3.4.3, where

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 8.1.2.

— ***group-name*** is the qualified name of the group corresponding to the `SELECT` data type, as specified in 8.3.4.2.

— ***exp:arraySize-usage***: If any of the instance elements contained in the `SELECT` group (see 8.3.4.2) represent anonymous aggregates, <xs:element ref="`exp:arraySize`" use="optional"/>. Otherwise, the ***exp:arraySize*** element shall be omitted.

NOTE 1 Each instance element in the group corresponding to the `SELECT` data type contains the optional XML attribute `path`, as specified in 8.4.5. The `path` attribute specifies the types in the select list that are instantiated in that value.

NOTE 2 In order to allow accessor elements (see 8.6.4) to refer to a value of the `SELECT` data type (instead of containing it), the `complexType` contains the `ref` attribute, and allows the `choice` group to be omitted. 9.4.3 requires either the ref attribute to be present or the content to contain an instance of the `choice` group.

See EXAMPLE 7.3.4.3.

### 8.3.4.4 XML identity constraints for SELECT data types

If the configuration option **generate-keys="true"** applies to the derived XML schema, and there is an EXPRESS attribute anywhere in the context schema such that the data type of the EXPRESS attribute is the defined (`SELECT`) data type, and either

— the configuration directive **exp-attribute="attribute-content"** applies to that attribute, or

— the configuration directive **exp-attribute="attribute-tag"** applies to that attribute, and **content="ref"** or **content="unspecified"** applies to that attribute,

then the derived XML schema shall contain a `key` definition corresponding to the defined data type.

 

NOTE 1 **exp-attribute="attribute-tag"** and **content="unspecified"** are the defaults.

The `key` definition shall have the form:

```
<xs:key name="schema___identifier-key">
 <xs:selector xpath="select-list-item-1 | … | select-list-item-n"/>
 <xs:field xpath="@id"/>
</xs:key>
```

where:

— ***schema*** is derived from the identifier for the EXPRESS schema as specified in 8.1.2,

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 8.1.2, and

— ***select-list-item-1*** … ***select-list-item-n*** are the qualified names of the instance elements corresponding, as specified in 8.4, to each data type in the working select list.

NOTE 2 The XML `key` definition corresponding to the `SELECT` data type is the equivalent to the `key` definitions that correspond to instance elements (see 8.4.6), in much the same way that the XML `group` declaration corresponding to the `SELECT` data type is equivalent to the instance element declarations.

EXAMPLE Continuing the example in 8.3.4.2 above, the XML `key` declaration would be:

```
<xs:key name="Tns___S-key">
  <xs:selector xpath="Tns:Label-wrapper
    | Tns:Length_measure-wrapper
    | Tns:Pipe_entity
    | Tns:Traffic_light-wrapper" />
  <xs:field xpath="@id" />
</xs:key>
```

### 8.3.5 Defined data type underlying type

When the underlying type of the EXPRESS defined data type is another defined data type, the corresponding XML schema data type model shall depend on the XML schema data type corresponding to the fundamental type. Exception: If the underlying type is explicitly mapped to an XML schema data type by the **map** configuration directive (see 10.2.13), the XML schema data type specified in the directive shall be considered "the XML schema data type corresponding to the fundamental type" for all purposes in this clause.

If the XML schema data type corresponding to the fundamental type is a `simpleType`, the XML schema data type definition shall have the form specified in 7.3.5 for `simpleTypes`, and if the XML schema data type corresponding to the fundamental type is a `complexType`, the XML schema data type definition shall have the form specified in 7.3.5 for `complexTypes`, where in either case:

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 8.1.2.

— ***(content model for the fundamental type)***: When the `complexType` corresponding to the underlying type has complex content is the XML particle that describes the content of the XML data type that corresponds to the fundamental of the defined data type: the `sequence` particle specified in 8.2.2, if the fundamental type is an aggregation data type, or the `group` specified in 8.3.4.2, if the fundamental type is a `SELECT` type.

— ***underlying-type*** and ***contentUsage*** are as specified in 7.3.5.

NOTE The content model need not include the XML attributes of the fundamental type.

See EXAMPLEs in 7.3.5

### 8.3.6 Defined data types mapped by map configuration directive

If the defined data type is explicitly mapped to an XML schema data type by a **map** configuration directive (see 10.2.13), the XML schema data type model shall be a `simpleType` derived by restriction from the XML schema data type specified in the **map** directive. The XML schema data type definition shall have the form:

```
<xs:simpleType name="identifier">
 <xs:restriction base="configured-type"/>
</xs:simpleType>
```

where

— ***identifier*** is derived from the EXPRESS identifier for the defined data type as specified in 8.1.2, and

— ***configured-type*** is the XML schema data type specified in the **map** configuration directive.

EXAMPLE For the EXPRESS declaration:

```
TYPE identifier = STRING;
END_TYPE;
```

Given the configuration directive:

```
<type select="identifier" map="xs:Name">
```

The definition in the derived XML schema is

```
<xs:simpleType name = "identifier">
   <xs:restriction base = "xs:Name">
   </xs:restriction>
</xs:simpleType>
```

## 8.4 Instance elements corresponding to EXPRESS data types

An *instance element* is an XML element that is named for an EXPRESS data type and represents a value of that EXPRESS data type.

A *by-value instance element* is an instance element whose content is the data value.

A *by-reference instance element* is an instance element that has no content but contains a reference to a by-value instance element whose content is the data value.

NOTE 1 Most instance elements for data types that are not entity data types will be by-value instance elements.

The instance element corresponding to an EXPRESS entity data type shall be declared as specified in 8.5.

The instance element corresponding to an EXPRESS data type that is neither an entity data type nor a select type shall be declared when any of the following applies:

— the EXPRESS data type appears as the base-type of an aggregation data type that uses the sequence-of-elements representation (see 8.2.2);

— the EXPRESS data type appears in the working select list of a `SELECT` data type (see 8.3.4);

— the value of the EXPRESS data type can appear by-reference (see 8.4.7).

Otherwise, the instance element may be declared in the derived XML schema, but it is not required.

Exceptions: The instance element shall not be declared in the derived XML schema when it is declared in the Base XML Schema, as indicated in 8.4.1. No instance element shall be declared for a defined data type whose underlying type is a `SELECT` data type.

NOTE 2 For a defined data type whose underlying type is a `SELECT` data type, the XML `group` declared as specified in 8.3.4.2 is the equivalent of the instance element. But a defined data type whose *underlying* type is a defined data type and whose *fundamental* type is a `SELECT` data type (that is, a specialization of a `SELECT` data type), *does* have an instance element. The above exception does not apply to such data types.

NOTE 3 The requirement for the instance element declaration does *not* apply to a data type that only appears as the data type of an EXPRESS attribute with **exp-attribute="attribute-tag"**, unless **content="ref"** or **content="unspecified"** also applies.

The instance element declaration shall be as specified according to the EXPRESS data type in 8.4.1 through 8.4.4. In addition, XML identity constraints for the instance element shall be as specified in 8.4.6.

NOTE 4 The typical usage for a value of an EXPRESS data type that is not an entity data type is as the value of an entity attribute. In such cases, the EXPRESS data type value is represented as the content of an XML element or an XML attribute that represents the EXPRESS entity attribute (see 8.6). However, a value of an EXPRESS data type may also appear in an instance document as a component of an aggregation value, or as a value of a `SELECT` data type. In addition, when required/permitted by configuration directives, a value that occurs as the value of an EXPRESS attribute may be encoded as an "independent" instance element, and referenced by the attribute(s) of which it is the value. In these contexts, the value may be represented by an instance element.

### 8.4.1 Instance elements for simple data types

The Base XML Schema (see Annex C) declares an instance element for each EXPRESS simple data type. The data type of the instance element is the XML Schema data type that corresponds to the EXPRESS data type, and the instance element adds the `instanceAttributes` attribute group (see 8.4.5).

In some cases, additional instance elements for simple types may be declared in the derived XML schema. See below.

#### 8.4.1.1 BINARY data type

The instance element corresponding to the EXPRESS `BINARY` data type shall be the `exp:hexBinary-wrapper` element. Exceptions:

— If the EXPRESS `BINARY` data type is mapped to the `exp:base64Binary` data type by a **map** configuration directive (see 10.2.13), the corresponding instance element shall be the `exp:base64Binary-wrapper` element.

— If the EXPRESS BINARY data type specifies a maximum or FIXED length, the corresponding instance element shall be as specified in 8.4.1.1.1.

The exp:base64Binary-wrapper instance element is declared in the Base XML Schema as follows:

```
<xs:element name="base64Binary-wrapper" nillable="true" >
 <xs:complexType>
    <xs:simpleContent>
       <xs:extension base="exp:base64Binary">
          <xs:attributeGroup ref="exp:instanceAttributes"/>
       </xs:extension>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

The exp:hexBinary-wrapper instance element is declared in the Base XML Schema and in 7.4.1.1.

#### 8.4.1.1.1 Constrained BINARY instance element

When required, the instance element corresponding to an EXPRESS BINARY data type for which a maximum or FIXED length is specified (see 8.2.1.1.1) shall be declared in the derived XML schema as follows:

```
<xs:element name="typename.minBits.bits-wrapper"
         nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="binary-type">
        <xs:attributeGroup ref="exp:instanceAttributes"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

where:

— ***typename*** is Binary, or base64Binary if **map="xs:base64Binary"** applies to the BINARY data type.

— ***binary-type*** is the qualified name of the XML schema data type declared for the constrained BINARY data type as specified in 8.2.1.1.1.

— ***bits*** is the fixed or maximum length in bits specified for the EXPRESS data type.

— ***minBits*** shall be set to ***bits*** if the length specified for the EXPRESS BINARY data type is declared FIXED; otherwise, it shall be 0.

See EXAMPLE in 7.4.1.1.1.

### 8.4.1.2 BOOLEAN data type

The instance element corresponding to the EXPRESS BOOLEAN data type is the exp:boolean-wrapper element. The exp:boolean-wrapper element is declared in the Base XML Schema and in 7.4.1.2.

 

#### 8.4.1.3 INTEGER data type

The instance element corresponding to the EXPRESS `INTEGER` data type is the `exp:long-wrapper` element. Exception: If the EXPRESS `INTEGER` data type is mapped to the `xs:decimal` data type by a **map** configuration directives (see 10.2.13), the corresponding instance element shall be the `exp:integer-wrapper` element.

The `exp:integer-wrapper` instance element is declared in the Base XML Schema, as follows:

```
<xs:element name="integer-wrapper" nillable="true" >
 <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:integer">
            <xs:attributeGroup ref="exp:instanceAttributes"/>
      </xs:extension>
     </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

The `exp:long-wrapper` instance element is declared in the Base XML Schema and in 7.4.1.3.

#### 8.4.1.4 LOGICAL data type

The instance element corresponding to the EXPRESS `LOGICAL` data type is the `exp:logical-wrapper` element. The `exp:logical-wrapper` element is declared in the Base XML Schema and in 7.4.1.4.

#### 8.4.1.5 NUMBER and REAL data types

The instance element corresponding to the EXPRESS `NUMBER` data type is the `exp:decimal-wrapper` element. Exception: If the EXPRESS `NUMBER` data type is mapped to the `xs:double` data type by a **map** configuration directives (see 10.2.13), the corresponding instance element shall be the `exp:double-wrapper` element.

The instance element corresponding to the EXPRESS `REAL` data type is the `exp:double-wrapper` element. Exception: If the EXPRESS `REAL` data type is mapped to the `xs:decimal` data type by a **map** configuration directives (see 10.2.13), the corresponding instance element shall be the `exp:decimal-wrapper` element.

These instance elements are declared in the Base XML Schema, and in 7.4.1.5 and 7.4.1.6.

#### 8.4.1.6 STRING data type

The instance element corresponding to the EXPRESS `STRING` data type is the `exp:string-wrapper` element. Exceptions:

— If the EXPRESS `STRING` data type is mapped to another XML data type by a **map** configuration directive (see 10.2.13), the corresponding instance element shall be as specified in 8.4.1.6.2.

— If the EXPRESS `STRING` data type specifies a maximum or `FIXED` length, the corresponding instance element shall be as specified in 8.4.1.6.1.

The `exp:string-wrapper` element is declared in the Base XML Schema and in 7.4.1.7.

#### 8.4.1.6.1 Constrained STRING instance element

When required, the instance element corresponding to an EXPRESS STRING data for which a maximum or FIXED length is specified (see 8.2.1.7.1), shall be declared in the derived XML schema as shown in 7.4.1.7.1, where:

— ***stringtype*** the local part of the name of the XML schema datatype that corresponds to the constrained STRING data type as specified in 8.2.1.7.1.

— ***Tns:stringtype*** is the qualified name of the XML schema datatype that corresponds to the constrained STRING data type.

See EXAMPLE in 7.4.1.7.1.

#### 8.4.1.6.2 XML data types for EXPRESS STRING data types

Table 4 specifies the XML data types that the EXPRESS STRING data type may be mapped to using the **map** configuration directive, and their corresponding instance elements (defined in Annex C).

**Table 4 — Instance elements for STRING data types mapped to XML data types**

| XML data type | Corresponding instance element |
|---|---|
| xs:normalizedString (default) | exp:string-wrapper |
| xs:string | exp:generalString-wrapper |
| xs:language | exp:language-wrapper |
| xs:Name | exp:Name-wrapper |
| xs:QName | exp:QName-wrapper |
| xs:NMTOKEN | exp:NMTOKEN-wrapper |
| xs:anyURI | exp:anyURI-wrapper |

### 8.4.2 Instance elements for anonymous aggregation data types

When an instance element representing an anonymous aggregation data type is required, as specified in 8.4 above, it shall be declared as specified in this subclause.

The instance element for an anonymous aggregation type shall correspond to *all* anonymous aggregation data types in the context schema that have the same base-type, regardless of the kind of aggregate (SET, LIST, BAG or ARRAY). Exception: If two anonymous aggregations of the same base-type are required by configuration directives to use different structures (a list of values structure and a sequence of elements structure), the XML schema shall contain two instance element declarations, one for each of the two structures.

Because the instance element represents a value of any aggregation data type with the same base-types, it shall include all of the ref, arraySize, itemType and cType attributes defined in 8.2.2.1.

 

Because the instance element may be referenceable (see 8.4.7) or represent a "row" in a higher-level aggregate value, the instance element shall include the `instanceAttributes` group (see 8.4.5).

The XML schema data type in the instance element declaration shall depend on the XML schema data type corresponding to the aggregation data type, as specified in 8.2.2, but modified as follows:

— If the XML data type corresponding to the aggregation data type has the list-of-values form, as specified in 8.2.2, the instance element declaration shall be as specified in 8.4.2.1 below.

— If the XML data type corresponding to the aggregation data type has the sequence of elements form, as specified in 8.2.2, the instance element declaration shall be as specified in 8.4.2.2 below.

— If the XML data type corresponding to the aggregation data type has the list of references form, as specified in 8.2.2.8.1, the instance element declaration shall be as specified in 8.4.2.3 below.

— If the XML data type corresponding to the aggregation data type has the sequence-of-rows form, as specified in 8.2.2.4, the instance element declaration shall be as specified in 8.4.2.2 for the sequence-of-elements form, where the base-type is the data type of the "row", that is, the nested aggregation data type.

NOTE 1 When the outer aggregation data type has the sequence-of-rows form, an instance element is always required for the nested aggregation data type, regardless of its form.

— If the XML data type corresponding to the aggregation data type uses the multi-dimensional-array form, as specified in 8.2.2.4, the instance element declaration shall be as specified in 8.4.2.2 for the sequence-of-elements form, except that the base-type shall be considered to be the deepest underlying type of the multi-dimensional array structure, for all purposes.

NOTE 2 As a consequence the sequence-of-elements instance element represents all aggregations of the (deepest) base-type instance element, regardless of the number of dimensions.

NOTE 3 As a consequence the list-of-values instance element represents all serialized aggregations of values of the (deepest) base-type, regardless of the number of dimensions.

### 8.4.2.1 Instance elements for list-of-values form

If the XML schema data type corresponding to the aggregation data type uses the list-of-values form (see 8.2.2.3), the content model for the instance element shall be a simple list, extended by the XML instance attributes and the XML aggregate attributes.

The instance element declaration shall have the form specified in 7.4.2.1, where:

— ***element-name***, the XML name of the instance element, shall have the form: `Seq-`***base***, where ***base*** is the local part of the name of the XML Schema data type that corresponds, as specified in 8.2, to the base-type of the aggregation data type.

— ***unrestricted-type*** shall be the qualified name of the unrestricted XML data type that supports all aggregations of the base-type of the aggregation data type, as specified in 8.2.2.3.3.

NOTE 1 The instance element corresponding to the base-type is never used in this representation.

NOTE 2 There are many required behaviours of the XML attributes of this instance element that are declared to be `use="optional"`. These behaviours are specified in 9.8.

See EXAMPLE in 7.4.2.1.

### 8.4.2.2 Instance elements for all sequence-of-elements forms

If the XML schema data type corresponding to the aggregation data type uses the sequence-of-elements form (see 8.2.2.2), or the sequence-of-rows form (see 8.2.2.4.1) or the multi-dimensional array form (see 8.2.2.4.2), the instance element declaration shall have the form specified in 7.4.2.2, where:

— ***element-name***, the XML name of the instance element, shall have the form: `Seq-`***base***, where ***base*** is the local part of the ***base-instance*** name (see below).

NOTE 1 For the sequence-of-rows form, where the base-type is a nested anonymous aggregation data type, this specification is recursive. That is, the XML schema data type name that corresponds to the most deeply nested aggregation data type will have the form `Seq-`***base***, where ***base*** is the deepest base type, and the XML schema data type name of the next outer anonymous aggregation data type will have the form `Seq-seq-`***base***, and so on.

— ***base-instance*** : Let the term "base-type" refer to the EXPRESS base-type of the aggregation data type for the sequence-of-elements or sequence-of-rows form, or to the deepest underlying type for the multi-dimensional array form. Then if base-type is an entity data type, ***base-instance*** shall be the qualified name of the corresponding complexEntity group (see 8.5.6.2). If base-type is a SELECT data type, ***base-instance*** shall be the qualified name of the corresponding group (see 8.3.4.2). Otherwise, ***base-instance*** shall be the qualified name of the corresponding instance element (see 8.4).

NOTE 2 When the instance element represents a sequence-of-rows structure, the base-type is the data type of the "row", that is, the nested aggregation data type. When the instance element represents a multi-dimensional array structure, the base-type is the deepest underlying type, as specified in 8.2.2.4.

— ***element-or-group*** shall be `xs:element`, if ***base-instance*** is an element, or `xs:group`, if ***base-instance*** is a group.

NOTE 3 The XML data type that corresponds to the aggregation data type, as specified in 8.2.2, is not actually used in this declaration.

NOTE 4 Although the `ref`, `cType` and `arraySize` attributes are declared `use="optional"`, clause 9.8 requires them to appear in certain circumstances, in particular when the instance element represents a multi-dimensional aggregate value.

EXAMPLE

The EXPRESS anonymous data type:

```
LIST [1:?] OF ARRAY [1:3] OF REAL
```

With configuration directive flatten="false";

The instance elements declaration will be:

```
<xs:element name="Seq-double-wrapper" nillable="true">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="exp:double-wrapper" minOccurs="0"
           maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup ref="exp:instanceAttributes"/>
    <xs:attribute use="optional" type="xs:IDREF" name="ref"/>
    <xs:attribute fixed="exp:double-wrapper" ref="exp:itemType"/>
    <xs:attribute ref="exp:arraySize" use="optional"/>
```

 

```
      <xs:attribute ref="exp:cType" default="set"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="Seq-seq-double-wrapper" nillable="true">
    <xs:complexType>
          <xs:sequence>
            <xs:element ref="Tns:Seq-double-wrapper"
                 minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute ref="exp:itemType"
              fixed="Tns:Seq-double-wrapper"/>
          <xs:attribute use="optional" type="xs:IDREF" name="ref"/>
          <xs:attribute ref="exp:arraySize"  use="optional"/>
          <xs:attribute ref="exp:cType" default="set"/>
          <xs:attributeGroup ref="exp:instanceAttributes" />
    </xs:complexType>
  </xs:element>
```

NOTE When **flatten= "false"** applies to the nested aggragate, the sequence-of-rows form applies.

#### 8.4.2.3 Instance elements for list-of-references form

If the XML schema data type corresponding to the aggregation data type uses the list of references form (see 8.2.2.8.1), the content model for the instance element shall be a simple list, extended by the XML instance attributes and the XML aggregate attributes.

The instance element declaration shall have the form:

```
<xs:element name="element-name" nillable="true" >
 <xs:complexType>
    <xs:simpleContent>
       <xs:restriction base="exp:Seq-IDREF-wrapper">
          <xs:attribute ref="exp:itemType" fixed="actual-base-type" />
       </xs:restriction>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

where:

— ***element-name*** shall have the form: `Seq-`***base***`-ref`, where ***base*** is the local part of the name of the XML Schema data type that corresponds, as specified in 8.2, to the entity data type.

— ***actual-base-type***, the `itemType` attribute value, shall be the qualified name of the XML data type that corresponds to the entity data type, as specified in 8.2.

NOTE 1 The actual XML representation is a simple list of references to the entity instance elements, with the addition of the XML instance attributes and the aggregation attributes. This representation is analagous to the simple lists described in 8.4.2.1. The XML Schema structure here is different, because the corresponding unrestricted data type is `exp:Seq-IDREF` and it doesn't specify a value for `exp:itemType`.

NOTE 2 There are many required behaviours of the XML attributes of this instance element that are declared to be `use="optional"`. These behaviours are specified in 9.8.

EXAMPLE

For the EXPRESS anonymous data type:

```
ARRAY [1:3] OF Pt3d;
```

The instance element declaration will be:

```
<xs:element name="Tns:Seq-Ptd3d-ref" nillable="true" >
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="exp:Seq-IDREF-wrapper">
        <xs:attribute ref="exp:itemType" fixed="Tns:Pt3d" />
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

### 8.4.3 Instance elements for defined data types

For each EXPRESS defined data type that appears as the base-type of an aggregation data type or in the select-list of a `SELECT` data type, the derived XML schema shall contain the declaration for an instance element. Exception: There shall be no instance element that corresponds to a defined data type whose underlying type is a `SELECT` data type; the corresponding XML schema object is a group (see 8.3.4.2).

NOTE 1 When the underlying type is a `SELECT` type and the working select list contains more than one data type, the XML schema object corresponding to the defined data type is a group (see 8.3.4.2). When the working select list contains only one data type, the instance element for the sole choice is used for the defined data type (see 8.3.4).

NOTE 2 For a defined data type whose fundamental type is a `SELECT` data type but whose *underlying type* is not, there is an instance element as specified in the subclause.

The instance element declaration shall have one of the forms specified in 7.4.3, dependent upon whether the XML schema data type corresponding to the defined data type is a `complexType` or a `simpletype`. In both cases:

— ***type-name*** is the qualified name of the XML schema data type corresponding to the EXPRESS defined data type (see 8.1.3).

— ***local-type-name*** and ***element-name*** are as specified in 7.4.3.

See EXAMPLE in 7.4.3.

### 8.4.4 Instance elements for entity data types

For an EXPRESS entity data type, the corresponding instance element, if any (see 8.5), shall be as specified in 8.5.5.

### 8.4.5 Instance element attributes

All instance elements shall have a `complexType` that includes the `instanceAttributes` XML attribute group. The attribute group is defined in the Base XML Schema and is shown in 7.4.5.

The attributes are as defined in 7.4.5.

 

### 8.4.6 XML identity-constraints for instance elements

This subclause applies only to a derived XML schema to which the configuration directive **generate-keys="true"** applies (see 10.2.23). If **generate-keys="false"** (the default) applies, the derived XML schema shall contain no identity constraints for EXPRESS data types that are not entity data types.

This subclause specifies the XML identity constraints for instance elements corresponding to EXPRESS data types that are not entity data types. XML identity-constraints for entity data types are specified in 8.5.9.

If the EXPRESS data type to which the instance element corresponds is referenceable (see 8.4.7), the derived XML schema shall contain a corresponding XML `key` definition, as specified in 8.4.6.1. Exception: If the EXPRESS datatype is an unconstrained simple data type, the corresponding `key` definition appears in the Base XML Schema, as described in 8.4.6.2, and shall not appear in the derived XML schema.

A reference to an instance element may be contained in either:

— a by-reference instance element (see 9.10.2) that corresponds to the same EXPRESS data type, or

— a by-reference accessor element (see 9.4.3.1) or a by-reference accessor attribute (see 9.4.2) that corresponds to an EXPRESS attribute whose data type is the EXPRESS data type.

The unit of serialization can contain by-reference instance elements if **content="ref"** or **content="unspecified"** applies to any use of the EXPRESS data type as either:

— the base-type of an aggregation type that is mapped using the sequence-of-elements form, or

— the data type of an EXPRESS attribute to which **exp-attribute="double-tag"** applies.

If a unit of serialization conforming to the XML schema and the configuration directives can contain by-reference instance elements corresponding to the EXPRESS data type, the derived XML schema shall contain a corresponding XML `keyref` definition, as specified in 8.4.6.3. Exception: If the EXPRESS datatype is an unconstrained simple data type, the corresponding `keyref` definition appears in the Base XML Schema, as described in 8.4.6.2, and shall not appear in the derived XML schema.

NOTE 1 If the EXPRESS data type is a *constrained* `BINARY` or `STRING` data type, the `key` and `keyref` definitions are not contained in the Base XML Schema. The instance element definition appears in the derived XML schema (see 8.4.1.1.1 and 8.4.1.6.1), and the corresponding `key` and keyref definitions are as specified in 8.4.6.1 and 8.4.6.3.

NOTE 2 When the instance element may be referenced by an accessor element or attribute, an additional `keyref` definition is created for that accessor. See 8.6.3 and 8.6.4.

NOTE 3 XML `key` and `keyref` components corresponding to the EXPRESS data type are declared in order to ensure that a by-reference element or attribute references a by-value element of the appropriate type. A value can appear as the value of an EXPRESS attribute or in an aggregate value that is, or is contained in, the value of an EXPRESS attribute. When the configuration directives permit a data value to appear "by-reference", the XML instance document may contain two corresponding instance elements (see 9.10) – a by-reference element in the content of the accessor element corresponding to the EXPRESS attribute, and an independent by-value element that is an immediate child of the unit of serialization. The by-value element contains an `id` XML attribute whose value is a local XML identifier for that value within the unit of serialization. The by-reference element contains a `ref` XML attribute whose value is that local XML identifier. XML schema identity constraints (on the `IDREF` data type of the `ref` attribute) require that there be an element whose `id` value matches the value of the `ref` attribute, which ensures "referential integrity". The purpose of the definitions in

this clause is to provide XML schema declarations that specify that the XML element whose `id` value matches the `ref` value shall be a by-value instance element for the same data type.

#### 8.4.6.1 Key definitions for non-entity instance elements

For a defined data type, a constrained `BINARY` or `STRING` data type, or an anonymous aggregation data type, the `key` definition shall have the form:

```
<xs:key name="schema___instance-name-key">
  <xs:selector xpath="Tns:instance-name"/>
 <xs:field xpath="@id"/>
</xs:key>
```

where:

— ***schema*** is the identifier for the EXPRESS schema, mapped as specified in 8.1.2.

— ***instance-name*** is the local part of the name of the instance element corresponding to the EXPRESS data type. If the EXPRESS data type is a defined data type, the name shall be as specified in 8.4.3. If the EXPRESS data type is a constrained `BINARY` data type or `STRING` data type, the name shall be as specified in 8.4.1.1.1 or 8.4.1.6.1, respectively. If the EXPRESS data type is an anonymous aggregation data type, the name shall be as specified in 8.4.2.

— ***Tns:instance-name*** is the fully qualified name of the instance element corresponding to the entity data type.

NOTE Since by-value instance elements that will be referenced only occur in the immediate content of the unit of serialization element (see 9.10.2), the `xs:selector xpath` expression for the `key` refers to instance elements that are immediate children of the unit of serialization element. Such elements are required (by 9.10.2) to have an `id` attribute.

EXAMPLE

For the EXPRESS anonymous data type appearing in schema `APPLICATION_SCHEMA`:

```
ARRAY [1:3] OF STRING
```

the XML instance element declaration will be:

```
<xs:element name="Tns:Seq-string-wrapper">
...
</xs:element>
```

as described in 8.4.2 above. The corresponding key and keyref declarations would be:

```
<xs:key name="Application_schema___Seq-string-wrapper-key">
  <xs:selector xpath="Tns:Seq-string-wrapper"/>
  <xs:field xpath="@id"/>
</xs:key>

<xs:keyref name="Application_schema___Seq-string-wrapper-keyref"
     refer="Tns:Application_schema___Seq-string-wrapper-key">
  <xs:selector xpath=".//Tns:Seq-string-wrapper"/>
  <xs:field xpath="@ref"/>
</xs:keyref>
```

 

#### 8.4.6.2 Key definitions for simple data types

If the EXPRESS datatype is an unconstrained simple data type, the corresponding `key` definitions, and the `keyref` declarations for the corresponding instance elements, appear in the Base XML Schema. They shall not appear in the derived XML schema.

Table 5 summarizes the `key` definitions in the Base XML Schema.

**Table 5 — XML key names for anonymous EXPRESS data types**

| EXPRESS data type | XML data type | Instance element name | Key name |
|---|---|---|---|
| BINARY | exp:hexBinary | exp:hexBinary-wrapper | exp:hexBinary-key |
| BINARY | exp:base64Binary | exp:base64Binary-wrapper | exp:base64Binary-key |
| BOOLEAN | xs:boolean | exp:boolean-wrapper | exp:boolean-key |
| INTEGER | xs:long | exp:long-wrapper | exp:long-key |
| INTEGER | xs:decimal | exp:integer-wrapper | exp:integer-key |
| LOGICAL | exp:logical | exp:logical-wrapper | exp:logical-key |
| NUMBER | xs:decimal | exp:decimal-wrapper | exp:decimal-key |
| NUMBER | xs:double | exp:double-wrapper | exp:double-key |
| REAL | xs:decimal | exp:decimal-wrapper | exp:decimal-key |
| REAL | xs:double | exp:double-wrapper | exp:double-key |
| STRING | xs:normalizedString | exp:string-wrapper | exp:string-key |

#### 8.4.6.3 Keyref definitions for by-reference instance elements

If a unit of serialization conforming to the XML schema and the configuration directives can contain by-reference instance elements corresponding to the EXPRESS data type, the derived XML schema shall contain a corresponding XML `keyref` definition. The `keyref` definition shall have the form:

```
<xs:keyref name="schema___instance-name-keyref" refer="keyname" >
 <xs:selector xpath=".//Tns:instance-name" />
 <xs:field xpath="@ref" />
</xs:keyref>
```

where:

— ***schema*** and ***instance-name*** are as specified in 8.4.6.1 above.

— ***keyname*** is the fully qualified name of the corresponding `key` component, as specified in 8.4.6.1, that is, "***Tns:schema___instance-name***`-key`", where ***Tns*** is the appropriate prefix for the derived XML schema.

NOTE The `xs:selector xpath` expression for the `keyref` refers to any by-reference instance element for the data type (any instance element with the `ref` attribute, see 9.10.2) appearing anywhere in the content of the unit of serialization element.

### 8.4.7 Referenceable instances

An EXPRESS data type is said to be *referenceable* if any element in the unit of serialization can contain a reference to a value of the EXPRESS data type in lieu of the value itself.

Because there are different defaults for the mapping of different EXPRESS data types, the practical determination that a data type is referenceable varies with the data type.

#### 8.4.7.1 Referenceable entity instances

In the default XML schema, every EXPRESS entity data type is referenceable unless none of its instances can be (part of) the value of any attribute in a unit of serialization conforming to the context schema.

For a configured XML schema, the following rules determine whether an EXPRESS entity data type is referenceable:

— If any value of the EXPRESS data type is a valid value of an EXPRESS attribute to which **exp-attribute="attribute-content"** applies, the entity data type is referenceable.

— Otherwise, if **exp-type="value"** applies to the entity data type, it is **not** referenceable.

— Otherwise, if any value of the EXPRESS data type is a valid value of an EXPRESS attribute to which **content="ref"** or **content="unspecified"** applies, the entity data type is referenceable.

— Otherwise, if any value of the EXPRESS data type is a valid value of a component of an aggregate value to which **content="ref"** or **content="unspecified"** applies, the entity data type is referenceable.

— Otherwise, the entity data type is not referenceable.

NOTE **exp-attribute="attribute-content"** forces the value of the attribute to be a reference to the entity instance. Any other use of the entity is permitted to be a reference unless **content="value"** applies to that use or **exp-type="value"** applies to the entity data type generally.

#### 8.4.7.2 Referenceable non-entity instances

In the default XML schema, every EXPRESS data type is referenceable.

For a configured XML schema, an EXPRESS data type that is not an entity data type is referenceable if any of the following applies:

— **exp-type="root"** applies to the EXPRESS data type,

 

— the EXPRESS data type is the data type of, or appears in the working select list of a SELECT data type that is the data type of, some EXPRESS attribute in the context schema to which **content="ref"** applies,

— the EXPRESS data type is the data type of, or appears in the working select list of a SELECT data type that is the data type of, some EXPRESS attribute in the context schema to which **exp-attribute="attribute-content"** does not apply, and **content="unspecified"** applies,

— the EXPRESS data type is the base-type of, or appears in the working select list of a SELECT data type that is the base type of, an aggregation data type that uses an XML form other than list-of-values (see 8.2.2) and to which **content="ref"** or **content="unspecified"** applies,

— the EXPRESS data type appears in the working select list of a SELECT data type that appears as the data type of an EXPRESS attribute to which **exp-attribute="attribute-content"** applies, or

— the data type is an aggregation data type that uses an XML form other than list-of-values (see 8.2.2) and appears as the data type of an EXPRESS attribute to which **exp-attribute="attribute-content"** applies.

NOTE In short, a data type is referenceable unless exp-type="value" applies to it, or content="value" applies (explicitly or implicitly) to all usages of the data type.

## 8.5 XML Schema definitions and declarations for EXPRESS entity data types

For each EXPRESS entity data type that is declared in or interfaced into the context schema, the corresponding XML schema definitions and declarations shall depend on the EXPRESS entity declaration, on the configuration directives and on the structure of the associated type graph, as specified in 8.5.1.

For each EXPRESS entity data type that is declared in or interfaced into the context schema, if **keep="true"** (see 10.2.11) applies to the entity data type, the entity data type shall be included in the mapping. If **keep="false"** applies to the entity data type, the entity data type shall not be included in the mapping, and there shall be no XML Schema declarations that correspond to that entity data type.

For each entity data type included in the mapping, either the *inheritance mapping* or the *inheritance-free mapping* (both defined below) applies to the entity data type as follows:

— If the configuration directive **inheritance="false"** (see 10.2.15) is specified for the entity, the inheritance-free mapping applies to the entity data type.

— If the configuration directive **inheritance="false"** is specified for the context schema, and no **inheritance** directive is specified for the entity, the inheritance-free mapping applies to the entity data type.

— If **inheritance="true"** is specified, and the entity data type has more than one immediate supertype, the inheritance-free mapping applies to the entity data type.

— Otherwise, the inheritance mapping applies to the entity data type.

NOTE 1 XML schema "inheritance" does not support multiple supertypes or ANDOR relationships among subtypes. If these features are present in a type graph, proper representation of some entity data types in that type graph is not possible using XML schema inheritance – namely, any entity data type that has multiple supertypes. For this reason, those entity data types have the inheritance-free mapping even when **inheritance="true"** is specified.

For each EXPRESS entity data type included in the mapping, the derived XML schema shall contain:

— a `complexType` definition for the corresponding XML data type, as specified in 8.5.4,

— a corresponding instance element declaration, as specified in 8.5.5,

— corresponding XML group declarations, as specified in 8.5.6,

— a corresponding single entity value declaration, if required, as specified in 8.5.7,

— corresponding proxy element declarations, as specified in 8.5.8, and

— corresponding XML uniqueness constraints, as specified in 8.5.10.

Additionally, if the configuration directive **generate-keys="true"** applies, the derived XML schema shall also contain:

— corresponding XML identity constraints, as specified in 8.5.9.

Exception: If, for a given entity data type, all of the following hold:

— the inheritance-free mapping applies to the entity data type,

— the entity data type is declared to be abstract (see 6.3), and

— the entity data type has no immediate subtype that uses the inheritance mapping,

there shall be corresponding XML group declarations, as specified in 8.5.6, corresponding XML uniqueness constraints, as specified in 8.5.10, possible corresponding identity constraint, as specified in 8.5.9.1, and possibly single entity value declarations, as specified in 8.5.7, , but no corresponding XML data type, instance element, or proxy element shall be declared.

NOTE 2 The XML data type and instance element declaration for an abstract entity data type is syntactically necessary for the inheritance mapping.

In addition, for each **entity** configuration directive (see 10.3.3) in which the value of the **select** attribute contains more than one entity name, the derived XML schema shall contain declarations that correspond to an entity data type that is not declared in the EXPRESS schema, called a *synthetic entity data type*. The synthetic entity data type is treated as an entity data type that is a subtype of all the entity data types named by the value of the **select** attribute.

For every synthetic entity data type, the derived XML schema shall contain:

— a `complexType` definition for the corresponding XML data type, as specified in 8.5.4.5,

— a corresponding entity instance element declaration, as specified in 8.5.5,

— corresponding proxy element declarations, as specified in 8.5.8, and

— possibly corresponding XML identity constraints, as specified in 8.5.9.

NOTE 3 By definition, the inheritance-free mapping applies to all synthetic entity data types.

If the configuration directive **tag-source** applies to the entity data type (see 10.2.17), the derived XML schema shall also contain dynamic subtype elements for that entity data type, as specified in 8.5.11, and possibly corresponding XML identity constraints, as specified in 8.5.9.

 

### 8.5.1 Type graph associated with the EXPRESS entity data type

The collection of EXPRESS entity data types that are declared in or interfaced into, the context schema forms a set of "type graphs". Each type graph consists of a set of nodes that represent the entity data types and a set of directed edges that represent the `SUBTYPE OF` relationships. The set of type graphs is formed by the following algorithm:

— (1) If all EXPRESS entity data types have been assigned to some type graph, go to step (6).

— (2) Choose any entity data type not yet assigned to a type graph, and create a new type graph consisting of one node designating that entity data type.

— (3) For the given entity data type, add all its immediate supertypes to the type graph as nodes, eliminating any duplicates. Link the given entity data type to each supertype by a directed edge designated `SUBTYPE OF`.

— (4) For the given entity data type, add all its immediate subtypes to the type graph as nodes, eliminating any duplicates. Link each subtype to the given entity data type by a directed edge designated `SUBTYPE OF`.

— (5) Repeat steps (3) and (4) for each new entity data type added to the type graph (by step (3) or step (4)), until all entity data types in the type graph have been so processed. Then return to step (1).

— (6) For each synthetic entity data type created as required by 8.5, determine the type graph to which its supertypes belong, and add a node to that type graph that represents the synthetic entity data type, with `SUBTYPE OF` edges linked to each of its immediate supertypes.

— (7) For each entity data type to which a **tag-values** configuration directive applies, add a node to the type graph for each XML name appearing in the value of the **tag-values** directive, and link each such node to the entity data type as a "subtype of" that entity data type.

NOTE 1 Step (6) ensures that subtypes representing instances of multiple entity data types related by ANDOR are properly represented in the type graph.

A node in the type graph is said to be a *root node* if it represents an entity data type that has no supertypes. A node in the type graph is said to be a *leaf node* if it represents an entity data type that has no subtypes in the context schema, and no synthetic subtypes. A type graph is said to be a *tree structure*, if it has exactly one root node, and for each leaf node, it has exactly one path along directed edges from the leaf node to the root node.

A type graph is said to be *associated with* each of the EXPRESS entity data types represented in it. Using the construction algorithm defined above, every EXPRESS entity data type in the context schema is associated with exactly one type graph.

NOTE 2 See additional notes in 7.5.1.

### 8.5.2 Complex entity instances

Every EXPRESS entity instance instantiates one or more entity data types, all of which can be associated with nodes in a single type graph, as defined in 8.5.1. A complex entity instance is one that is described by more than one EXPRESS entity declaration. See 7.5.2 for definition and details of complex entity instances.

For some uncharacterized entity instances (defined in 7.5.2) to be represented in the unit of serialization, the derived XML schema may contain declarations that correspond to a synthetic entity data type (see 8.5.4.5) and those instances shall be represented as instances of the synthetic entity data

type, as specified in 9.3. Any other uncharacterized entity instance can be represented using the `exp:complexEntity` instance element (see 8.5.3.5), as specified in 9.3.

### 8.5.3 Base XML data types and elements for EXPRESS entity data types

The derived XML schema shall define, or include by reference, an XML data type representing the EXPRESS `GENERIC_ENTITY` data type, that is the supertype of all entity data types in the context schema. The derived XML schema shall also declare, or include by reference, a corresponding XML element.

The derived XML schema may also declare a *root entity instance element*, representing the supertype of all independently instantiable EXPRESS entity data types in the context schema.

These definitions and declarations are specified in this subclause.

#### 8.5.3.1 Base XML data type for entity data types – `exp:Entity`

See 7.5.3.1 for the definition of `exp:Entity`.

#### 8.5.3.2 Base XML data type for proxy elements – `exp:edokey`

An *external instance element* is an entity instance element that is contained in an external resource. A *proxy element* is an element that appears in the unit of serialization and contains the globally-unique, persistent identifiers for the external instance element and, thus, the information needed to access the external instance.

The derived XML schema includes declarations for proxy elements that correspond to external entity instance elements (see 8.5.8). The XML data types for the proxy elements are extensions of a complexType named `exp:edokey`. The purpose of the `exp:edokey` data type is to provide the basic data structure for proxy elements. The data type of a proxy element is derived from `exp:edokey` and depends on the data type of the external entity instance.

For convenience, the proxy elements are declared to belong to a common substitution group based on an abstract element also called `exp:edokey`. The `exp:edokey` data type and element are defined in the Base XML Schema (see Annex C) and in 7.5.3.2. See 7.5.3.2 for the interpretation of the XML attributes.

The key attributes appear only in the subtypes of `exp:edokey` that are defined for individual entity data types (see 8.5.8).

#### 8.5.3.3 The generic entity data type and instance element

The generic entity data type represents the supertype of all EXPRESS entity data types in the context schema. There is a corresponding XML element whose purpose is to be the root of a substitution group that includes all entity instance elements.

If no **uos Entity** configuration directive is given, the generic entity data type shall be `exp:Entity`, and no additional definition is needed in the derived XML schema. The generic entity instance element shall be `exp:Entity`, which is declared in the Base XML Schema (see Annex C) as:

```
<xs:element name="Entity" type="exp:Entity" abstract="true" />
```

 

If the configuration directive **uosEntity** is given (see 10.3.9), the derived XML schema shall contain a definition for the generic entity data type and a declaration for the corresponding generic entity instance element. These declarations shall have the form:

```
<xs:complexType name="generic-entity">
 <xs:complexContent>
     <xs:extension base="exp:Entity">
         added attributes
     </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="generic-entity" type="Tns:generic-entity"
     abstract="true" />
```

where:

— ***generic-entity*** is the name specified by the **uosEntity** configuration directive.

— ***added attributes*** are the attributes added by the **uosEntity** configuration directive, if any (see 10.3.9).

— ***Tns:generic-entity*** is the fully qualified form of the data type name.

#### 8.5.3.4 The root entity data type and instance element

The root entity data type represents the supertype of all independently instantiable EXPRESS entity data types in the context schema.

NOTE In EXPRESS, independently instantiable entity data types are those interfaced via `USE` instead of `REFERENCE`.

If no **rootEntity** configuration directive is given, the root entity data type shall be the same as the generic entity data type given in 8.5.3.3 above, and the root entity instance element shall be the same as the generic entity instance element.

If the configuration directive **rootEntity** is given (see 10.3.10), the derived XML schema shall contain a definition for the root entity data type and a declaration for the corresponding root entity instance element. These declarations shall have the form:

```
<xs:complexType name="root-entity">
 <xs:complexContent>
     <xs:extension base="generic-entity">
         added attributes
     </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="root-entity" type="Tns:root-entity" abstract="true" />
```

where:

— ***root-entity*** is the name specified by the **rootEntity** configuration directive.

— ***generic-entity*** is the fully qualified name of the generic entity data type, as specified in 8.5.3.3 above.

— ***added attributes*** are the attributes added by the **rootEntity** configuration directive, if any (see 10.3.10).

— ***Tns:root-entity*** is the fully qualified form of the data type name.

#### 8.5.3.5 Instance element for complex entity instances – `exp:complexEntity`

The `exp:complexEntity` instance element is used to represent uncharacterized complex entity instances (see 8.5.2 above) in a unit of serialization. The `exp:complexEntity` element is declared in the Base XML Schema and in 7.5.3.3.

The `exp:complexEntity` element shall contain the single entity value elements for each of the entity data types instantiated in the complex entity instance, as specified in 8.5.7.

The value of the `entities` XML attribute shall be a list of the EXPRESS identifiers for the entity data types corresponding to the single entity values. (See 8.5.7).

#### 8.5.3.6 Base XML data type and element for single entity values – `exp:Single-Entity`

The `exp:Single-Entity` element is the base XML element for the substitution group that represents single entity value elements (see 8.5.7) in a unit of serialization. The `exp:Single-Entity` data type is the base XML data type for single entity value elements. They are declared in the Base XML Schema and in 7.5.3.4.

The XML single entity data type corresponding to a given EXPRESS data type extends this data type to include the EXPRESS attributes declared in the corresponding EXPRESS entity declaration. See 8.5.7.

### 8.5.4 XML data type definitions for entity data types

The `complexType` definition for the EXPRESS entity data type shall be independent of, and appear outside the content of, all XML `complexType` definitions and `element` declarations. That is, the `complexType` shall belong to the schema namespace.

The structure of the `complexType` shall depend on the configuration directives, the structure of the associated type graph (see 8.5.1), and the presence of redeclared attributes, as follows:

If the entity data type is synthetic, the `complexType` definition shall be as specified in 8.5.4.5.

If the inheritance-free mapping applies to the entity data type, the `complexType` definition shall be as specified in 8.5.4.1.

If the inheritance mapping applies to the entity data type but the inheritance-free mapping applies to the immediate supertype, the `complexType` definition shall be as specified in 8.5.4.1.

If the inheritance mapping applies to the entity data type and to its immediate supertype, and the EXPRESS entity declaration contains no redeclared attributes, the `complexType` definition shall be as specified in 8.5.4.2.

If the inheritance mapping applies to the entity data type and to its immediate supertype, and the EXPRESS entity declaration contains *only* redeclared attributes, the `complexType` definition shall be as specified in 8.5.4.3. Exception: If the derived XML schema is to conform in class 1 (see 4.2), the redeclared attributes shall be ignored and the `complexType` definition shall be as specified in 8.5.4.2.

 

If the inheritance mapping applies to the entity data type and to its immediate supertype, and the EXPRESS entity declaration contains both redeclared attributes and additional attributes, the `complexType` definition shall be as specified in 8.5.4.4. Exceptions:

— If the derived XML schema is to conform in class 1 (see 4.2), the redeclared attributes shall be ignored and the `complexType` definition shall be as specified in 8.5.4.2.

— If none of the additional attributes is actually mapped to the derived XML schema (see 8.6.2), the `complexType` definition shall be as specified in 8.5.4.3.

— If none of the redeclared attributes is actually mapped to the XML schema, the `complexType` definition shall be as specified in 8.5.4.2.

NOTE When inheritance mapping is used, but the type graph (see 8.5.1) somewhere contains a synthetic type, or an entity data type that has multiple inheritance, some subgraphs use inheritance mapping, and others do not. The rules above describe three cases: the entity data type is in the part that uses inheritance, the entity data type is in the part that does not use inheritance mapping, or the entity data type is at a connection point.

#### 8.5.4.1 Inheritance-free data type definitions

For all EXPRESS entity data types for which the *inheritance-free mapping* is used, the `complexType` corresponding to the EXPRESS entity data type shall be an extension of the `generic-entity` data type. The extension shall contain declarations for the accessor elements and accessor attributes corresponding to the EXPRESS attributes of the entity data type, as specified below.

NOTE 1 The accessor elements are declared within an `<all>` group to maximize the order independence and extensibility of the EXPRESS attributes within any instance document.

That is, the `complexType` declaration shall have the form:

```
<xs:complexType name="identifier" abstract="abstract">
 <xs:complexContent>
    <xs:extension base="generic-entity">
       <xs:all>
          (accessor element declarations)
       </xs:all>
       (accessor attribute declarations)
    </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

where

— ***identifier*** is derived from the EXPRESS identifier for the entity data type as specified in 8.1.2, or the replacement name from the **name** configuration directive if so specified.

— ***abstract*** shall be `"true"` if the entity data type is declared to be "abstract" in EXPRESS (see 6.3); otherwise it shall be `"false"` and may be omitted.

— ***generic-entity***: If the configuration directive **root="true"** applies to the entity data type (see 10.3.3), ***generic-entity*** shall be the fully qualified name of the root entity data type, as specified in 8.5.3.4; otherwise ***generic-entity*** shall be the fully qualified name of the generic entity data type, as specified in 8.5.3.3.

— ***accessor element declarations*** and ***accessor attribute declarations*** are as specified below.

The `extension` group shall contain one accessor declaration as specified in 8.6 for each EXPRESS attribute that appears in the entity type declaration. The accessor element declarations, if any, shall be contained within an `<all>` particle.

In the type graph associated with the entity data type (see 8.5.1), there will be one or more paths between the node corresponding to the entity data type and one or more root nodes. For every node that appears on *any* such path, the `extension` group shall also contain one accessor declaration as specified in 8.6 for each EXPRESS attribute that appears in the entity type declaration for the entity data type that corresponds to that node. The set of attributes corresponding to a given node shall be mapped only once, regardless of the number of paths that node appears on.

NOTE 2 The above provision implies that every "inherited" attribute, using the EXPRESS notion of inheritance, will have one corresponding accessor declaration in the content model of the `complexType`.

NOTE 3 Per 8.6.2, some attributes of the EXPRESS entity data type may not be mapped to accessors.

If an EXPRESS attribute is redeclared on any path to any root node, the accessor declaration shall correspond to the attribute as redeclared, not to the attribute as originally declared. If an EXPRESS attribute is redeclared more than once on a given path to the entity data type whose declaration contained the original EXPRESS attribute, only the first redeclaration encountered on that path shall be used. If an EXPRESS attribute is not redeclared identically on every path to the entity data type whose declaration contained the original EXPRESS attribute, the declaration to be used shall be implementation defined. In this last case, the preprocessor shall either synthesize a type declaration for the attribute that combines the constraint in the multiple redeclarations, or use the original attribute declaration.

EXAMPLE

In EXPRESS:

```
ENTITY named_unit;
    dimensions : dimensional_exponents;
    unit_type  : unit_enum;
END_ENTITY;

ENTITY si_unit
  SUBTYPE OF(named_unit);
    prefix     : OPTIONAL si_prefix;
    name       : si_unit_name;
  DERIVE
    SELF\named_unit.dimensions : dimensional_exponents :=
       dimensions_for_si_unit(name);
END_ENTITY;
```

In XML Schema (using the directive **exp-attribute= 'attribute-tag'**):

```
<xs:complexType name="Named_unit">
   <xs:complexContent>
      <xs:extension base="exp:Entity">
        <xs:all>
          <xs:element name="Dimensions"
            type="Tns:Dimensional_exponents"/>
          <xs:element name="Unit_type" type="Tns:Unit_enum"/>
        </xs:all>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

 

```
<xs:complexType name="Si_unit">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:all>
        <xs:element name="Dimensions"
          type="Tns:Dimensional_exponents"
          />
        <xs:element name="Unit_type" type="Tns:Unit_enum" />
        <xs:element name="Prefix" type="Tns:Si_prefix"
          minOccurs="0" nillable="true"/>
        <xs:element name="Name" type="Tns:Si_unit_name" />
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

#### 8.5.4.2 Inheritance structures without redeclarations

When the inheritance mapping is used for the EXPRESS entity data type, and the EXPRESS entity declaration does not contain attribute redeclations, the corresponding XML data type shall be a `complexType` as specified in this subclause.

The complexType declaration shall have the form:

```
<xs:complexType name="identifier" abstract="abstract">
 <xs:complexContent>
    <xs:extension base="supertype">
       <xs:sequence>
          (accessor element declarations)
       </xs:sequence>
       (accessor attribute declarations)
    </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

where:

— ***identifier*** is derived from the EXPRESS identifier for the entity data type as specified in 8.1.2, or the replacement name from the **name** configuration directive if so specified.

— ***abstract*** shall be `"true"` if the EXPRESS entity data type is declared to be abstract (see 6.3), otherwise it shall be `"false"`.

— ***supertype*** shall be the qualified name for the XML data type corresponding to the (unique) supertype, as specified in 8.1.3. If the entity data type is the root of the associated type graph (see 8.5.1), and the configuration directive **root="true"** applies to the entity data type (see 10.3.3), ***supertype*** shall be the fully qualified name of the root entity data type, as specified in 8.5.3.4. If **root="false"** (the default) applies, ***supertype*** shall be the fully qualified name of the generic entity data type, as specified in 8.5.3.3.

— ***accessor element declarations*** and ***accessor attribute declarations:*** The `extension` shall contain one accessor element or accessor attribute declaration, as specified in 8.6, for each EXPRESS attribute declared in the EXPRESS entity declaration for the entity data type.

NOTE 1 This leads to an XML type hierarchy that reflects the EXPRESS data type hierarchy (when that hierarchy contains only single inheritance).

EXAMPLE 1

In EXPRESS:

```
ENTITY property_value
  value_name : STRING;
 END_ENTITY;

ENTITY string_value
  SUBTYPE OF ( property_value  );
  value_specification : STRING;
END_ENTITY;
```

In XML Schema:

```
<xs:element name="Property_value" type="Tns:Property_value"
  block="extension restriction" substitutionGroup="exp:Entity"
  nillable="true"/>

<xs:complexType name="Property_value">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:sequence>
        <xs:element name="Value_name" type="xs:normalizedString"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="String_value" type="Tns:String_value"
     block="extension restriction"
     substitutionGroup="Tns:Property_value" nillable="true">
</xs:element>

<xs:complexType name="String_value">
  <xs:complexContent>
    <xs:extension base="Tns:Property_value">
      <xs:sequence>
        <xs:element name="Value_specification"
          type="xs:normalizedString"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

EXAMPLE 2

In EXPRESS:

```
ENTITY A;
A_ATTR : STRING;
END_ENTITY;

ENTITY B SUBTYPE OF (A);
B_ATTR: INTEGER;
END_ENTITY;

ENTITY C SUBTYPE OF (B);
C_ATTR: BOOLEAN;
END_ENTITY;
```

 

In XML Schema:

```
<complexType name="C">
  <complexContent>
    <extension base="Tns:B">
      <sequence>
        <element name="C_attr" type="xs:boolean"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

According to XML schema, the implied structure is actually:

```
<sequence><!-- the C model -->

  <sequence><!-- the B model -->

    <sequence><!-- the A model -->
      <element name="A_attr" type="xs:normalizedString" />
    </sequence>

    <sequence><!-- the B extension particle -->
      <element name="B_attr" type="xs:integer" />
    </sequence>

  </sequence>

  <sequence><!-- the C extension particle -->
    <element name="C_attr" type="xs:boolean" />
  </sequence>

</sequence>
```

NOTE 2 Each level of extension produces an additional level of <sequence> in the internal structure model. According to *XML Schema Part 1: Structures*, the model of an extension (of a non-empty base) is a sequence containing two particles, of which the first is the content particle of the base and the second is the content particle of the extension. In the above, the outermost sequence is the effective model of C – a sequence that contains two sequence particles, the first being the effective model of B and the second being the particle in the C extension. And the effective model of B is a sequence that contains two particles -- the effective model of A and the B extension particle.

#### 8.5.4.3 Inheritance structures with redeclarations only

When the inheritance mapping is used for the EXPRESS entity data type, and the EXPRESS entity declaration contains one or more attribute redeclations, but no additional attributes for the entity datatype will be mapped to the derived XML Schema (see 8.6.2), the corresponding XML data type shall be a `complexType` that is a restriction of the XML data type corresponding to the (unique) supertype of the EXPRESS entity data type. Exception: When the redeclaration of an EXPRESS attribute in the subtype `RENAMES` the attribute in the supertype, the corresponding XML data type shall be as specified in 8.5.4.4.

The `restriction` shall contain declarations for all of the accessor elements and accessor attributes of the XML data type, including redeclarations for all of the accessor elements and accessor attributes that correspond to the EXPRESS attributes that have been redeclared.

The `complexType` definition shall have the form:

```
<xs:complexType name="identifier" abstract="abstract">
 <xs:complexContent>
    <xs:restriction base="supertype">
       <xs:sequence>
          (unchanged and redeclared accessor elements)
       </xs:sequence>
       (redeclared accessor attributes)
    </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

where:

— ***identifier*** is derived from the EXPRESS identifier for the entity data type as specified in 8.1.2, or the replacement name from the **name** configuration directive if specified, and the suffix `-temp` is added.

— ***abstract*** shall be `"true"` if the EXPRESS entity data type is declared to be abstract (see 6.3), otherwise it shall be `"false"`.

— ***supertype*** is the qualified name for the XML data type corresponding to the supertype, as specified in 8.1.3.

— ***(unchanged and redeclared accessor elements)*** is the set of accessor element declarations corresponding to all of the attributes of the supertype (including all attributes the supertype inherits) that are mapped to accessor elements. The accessor elements shall be declared in the same order that they appear in the declaration for the XML data type corresponding to the supertype, and the structure of the `sequence` particles shall match that of the supertype. For attributes that are not redeclared in the entity data type, each of these declarations shall be identical to the corresponding declaration for the accessor element in the XML data type corresponding to the supertype. For each redeclared attribute, the accessor element declaration shall be as specified in 8.6, but using the redeclaration as the EXPRESS attribute declaration.

NOTE 1 As stated in *XML Schema Part 1*, each extension produces an XML data structure that is a sequence particle containing two sequence particles. So when extension is used to represent an EXPRESS subtype (see 8.5.4.2 and 8.5.4.4), the first of these nested particles is the content model of the supertype, and the second is the particle containing the accessor elements proper to the subtype. The content model of the restriction must match this structure, with as many levels of nesting as there are supertypes between the entity data type and the root of the type graph.

— ***(redeclared accessor attributes)*** is the set of accessor attribute declarations corresponding to all of the EXPRESS attributes of the supertype (including all attributes the supertype inherits) that are mapped to accessor attributes and are redeclared in the entity declaration for this entity data type. For each redeclared attribute the accessor attribute declaration shall be as specified in 8.6, but using the redeclaration as the EXPRESS attribute declaration.

NOTE 2 Unchanged XML attributes need not be respecified, as stated in *XML Schema Part 1*.

EXAMPLE

In EXPRESS:

```
ENTITY property_value
  value_name : STRING;
  colour : STRING;
 END_ENTITY;

ENTITY string_value
```

 

```
  SUBTYPE OF ( property_value );
  value_specification : STRING;
  size : STRING;
END_ENTITY;

TYPE identifier=STRING;
END_TYPE;

ENTITY identifier_value
  SUBTYPE OF (string_value);
  SELF\string_value.value_specification : identifier;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Property_value">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:sequence>
        <xs:element name="Value_name" type="xs:normalizedString"/>
        <xs:element name="Colour" type="xs:normalizedString"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="String_value">
  <xs:complexContent>
    <xs:extension base="Tns:Property_value">
      <xs:sequence>
        <xs:element name="Value_specification"
          type="xs:normalizedString"/>
        <xs:element name="Size"
          type="xs:normalizedString"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:simpleType name="Identifier">
  <xs:restriction base="xs:normalizedString"/>
</xs:simpleType>

<xs:complexType name="Identifier_value">
  <xs:complexContent>
    <xs:restriction base="Tns:String_value">
      <xs:sequence>
        <xs:sequence>
          <xs:element name="Value_name" type="xs:normalizedString"/>
          <xs:element name="Colour" type="xs:normalizedString"/>
        </xs:sequence>
        <xs:sequence>
          <xs:element name="Value_specification"
            type="Tns:Identifier"/>
          <xs:element name="Size"
            type="xs:normalizedString"/>
        </xs:sequence>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

#### 8.5.4.4 Inheritance structures with additional and redeclared attributes

When the inheritance mapping is used for the EXPRESS entity data type, and the EXPRESS entity declaration contains additional attributes and one or more attribute redeclations, the corresponding XML data type shall be as specified in this subclause.

NOTE 1 In EXPRESS, redeclarations and new attributes can be defined within the same subtype, but in XML schema, a `complexType` can only be derived from another `complexType` by one of `restriction` or `extension`. Therefore, the equivalent XML schema data type is derived in two steps: an intermediate type derivation by restriction, and then the actual type derivation by extension.

The derived XML schema shall contain an *intermediate* `complexType` definition that is a restriction of the XML data type corresponding to the (unique) supertype of the entity data type. The `restriction` shall contain declarations for all of the accessor elements and accessor attributes of the XML data type corresponding to the supertype, including (re)declarations for all of the accessor elements and accessor attributes that correspond to the EXPRESS attributes that are redeclared in the entity data type. Exception: If the redeclaration of an EXPRESS attribute in the entity data type `RENAMES` the attribute in the supertype, the `restriction` shall contain *no* declaration for the supertype attribute.

The intermediate `complexType` definition shall have the form:

```
<xs:complexType name="identifier-temp" abstract="true">
 <xs:complexContent>
    <xs:restriction base="supertype">
       <xs:sequence>
              (unchanged and redeclared elements)
       </xs:sequence>
       (redeclared attributes)
    </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

where:

— ***identifier*** is derived from the EXPRESS identifier for the entity data type as specified in 8.1.2, or the replacement name from the **name** configuration directive if specified, and the suffix `-temp` is added.

— ***supertype*** is the qualified name for the XML data type corresponding to the supertype, as specified in 8.1.3.

— ***(unchanged and redeclared elements)*** is the set of accessor element declarations corresponding to all of the attributes of the supertype (including all attributes the supertype inherits) that are mapped to accessor elements. The accessor elements shall be declared in the same order that they appear in the declaration for the XML data type corresponding to the supertype, and the structure of the `sequence` particles shall match that of the supertype. For attributes that are not redeclared in the entity data type, each of these declarations shall be identical to the corresponding declaration for the accessor element in the XML data type corresponding to the supertype. For each redeclared attribute the accessor element declaration shall be as specified in 8.6, but using the redeclaration as the EXPRESS attribute declaration. Exception: EXPRESS attributes that are renamed in the redeclaration shall not be included here.

NOTE 2 As stated in *XML Schema Part 1*, each extension produces an XML data structure that is a sequence particle containing two sequence particles. So when extension is used to represent an EXPRESS subtype (see

 

8.5.4.2 and below), the first of these nested particles is the content model of the supertype, and the second is the particle containing the accessor elements proper to the subtype. The content model of the restriction must match this structure, with as many levels of nesting as there are supertypes between the entity data type and the root of the type graph.

— ***(redeclared attributes)*** is the set of accessor attribute declarations corresponding to all of the EXPRESS attributes of the supertype (including all attributes the supertype inherits) that are mapped to accessor attributes and are redeclared in the entity declaration for this entity data type. For each redeclared attribute the accessor attribute declaration shall be as specified in 8.6, but using the redeclaration as the EXPRESS attribute declaration. Exception: EXPRESS attributes that are renamed in the redeclaration shall not be included here.

NOTE 3 Unchanged XML attributes need not be respecified, as stated in ***XML Schema Part 1.***

The `complexType` that corresponds to the EXPRESS entity data type shall be an extension of the intermediate XML data type.

The `complexType` definition shall have the form:

```
<xs:complexType name="identifier" abstract="abstract">
 <xs:complexContent>
    <xs:extension base="identifier-temp">
       <xs:sequence>
          (renamed elements)
          (subtype elements)
       </xs:sequence>
       (renamed attributes)
       (subtype attributes)
    </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

where:

— ***identifier*** is derived from the EXPRESS identifier for the entity data type as specified in 8.1.2, or the replacement name from the **name** configuration directive if specified.

— ***abstract*** shall be `"true"` if the entity data type is declared to be abstract (see 6.3), otherwise it shall be `"false"`.

— ***(renamed elements)*** contains one accessor element declaration for each attribute in the entity declaration that is mapped to an accessor element and `RENAMES` an attribute of the supertype. Each such accessor element declaration shall be as specified in 8.6, using the redeclaration as the EXPRESS attribute declaration.

— ***(subtype elements)*** contains one accessor element declaration for each (new) attribute that is declared in the EXPRESS entity declaration for this entity data type and is specified by 8.6 to be mapped to an accessor element. Each such accessor element declaration shall be as specified in 8.6.

— ***(renamed attributes)*** contains one accessor attribute declaration for each attribute in the entity declaration that is mapped to an accessor attribute and `RENAMES` an attribute of the supertype. Each such accessor attribute declaration shall be as specified in 8.6, using the redeclaration as the EXPRESS attribute declaration.

— ***(subtype attributes)*** contains one accessor attribute declaration for each (new) attribute that is declared in the EXPRESS entity declaration for this entity data type and is specified by 8.6 to

be mapped to an accessor attribute. Each such accessor attribute declaration shall be as specified in 8.6.

NOTE 4 This leads to an XML type hierarchy that reflects the EXPRESS data type hierarchy (when that hierarchy contains only single inheritance).

NOTE 5 The derived XML schema does not retain the RENAMES relationship. In the XML schema, the supertype attribute is implicitly deleted, and the redeclaration is treated as a new attribute.

EXAMPLE

In EXPRESS:

```
ENTITY named_unit;
    dimensions : dimensional_exponents;
    unit_type  : unit_enum;
END_ENTITY;

ENTITY si_unit
  SUBTYPE OF(named_unit);
    prefix     : OPTIONAL si_prefix;
    name       : si_unit_name;
    SELF\named_unit.dimensions RENAMED dimensions_for_si_unit :
         dimensional_exponents;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name="Named_unit">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:sequence>
        <xs:element name="Dimensions" minOccurs="0"
          type="Tns:Dimensional_exponents"/>
        <xs:element name="Unit_type" type="Tns:Unit_enum"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Si_unit-temp" abstract="true">
  <xs:complexContent>
    <xs:restriction base="Tns:Named_unit">
      <xs:sequence>
        <xs:element name="Unit_type" type="Tns:Unit_enum"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Si_unit">
  <xs:complexContent>
    <xs:extension base="Si_unit-temp">
      <xs:sequence>
        <xs:element name="Prefix" type="Tns:Si_prefix"
          minOccurs="0"/>
        <xs:element name="Name" type="Tns:SI_unit_name"/>
        <xs:element name="Dimensions_for_si_unit" minOccurs="0"
          type="Tns:Dimensional_exponents"/>
      </xs:sequence>
```

 

```
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

In an XML document:

```
  <Tns:Si_unit>
    <Unit_type>lengthunit</Unit_type>
    <Prefix>milli</Prefix>
    <Name>metre</Name>
  </Tns:Si_unit>
```

#### 8.5.4.5 Synthetic entity data type definitions

For each **entity** configuration directive (see 10.3.3) in which the value of the **select** attribute contains more than one entity name, the derived XML schema shall contain declarations that correspond to a *synthetic entity data type*. The synthetic entity data type is treated as an entity data type that is a subtype of all the entity data types named in the value of the **select** attribute.

For each synthetic entity data type, the corresponding `complexType` shall be an extension of the `exp:Entity` data type having the form:

```
<xs:complexType name="synthetic-name">
 <xs:complexContent>
    <xs:extension base="generic-entity">
       <xs:sequence>
          accessor element
       </xs:sequence>
       accessor attribute declarations
 </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

where

— ***synthetic-name***: the value of the **name** attribute given in the **entity** configuration directive, if any; otherwise the concatenation of the identifiers for the entity data types appearing in the value of the **select** attribute, each mapped as specified in 8.1.2, and separated from the next by the character HYPHEN-MINUS. The identifiers for the entity data types shall appear in the constructed name in the same order as in the **select** attribute value.

— ***generic-entity***: If the configuration directive **root="true"** applies to the synthetic entity data type (see 10.3.3), ***generic-entity*** shall be the fully qualified name of the root entity data type, as specified in 8.5.3.4; otherwise ***generic-entity*** shall be the fully qualified name of the generic entity data type, as specified in 8.5.3.3.

— ***accessor element declarations*** and ***accessor attribute declarations*** are as specified below.

In the type graph associated with the synthetic entity data type (see 8.5.2), there will be one or more paths between the node corresponding to the synthetic entity data type and one or more root nodes. For every node that appears on *any* such path, the <sequence> group shall contain one element declaration as specified in 8.6 for each EXPRESS attribute that appears in the entity type declaration for the entity data type that corresponds to that node. The set of attributes corresponding to a given node shall be mapped only once, regardless of the number of paths that node appears on.

If an EXPRESS attribute is redeclared more than once on a given path to the entity data type whose declaration contained the original EXPRESS attribute, only the first redeclaration encountered on that path shall be used. If an EXPRESS attribute is not redeclared identically on every path to the entity data type whose declaration contained the original EXPRESS attribute, only the original declaration shall be used.

NOTE By definition, a synthetic entity data type has multiple immediate supertypes. Therefore the associated type graph is not a tree structure, and the pattern for synthetic entity data types is inheritance-free (see 8.5.3.5). There is no EXPRESS declaration for a synthetic entity data type, and therefore no "proper" attributes of that data type – all the attributes are "inherited". The above provisions imply that every EXPRESS attribute of entity instances characterized by the synthetic entity data type will have one corresponding accessor declaration in the content model of the `complexType`.

The corresponding XML schema elements are used to represent all uncharacterized entity instances that have that subgraph.

### 8.5.5 Instance elements corresponding to entity data types

An EXPRESS entity instance may be represented in the exchange document in several forms (see 9.3). A single `element` declaration shall be used to declare the XML element type used for all forms of entity instance element that correspond to the EXPRESS entity data type.

NOTE As stated in 8.5 above, there may be no instance element for an abstract entity data type that uses the inheritance-free mapping.

The instance element declaration shall be independent of, and appear outside the content of, all XML `complexType` definitions and `element` declarations. That is, the instance element type shall belong to the schema namespace.

The `element` declaration for the entity instance element shall have the form:

```
<xs:element name="identifier"
     type="qualified-type"
     nillable="true"
     block="blockattribute"
     substitutionGroup="supertype"/>
```

where:

— ***identifier*** is derived from the EXPRESS identifier for the entity data type as specified in 8.1.2.

— ***qualified-type*** is the qualified name for the XML schema data type that corresponds to the entity data type as specified in 8.5.

— ***blockattribute*** is `'extension restriction'` for entity instance elements and `'#all'` for non-entity instance elements.

— ***supertype*** : If the entity data type uses the inheritance mapping and has a unique supertype, ***supertype*** shall be the fully qualified name of the instance element corresponding to the supertype. Otherwise, if the configuration directive **root="true"** applies to the entity data type (see 10.3.3), ***supertype*** shall be the fully qualified name of the root entity instance element, as specified in 8.5.3.4. If **root="false"** (the default) applies, ***supertype*** shall be the fully qualified name of the generic entity instance element, as specified in 8.5.3.3.

EXAMPLE

In EXPRESS:

```
ENTITY named_unit_2
     SUBTYPE OF (named_unit);
-- all attributes omitted
END_ENTITY;
```

In XML Schema (using inheritance mapping):

```
<xs:element name="Named_unit_2" type="Tns:Named_unit_2"
nillable="true"
  block="extension restriction" substitutionGroup="Tns:Named_unit" />
```

In XML Schema (using inheritance-free mapping):

```
<xs:element name="Named_unit_2" type="Tns:Named_unit_2"
nillable="true"
  block="extension restriction" substitutionGroup="exp:Entity"/>
```

### 8.5.6 XML groups corresponding to entity data types

For every entity data type in the context schema, the XML schema shall contain XML declarations for two groups that correspond to the entity data type: a subtypes `group` as specified in 8.5.6.1, and a complexEntity group as specified in 8.5.6.2.

#### 8.5.6.1 subtypes group

The subtypes group shall represent a `choice` among the XML elements that may be used to represent instances of the entity data type.

The group shall contain one element declaration referring to the corresponding entity instance element (see 8.5.5) for the entity data type itself. Exception: If the entity data type is declared abstract (see 6.3), and is mapped using the inheritance-free mapping, the (possibly non-existent) instance element for the entity data type itself shall not appear in the group.

If external representation of instances of the entity data type is permitted, the group shall contain one element declaration referring to the corresponding proxy element (see 8.5.8) for the entity data type.

If the EXPRESS entity data type is mapped using the inheritance-free mapping, the `<choice>` group shall also contain one group declaration, referring to the corresponding `choice` group, for each immediate subtype of the entity data type that appears in the associated type graph and uses the inheritance-free mapping. Figure 1 - in Clause 7 illustrates the choice group for an inheritance-free mapping.

If the EXPRESS entity data type is mapped using the inheritance mapping, the `<choice>` group shall also contain one group declaration, referring to the corresponding `choice` group, for each subtype of the entity data type, that appears anywhere in the associated type graph, and uses the inheritance-free mapping. Figure 2 - uses the inheritance mapping, but because G has more than one immediate supertype, G uses the inheritance-free mapping. A's group consists of A and G.

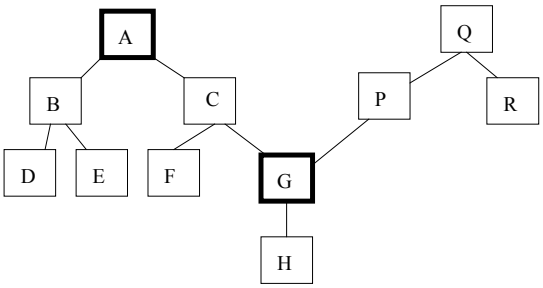


**Figure 2 - Choice group for inheritance mapping**

NOTE 1 The specified nodes represent all subtypes of the entity data type that are declared in, or interfaced into, the context schema (except for any to which **keep="false"** applies) and those created as complex entity data types per 8.5.2.

NOTE 2 If the entity data type is mapped using the inheritance mapping, the substitution group for the corresponding entity instance element contains the instance elements for all of its subtypes. So the subtype instance elements are implicitly in the `choice` group by being valid substitutes for the entity instance element.

NOTE 3 If the inheritance mapping applies generally, but the entity is inheritance-free because it uses multiple inheritance, the substitution group contains all of its immediate subtypes. It however is not in the substitution group for any of its supertypes, and therefore must appear explicitly in the choice groups for each of its supertypes.

That is, the group declaration shall have the form:

```
<xs:group name="identifier-group">
 <xs:choice>
     <xs:element ref="Tns:identifier" />
     <xs:element ref="Tns:identifier-proxy" />
     <xs:group ref="Tns:subtype1-group" />
         ...
     <xs:group ref="Tns:subtypen-group " />
 </xs:choice>
</xs:group>
```

where:

— ***Identifier***`-group`: ***Identifier*** is the local part of the XML name for the instance element corresponding to the entity data type as specified in 8.5.5. '`-group`' is a constant literal suffix.

— ***Tns:identifier*** is the qualified name for the instance element corresponding to the entity data type as specified in 8.5.5. '`-proxy`' is a constant literal suffix.

— ***Tns:subtype***$_1$`-group` ... ***Tns:subtype***$_n$`-group` are the qualified names for the `choice` groups that correspond to subtypes of the entity data type, as specified above.

EXAMPLE (extending the EXAMPLE in 7.5.6.1)

If `si_unit` and `named_unit` use the inheritance mapping, the `Named_unit-group` is declared:

```
<xs:group name="Named_unit-group">
  <xs:choice>
    <xs:element ref="Tns:Named_unit" />
    <xs:element ref="Tns:Named_unit-proxy" />
  </xs:choice>
</xs:group>
```

If external references are not permitted, the proxy element reference is omitted.

#### 8.5.6.2 complexEntity groups corresponding to entity data types

For every entity data type in the context schema, the XML schema shall contain an XML declaration for a complexEntity `group` that corresponds to the entity data type. The complexEntity group shall have a choice group that consists of the subtypes group specified in 8.5.6.1 above. Additionally, if some instances of the entity data type may be uncharacterized, the group shall contain one element declaration referring to the `exp:complexEntity` element.

The complexEntity group declaration shall have the form specified in 7.5.6.2 where:

— ***Identifier***`-complexEntity-group`: ***Identifier*** is the local part of the XML name for the instance element corresponding to the entity data type as specified in 8.5.5. '`-complexEntity -group`' is a constant literal suffix.

— ***Tns:identifier*** is as defined in 8.5.6.1.

— ***complexEntity-usage****:* If any instance of the entity data type may be uncharacterized, the choice group shall contain <xs:element ref=”`exp:complexEntity`”/> Otherwise, the complexEntity element shall be omitted.

See EXAMPLE in 7.5.6.2.

### 8.5.7 Single entity value elements corresponding to entity data types

For each EXPRESS entity data type that is a subtype of another EXPRESS entity data type and may appear in the type graph of an uncharacterized entity instance in the unit of serialization (see 8.5.2), the derived EXPRESS schema shall contain a data type definition and an element declaration for the corresponding single entity value.

Exceptions:

— The XML declarations specified in this subclause shall not appear for entity data types that are *roots* of their associated type graphs (see 8.5.1).

— The XML declarations specified in this subclause shall not appear if all valid instances of the entity data type in any corresponding unit of serialization are characterized either by entity data types in the context schema or by synthetic entity data types (see 8.5).

— The XML declarations specified in this subclause are permitted, but not required, for entity data types that have no uncharacterized `ANDOR` relationships with other entity data types in the type graph. In making this determination, the schema generator may consider the entire set of declarations in the context schema, including entity declarations, supertype clauses, subtype constraints, and other local and global rules.

NOTE This last determination requires a complex algorithm and perhaps additional knowledge. The schema generator is not required to perform such an algorithm or to apply additional knowledge. When in doubt, it should generate the declarations specified in the subclause.

The XML data type definition shall have the form:

```
<xs:complexType name="identifier-value">
 <xs:complexContent>
    <xs:extension base="exp:Single-Entity">
       <xs:all>
          (accessor element declarations)
       </xs:all>
       (accessor attribute declarations)
    </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

and the XML element declaration shall have the form:

```
<xs:element name="identifier-value" type="identifier-value"
    substitutionGroup="exp:Single-Entity" />
```

where

— ***identifier*** is the value of the **name** configuration directive that applies to the entity data type, if any; otherwise the XML name derived from the EXPRESS identifier for the entity data type as specified in 8.1.2.

— ***(accessor element declarations)*** and ***(accessor attribute declarations)***: For each EXPRESS attribute that appears in the entity declaration, the `extension` shall contain one accessor declaration as specified in 8.6. The accessor element declarations, if any, shall be contained within an `<all>` particle. Exception: There shall be no accessor corresponding to an EXPRESS attribute that is redeclared in the entity declaration.

See EXAMPLE in 7.5.7.

### 8.5.8 Proxy elements corresponding to entity data types

The XML data type of the proxy element shall be an extension of the `exp:edokey` data type (see 8.5.3.2), consisting of a <sequence> group that contains the accessor elements corresponding to those EXPRESS attributes of the entity data type that form a unique key for the entity instance within the referenced resource.

The `complexType` and element declarations shall have the form specified in 7.5.8 where:

— ***identifier*** the XML name for the corresponding entity instance element, as specified in 8.5.5 or 8.5.11. The name of the proxy element and data type shall be the concatenation of this XML name and the characters '-proxy'.

— ***Tns:identifier***-proxy is the fully qualified name of the data type of the proxy element.

— ***(key attribute declarations)***: declarations for the accessor elements, as specified in 8.6, that correspond to those EXPRESS attributes of the entity data type that form a unique key for the corresponding entity instances. Within the repository designated by the value of the `exp:authority` attribute, the values of the key attributes for each instance of the entity data type shall identify a unique entity instance.

See EXAMPLE in 7.5.8.

 

### 8.5.9 XML Identity constraints corresponding to entity data types

This subclause applies only to a derived XML schema in conformance class 2.

In order to ensure that a by-reference instance element actually references a by-value instance element of the same type, `xs:key` and `xs:keyref` components corresponding to the entity data type shall be declared, as specified in this subclause.

If the EXPRESS entity data type is referenceable (see 8.4.7), and the configuration directive **generate-keys**="true" applies (see 10.2.23), the derived XML schema shall contain one or more corresponding XML `key` definitions, as specified in 8.5.9.1.

A reference to an entity instance element may be contained in either:

— a by-reference instance element (see 9.3.3) that corresponds to the entity data type, or

— a by-reference accessor element (see 9.4.3.1) or a by-reference accessor attribute (see 9.4.2) that corresponds to an EXPRESS attribute whose data type is the entity data type.

The unit of serialization can contain by-reference instance elements for the entity data type if **content="ref"** or **content="unspecified"** applies to any use of the entity data type as either:

— the base-type of an aggregation type that is mapped using the sequence-of-elements form, or

— the data type of an EXPRESS attribute to which **exp-attribute="double-tag"** applies.

If the entity data type may have by-reference instance elements, the derived XML schema shall contain a corresponding XML `keyref` definition, as specified in 8.5.9.2.

The `key` and `keyref` components shall be defined in the content of the unit of serialization element declaration.

NOTE 1 When an EXPRESS entity instance is the value of an attribute of another entity instance, the XML instance document may contain two corresponding entity instance elements (see 9.3.3) – a by-reference element in the content of the accessor element corresponding to the EXPRESS attribute, and an independent by-value element elsewhere in the content of the unit of serialization. The by-value element contains an `id` XML attribute whose value is a local XML identifier for that entity instance within the unit of serialization. The by-reference element contains a `ref` XML attribute whose value is that local XML identifier. XML schema identity constraints (on the `IDREF` data type of the `ref` attribute) require that there be an element whose `id` value matches the value of the `ref` attribute, which ensures "referential integrity". The purpose of this clause is to provide XML schema declarations that specify that the XML element whose `id` value matches the `ref` value shall be a by-value instance element for the corresponding entity data type. That is, the `xs:key` and `xs:keyref` components provide the type safety between each by-reference instance element and the corresponding by-value instance element.

NOTE 2 If the configuration directive **exp-attribute="attribute-tag"** or **exp-attribute="attribute-content"** is applies to an EXPRESS attribute whose data type is the entity data type, additional `keyref` definitions are required, as specified in 8.6.3 and 8.6.4.

#### 8.5.9.1 Key definitions for entity instance elements

For each EXPRESS entity data type that is referenceable, several identity constraints may be declared in the derived XML schema, as specified in this subclause.

If the EXPRESS entity data type is not declared to be abstract (see 6.3), two `xs:key` schema components shall be defined, one for by-value instance elements corresponding to the entity data type, and one for by-proxy instance elements. The `xs:key` components shall have the form:

```
<xs:key name="schema___entity-key">
 <xs:selector xpath="instance-name" | exp:complexEntity />
 <xs:field xpath="@id" />
</xs:key>

<xs:key name="schema___entity-proxyKey">
 <xs:selector xpath="instance-name-proxy" />
 <xs:field xpath="@id" />
</xs:key>
```

where:

— ***schema*** is the identifier for the EXPRESS schema, mapped as specified in 8.1.2.

— ***entity*** is the identifier for the EXPRESS entity data type, mapped as specified in 8.1.2.

— ***instance-name*** is the qualified name of the instance element corresponding to the entity data type, as specified in 8.5.5.

NOTE 1 This key component requires all corresponding by-value elements that occur in the immediate content of the unit of serialization element to have an `id` attribute with a unique value.

NOTE 2 The `keyref` components defined in 8.5.9.2 require a by-reference instance element to refer to the corresponding by-value element, by referring to the ***schema___entity***`-key`, and a by-proxy reference to refer to the ***schema___entity***`-proxykey`. Since by-value and by-proxy elements that will be referenced only occur in the immediate content of the unit of serialization element (see 9.3), the `xs:selector xpath` expressions refer to instance elements corresponding to the entity data type that are immediate children of the unit of serialization element.

In addition, if any of the potential references to the entity data type are from EXPRESS attributes to which the configuration directive **exp-attribute = "attribute-content"** or **exp-attribute = "attribute-tag"** applies, the derived XML schema shall contain an additional `key` definition for all of the by-value and by-proxy instance elements that may be used to represent a valid value of the EXPRESS entity data type. The `xs:key` component shall have the form:

```
<xs:key name="schema___entity-allKey">
 <xs:selector xpath="instance-name | subtype1 | … | subtypeN |
     instance-name-proxy | subtype1-proxy | … | subtypeN-proxy |
     exp:complexEntity/instance-name | exp:complexEntity/subtype1 |
exp:complexEntity/subtypeN"/>
 <xs:field xpath="@id" />
</xs:key>
```

where:

— ***schema***, ***entity*** and ***instance-name*** are as specified above. If the EXPRESS entity data type is declared to be abstract (see 6.3), ***instance-name*** shall not appear in the list.

— ***subtype1***, ..., ***subtypeN*** are the qualified names of the instance elements corresponding to the instantiable subtypes of the entity data type. The list includes all of the subtypes that appear anywhere in the type graph. If the entity data type is a leaf node of the type graph, there are none.

 

NOTE 3 The `-subtypesKey` and `-proxySubtypesKey` components are only defined when the EXPRESS entity data type has subtypes and is the data type of one or more attribute-tagged EXPRESS attributes (see 8.6.4.2). The `-allKey` component is only defined when the EXPRESS entity data type is the data type of one or more attribute-content EXPRESS attributes (see 8.6.3), but it is defined even when the EXPRESS entity data type has no subtypes.

NOTE 4 These `key` components may be required even when the EXPRESS entity data type itself is declared to be abstract. They require all by-value and by-proxy instance elements that represent a valid value of the entity data type and occur in the immediate content of the unit of serialization element to have an `id` attribute with a unique value.

NOTE 5 The `keyref` components defined in 8.6.3 and 8.6.4, by referring to these `key` components, require accessor elements and accessor attributes to refer to a by-value element or by-proxy element for a valid instance of the named entity data type.. Since by-value and by-proxy elements that will be referenced only occur in the immediate content of the unit of serialization element (see 9.3), the `xs:selector xpath` expression refers to elements that are immediate children of the unit of serialization element.

EXAMPLE

In EXPRESS:

```
SCHEMA person_and_organization;
  ENTITY Organization;
  -- all attributes omitted
  END_ENTITY;
END_SCHEMA;
```

In XML Schema (within the unit of serialization element declaration).

```
<xs:key name="Person_and_organization___Organization-key">
  <xs:selector xpath="Tns:Organization"/>
  <xs:field xpath="@id"/>
</xs:key>
```

### 8.5.9.2 Keyref definitions for entity instance elements

If the EXPRESS entity data type can have by-reference instance elements, two corresponding `xs:keyref` schema component shall be defined.

The `xs:keyref` definitions shall have the form:

```
<xs:keyref name="schema___entity-keyref"
  refer="Tns:schema___entity-key">
 <xs:selector xpath=".//instance-name"/>
 <xs:field xpath="@ref"/>
</xs:keyref>

<xs:keyref name="schema___entity-proxyKeyref"
        refer="Tns:schema___entity-proxyKey" >
 <xs:selector xpath=".//instance-name"/>
 <xs:field xpath="@proxy"/>
</xs:keyref>
```

where:

— ***schema*** is the identifier for the EXPRESS schema, mapped as specified in 8.1.2.

— ***entity*** is the identifier for the EXPRESS entity data type, mapped as specified in 8.1.2.

— ***instance-name*** is the qualified name of the instance element corresponding to the entity data type, as specified in 8.5.5.

— ***Tns:*** is the prefix for the target namespace of the derived XML schema. The value of the `refer` XML attribute shall be the fully qualified name of the corresponding `xs:key` component specified above.

NOTE The `xs:selector xpath` expression for the `xs:keyref` refers to any by-reference element for the entity data type anywhere in the content of the unit of serialization element.

EXAMPLE (continuing the example in 8.5.9.1 above):

In EXPRESS:

```
SCHEMA person_and_organization;
  ENTITY Organization;
  -- all attributes omitted
  END_ENTITY;
END_SCHEMA;
```

In XML Schema (within the unit of serialization element declaration).

```
<xs:keyref name="Person_and_organization___Organization-keyref"
     refer="Tns:Person_and_organization___Organization-key">
  <xs:selector xpath=".//Tns:Organization"/>
  <xs:field xpath="@ref"/>
</xs:keyref>

<xs:keyref name="Person_and_organization___Organization-proxyKeyref"
  refer="Tns:Person_and_organization___Organization-proxyKey" >
  <xs:selector xpath=".//Tns:Organization" />
  <xs:field xpath="@proxy" />
</xs:keyref>
```

### 8.5.10 XML Uniqueness constraints for entity data types

If the entity declaration for the EXPRESS entity data type contains one or more `UNIQUE` rules, corresponding uniqueness constraints shall be declared in the derived XML schema, as specified in this subclause.

For each `UNIQUE` rule contained in the entity declaration, an `xs:unique` schema component shall be defined in the derived XML schema. The `xs:unique` component shall have the form specified in 7.5.9, where:

— ***entity*** is the identifier for the EXPRESS entity data type, mapped as specified in 8.1.2.

— ***instance-name*** is the qualified name of the instance element corresponding to the entity data type, as specified in 8.5.5. Exception: Any abstract entity types shall be omitted.

— ***subtype1***, ..., ***subtypeN*** are the qualified names of the instance elements corresponding to the instantiable subtypes of the entity data type. The list includes all of the subtypes that appear anywhere in the type graph. If the entity data type is a leaf node of the type graph, there are none.

NOTE Listing the subtype instance elements requires the uniqueness constraint to hold over all instances of the entity data type, including those labeled as instances of its subtypes.

 

— ***rulename*** is the EXPRESS identifier for the UNIQUE rule, converted as specified in 8.1.2. If the rule has no identifier, the ***rulename*** for the first UNIQUE rule shall be 1, the second 2, and so on.

— ***attribute1***, ..., ***attributeN*** are the XML names for the accessor attributes or accessor elements, whose data types are not entity data types, corresponding to the EXPRESS attributes specified in the UNIQUE rule. If the XML accessor is an accessor attribute, the XML name shall be preceded by the AT_SIGN (@).

— ***entity-attribute1, ..., entity-attributeN*** are the XML names for the accessor attributes or accessor elements, whose data types are entity data types, corresponding to the EXPRESS attributes specified in the UNIQUE rule. If the XML accessor is an accessor attribute, the XML name shall be preceded by the AT_SIGN (@).

See EXAMPLE in 7.5.9.

### 8.5.11 Dynamic subtype elements corresponding to entity data types

If a **tag-source** directive applies to the entity data type, the data type is considered to have *dynamic subtypes* that are not explicitly modeled in the context schema, but can be determined by the value of the EXPRESS attribute designated by the value of **tag-source**. The value of the corresponding **tag-values** directive shall be a list of the names of the dynamic subtypes.

For each XML name in the tag-values list, the derived XML schema shall contain a corresponding entity instance element declaration of the form:

```
<xs:element name="XMLname"  type="qualified-name"  nillable="true"
 block="blockattribute" substitutionGroup="qualified-name"/>
```

where:

— ***XMLname*** is the name from the **tag-values** list.

— ***qualified-name*** is the qualified name for the XML data type (and the instance element) that corresponds to the EXPRESS entity data type as specified in 8.5.3.5.

— ***blockattribute***: if the EXPRESS entity data type uses inheritance mapping, the value of block shall be "extension restriction". If the entity data type uses inheritance-free mapping, the value of block shall be "#all".

— the ***substitutionGroup*** attribute is omitted if the inheritance-free representation is used for the entity data type.

EXAMPLE

In EXPRESS:

```
TYPE label = STRING;
END_TYPE;

ENTITY action_relationship;
  name: label;
  description: STRING;
  relating_action: action;
  related_action: action;
END_ENTITY;

ENTITY action;
```

```
  -- all attributes omitted
END_ENTITY;
```

With configuration directive:

```
<entity select="Action_relationship" tag_source="Name" tag_values=
"Action_precedes_action Action_requires_action Action_replaces_action"
/>
```

In XML Schema (using inheritance mapping):

```
<xs:element name="Action_precedes_action"
  type="Tns:Action_relationship" nillable="true"
  block="extension restriction"
  substitutionGroup="Tns:Action_relationship" />

<xs:element name="Action_requires_action"
  type="Tns:Action_relationship" nillable="true"
  block="extension restriction"
  substitutionGroup="Tns:Action_relationship" />

<xs:element name="Action_replaces_action"
  type="Tns:Action_relationship" nillable="true"
  block="extension restriction"
  substitutionGroup="Tns:Action_relationship" />
```

In XML Schema (using inheritance-free mapping):

```
<xs:element name="Action_precedes_action"
  type="Tns:Action_relationship" nillable="true" block="#all"/>

<xs:element name="Action_requires_action"
  type="Tns:Action_relationship" nillable="true" block="#all"/>

<xs:element name="Action_replaces_action"
  type="Tns:Action_relationship" nillable="true" block="#all"/>
```

## 8.6 XML Schema declarations for EXPRESS attributes

This subclause specifies the mapping of EXPRESS attributes to XML schema.

8.6.2 specifies which EXPRESS attributes of the owning entity shall be mapped to the XML schema.

For each EXPRESS attribute that is specified in 8.6.2 to be mapped to the XML schema, the mapping shall depend on the value of configuration directive **exp-attribute** that applies to the EXPRESS attribute (see 10.5.1).

If **exp-attribute="attribute-content"** applies to the EXPRESS attribute, the EXPRESS attribute shall be mapped to an XML attribute, referred to as the *accessor attribute* corresponding to that EXPRESS attribute, as specified in 8.6.3.

If **exp-attribute="attribute-tag"** or **exp-attribute="double-tag"** applies to the EXPRESS attribute, the EXPRESS attribute shall be mapped to an XML element, referred to as the *accessor element* corresponding to that EXPRESS attribute, as specified in 8.6.4.

If **exp-attribute="type-tag"** applies to the EXPRESS attribute, the EXPRESS attribute shall be mapped as specified in 8.6.5.

If **exp-attribute="no-tag"** applies to the EXPRESS attribute, the EXPRESS attribute shall be mapped as specified in 8.6.6.

### 8.6.1 Accessor element and attribute naming

If a **name** configuration directive (see 10.3.4) applies to the EXPRESS attribute, the name of the accessor element or accessor attribute corresponding to the EXPRESS attribute shall be as specified by that directive.

If no **name** configuration directive applies to the EXPRESS attribute, the name of the accessor element or accessor attribute corresponding to the EXPRESS attribute shall be derived from the EXPRESS identifier for the attribute as specified in 8.1.2.

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

— the XML name derived as specified in 8.1.2 from the identifier for the EXPRESS entity data type whose entity-declaration declares the attribute, followed by

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

— the XML name derived from the EXPRESS attribute identifier as specified in 8.1.2.

NOTE 1 The above exception applies to both characterized and uncharacterized EXPRESS complex entity data types (see 8.5.4.5).

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. As specified in 8.5.3.5, the XML Schema data type corresponding to the entity data type contains only one accessor element. The above exception does not apply.

### 8.6.2 EXPRESS attributes mapped to XML schema

This subclause specifies the kinds of EXPRESS attributes that are mapped to XML schema and the circumstances under which they are mapped.

#### 8.6.2.1 Explicit attributes

Each explicit EXPRESS attribute appearing in the entity declaration shall be mapped as follows:

— If the configuration directive **keep="false"** applies to the EXPRESS attribute (see 10.5.9 and 10.3.5), the attribute is not mapped to the derived XML schema.

— If the configuration directive **keep="true"** applies to the EXPRESS attribute, there shall be a corresponding accessor element declaration, as specified in 8.6.4, or accessor attribute declaration, as specified in 8.6.3.

— If the EXPRESS attribute is declared to be a **tag-source** (see 10.2.17) and the configuration directive **keep="true"** applies, it shall be mapped as if it were declared `OPTIONAL`.

For each explicit EXPRESS attribute that is created by an **inverse** configuration directive, as specified in 10.3.5, there shall be a corresponding accessor declaration. The new EXPRESS attribute shall be mapped as if it had been declared as an explicit attribute in the context schema.

#### 8.6.2.2 INVERSE attributes

For each EXPRESS `INVERSE` attribute that is converted to an explicit attribute by an **inverse** configuration directive, as specified in 10.3.5, there shall be a corresponding accessor declaration. The `INVERSE` attribute shall be mapped as if it had been declared as an explicit attribute in the context schema.

Each EXPRESS `INVERSE` attribute that is *not* converted to an explicit attribute, and each explicit EXPRESS attribute that is converted to an `INVERSE` attribute by an **inverse** configuration directive shall be mapped as follows:

— If the configuration directive **keep="false"** applies to the EXPRESS attribute (see 10.5.9), the attribute is not mapped to the derived XML schema.

— If the configuration directive **keep="true"** applies to the EXPRESS attribute, there shall be a corresponding accessor declaration.

#### 8.6.2.3 DERIVEd attributes

EXPRESS `DERIVE`d attributes are mapped to the derived XML schema only when required by the configuration directive **keep** (see 10.2.11).

If the configuration directive **keep="false"** applies to the EXPRESS attribute (see 10.5.9), the attribute is not mapped to the derived XML schema.

If the configuration directive **keep="true"** applies to the EXPRESS attribute, there shall be a corresponding accessor declaration. The `DERIVE`d attribute shall be mapped to the XML schema as if it were declared `OPTIONAL`.

#### 8.6.2.4 Redeclared attributes

If an EXPRESS attribute is redeclared in a subtype, the data type of the redeclared attribute affects the declaration of the corresponding accessor elements, as specified in 8.5.4. Except as specified in (the appropriate subclause of) 8.5.4, a redeclared attribute is not mapped to the derived XML schema.

If an explicit attribute is redeclared as `DERIVE`d or `INVERSE` in a subtype, the original attribute in the supertype shall be mapped as if it were declared `OPTIONAL`.

#### 8.6.2.5 Generic attributes

An EXPRESS abstract entity data type (see 6.3) may have attributes whose EXPRESS data types are "generalized types" that are not instantiable. Such attributes are said to be *generic attributes*. They are required by ISO 10303-11 to be redeclared in every instantiable subtype of the abstract entity data type.

When the abstract entity data type is mapped to XML schema as specified in 8.5, no generic attribute shall be mapped to XML schema as an attribute of the abstract entity data type. Only the redeclarations of the generic attribute in the instantiable subtypes shall be mapped.

NOTE In the XML schema, the redeclared attributes will appear first in the extensions that correspond to the subtypes, and they will appear to be new attributes of the subtypes.

EXAMPLE

In EXPRESS:

 

```
ENTITY approval_relationship ABSTRACT SUPERTYPE;
  approval_data : approval;
  approval_item : GENERIC_ENTITY;
END_ENTITY;
```

In XML schema (with inheritance mapping and **exp-attribute="double-tag"**):

```
<xs:complexType name="Approval_relationship">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:sequence>
        <xs:element name="Approval_data">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="Tns:Approval" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <!-- there is no declaration for Approval_item -->
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

### 8.6.3 Accessor attributes

For each EXPRESS attribute that is mapped to an accessor attribute (see 8.6 above), the `complexType` corresponding to the entity data type shall contain one corresponding attribute declaration for the accessor attribute.

The name of the accessor attribute shall be as specified in 8.6.1. If the EXPRESS attribute is mandatory, the `use` attribute of the accessor attribute shall be specified in the attribute declaration to have the value `"required"`.

If the EXPRESS attribute is declared to be `OPTIONAL`, or specified in 8.6.2 to be treated as `OPTIONAL`, the `use` attribute of the accessor attribute shall be specified in the attribute declaration to have the value `"optional"`.

NOTE 1 An unset EXPRESS attribute is indicated in the instance data by the absence of the accessor attribute. (See 9.3.1)

NOTE 2 When an EXPRESS `DERIVE`d attribute is mapped to XML Schema (see 8.6.2.3), it is treated as if it were declared to be `OPTIONAL`.

NOTE 3 If the EXPRESS attribute is redeclared as `DERIVE`d or `INVERSE` in any subtype of this entity data type, the EXPRESS attribute is treated as being `OPTIONAL`, as specified in 8.6.2.2 and 8.6.2.4.

The data type of the accessor attribute shall be declared as follows:

— If a **map** configuration directive (see 10.2.13) applies to the EXPRESS attribute, the data type of the accessor attribute shall be declared to be the data type specified by the **map** directive;

— If the data type of the EXPRESS attribute is an entity data type, the XML data type of the accessor attribute shall be declared to be `IDREF`;

— If the data type of the EXPRESS attribute is an aggregation data type, or a defined data type whose fundamental type is an aggregation data type, and the corresponding XML data has the

sequence-of-elements form or any form of aggregate of aggregates (see 8.2.2), the XML data type of the accessor attribute shall be declared to be `IDREF`;

— If the data type of the EXPRESS attribute is a `SELECT` data type, the XML data type of the accessor attribute shall be declared to be `IDREF`;

— If the configuration directive **content="ref"** applies to the attribute (see 10.5.2), the XML data type of the accessor attribute shall be declared to be `IDREF`;

— Otherwise, the data type of the accessor attribute shall be declared to be the XML schema data type that corresponds to the data type of the EXPRESS attribute, as specified in 8.2.

If the XML data type of the accessor attribute is `IDREF`, the derived XML schema shall also contain a `keyref` declaration for the attribute, as specified in 8.6.3.1.

EXAMPLE 1 Example of attributes whose data types are simple:

In EXPRESS:

```
TYPE Identifier = STRING;
END_TYPE;

TYPE Weight_measure = REAL;
END_TYPE;

ENTITY Pipe;
    Id : Identifier;
    Description : STRING;
    Nominal_size : NUMBER;
    Diameter : REAL;
    Bends : INTEGER;
    Available : BOOLEAN;
    Valid : LOGICAL;
    Image : BINARY;
    Weight : Weight_measure;
END_ENTITY;
```

In XML Schema:

```
<xs:simpleType name = "Identifier">
  <xs:restriction base = "xs:normalizedString"/>
</xs:simpleType>

<xs:simpleType name = "Weight_measure">
  <xs:restriction base = "xs:double"/>
</xs:simpleType>

<xs:complexType name = "Tns:Pipe">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:attribute name="Id" type="Tns:Identifier" use="required"/>
      <xs:attribute name="Description" type="xs:normalizedString"
         use="required"/>
      <xs:attribute name="Nominal_size" type="xs:decimal"
         use="required"/>
      <xs:attribute name="Diameter"  type="xs:double"
         use="required"/>
      <xs:attribute name="Bends" type="xs:long" use="required"/>
```

 

```
        <xs:attribute name="Available" type="xs:boolean"
           use="required"/>
        <xs:attribute name="Valid" type="exp:logical" use="required"/>
        <xs:attribute name="Image" type="xs:hexBinary" use="required"/>
        <xs:attribute name="Weight" type="Tns:Weight_measure"
           use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

In XML instance data:

```
<Tns:Pipe id="i10"
   Id="P1-1"
   Nominal_size="3.0"
   Diameter="2.75"
   Bends="2"
   Available="true"
   Valid="unknown"
   Weight="34.8"
   Description="Standard rigid double-elbow"
   Image="8AF009F3ADCBEA04"
/>
```

Note that the uppercase `Id` attribute (derived from the EXPRESS attribute `id`) and the lowercase `id` attribute (needed for XML) are different XML attributes. Note also that the ordering of the XML attributes is irrelevant.

EXAMPLE 2 Example of an attribute whose data type is an aggregate of a simple data type, using the list-of-values form:

In EXPRESS:

```
ENTITY Pt3d;
   c : ARRAY [1:3] OF REAL;
END_ENTITY;
```

In XML schema:

```
<xs:complexType name = "Pt3d">
   <xs:complexContent>
      <xs:extension base = "exp:Entity">
        <xs:attribute name="C">
          <xs:simpleType>
            <xs:restriction>
              <xs:simpleType>
                <xs:list itemType="xs:double"/>
              </xs:simpleType>
              <xs:minLength value="3"/>
              <xs:maxLength value="3"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

<xs:element name="Pt3d" type="Tns:Pt3d" nillable="true"/>
```

EXAMPLE 3 Example of an attribute whose data type is an aggregate of an entity data type:

In EXPRESS, extending Example 2:

```
ENTITY Polyline;
  points : LIST OF Pt3d;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name = "Polyline">
  <xs:complexContent>
    <xs:extension base = "exp:Entity">
      <xs:attribute name="Points" type="xs:IDREF"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="Polyline" type="Tns:Polyline" nillable="true"/>
```

In XML instance data:

```
<Tns:Polyline id="e001" Points="a123" />

<Tns:Seq-Pt3d id="a123" >
  <Tns:Pt3d id="e004" C="1.0 0.0 0.0" />
  <Tns:Pt3d id="e005" C="0.0 1.0 0.0" />
  <Tns:Pt3d ref="e006" xsi:nil="true" />
</Tns:Polyline>

<Tns:Pt3d id="e006" C="0.0 0.0 1.0" />
```

#### 8.6.3.1 Keyref definitions for accessor attributes

If the XML data type of the accessor attribute is `IDREF`, the derived XML schema shall contain a XML `keyref` definition corresponding to the EXPRESS attribute. The `keyref` definition shall have the form:

```
<xs:keyref name="schema___entity___attribute-keyref" refer="keyname" >
 <xs:selector xpath=".//Tns:entity"/>
 <xs:field xpath="@attribute"/>
</xs:keyref>
```

where:

— ***schema*** is derived from the identifier for the EXPRESS schema as specified in 8.1.2.

— ***entity*** is the local part of the XML data type name corresponding to the owning EXPRESS entity, as specified in 8.5.3.

— ***attribute*** is the local part of the XML name corresponding to the EXPRESS attribute, as specified in 8.6.1.

— ***Tns:entity*** is fully qualified XML name of the instance element corresponding to the owning EXPRESS entity, as specified in 8.5.5.

— ***keyname*** is the fully qualified name of the `key` component corresponding to the data type of the EXPRESS attribute. For entity data types, ***keyname*** shall refer to the ***Tns:associate***-`allKey` component, where ***associate*** corresponds to the data type of the attribute (the associated entity data

 

type, not the owning entity data type). For SELECT data types, ***keyname*** shall refer to the key component specified in 8.3.4.4. For other data types, ***keyname*** shall refer to the key component specified in 8.4.6.

NOTE The xs:selector xpath expression for the keyref refers to any instance element for the owning entity data type appearing anywhere in the content of the unit of serialization element. The xs:field value refers to the accessor attribute. The keyref definition requires the value of that XML attribute to refer to an element of the type identified in the key component designated by ***keyname***.

EXAMPLE extending Example 3 above:

In EXPRESS:

```
SCHEMA geometry;

ENTITY pt3d;
  c : ARRAY [1:3] OF REAL;
END_ENTITY;

ENTITY polyline;
  points : LIST OF Pt3d;
END_ENTITY;

END_SCHEMA;
```

In XML schema, the declaration for polyline is:

```
<xs:complexType name = "Polyline">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:attribute name="Points" type="xs:IDREF" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Since the accessor attribute Points is declared to have data type IDREF, the following keyref declaration is required:

```
<xs:keyref name="Tns___Polyline___Points-keyref"
     refer="Tns:Geometry___Seq-Pt3d-key" >
  <xs:selector xpath=".//Tns:Polyline" />
  <xs:field xpath="@Points" />
</xs:keyref>
```

### 8.6.4 Accessor elements

For each EXPRESS attribute that is mapped to an accessor element (see 8.6 above), the <sequence> or <all> particle in the complexType corresponding to the entity data type shall contain one corresponding declaration for the attribute element.

The accessor element shall be declared as follows:

— The name of the accessor element shall as specified in 8.6.1.

— The maxOccurs attribute of the accessor element need not be specified in the element declaration. If it is specified, it shall have the value "1".

— If the EXPRESS attribute is mandatory, the `minOccurs` attribute of the accessor element need not be specified in the element declaration. If it is specified, it shall have the value `"1"`.

— If the EXPRESS attribute is declared to be `OPTIONAL`, or specified in 8.6.2 to be treated as `OPTIONAL`, the XML `element` declaration shall contain the XML attribute `minOccurs="0"`, and the XML attribute `nillable="true"`.

NOTE 1 An unset EXPRESS attribute is indicated in the instance data either by the absence of the attribute element or by an empty accessor element with XML attribute `xsi:nil="true"`. (See 9.3.1)

NOTE 2 When an EXPRESS `DERIVE`d attribute is mapped to XML Schema, it is treated as if it were declared to be `OPTIONAL`. (See 8.6.2.3)

NOTE 3 If the EXPRESS attribute is redeclared as `DERIVE`d or `INVERSE` in any subtype of this entity data type, the EXPRESS attribute is treated as being `OPTIONAL`, as specified in 8.6.2.3 and 8.6.2.2 .

— The accessor element shall be declared to have the XML attribute `exp:attributeType`, defined in the Base XML Schema, with a `fixed` value. If the EXPRESS attribute is explicit (or treated as explicit), the value shall be `"explicit"`; if the EXPRESS attribute is `INVERSE` (or treated as `INVERSE`), the value shall be `"inverse"`; if the EXPRESS attribute is `DERIVE`d , the value shall be `"derived"`. Exception: If the EXPRESS attribute is explicit, the `exp:attributeType` XML attribute may be omitted.

— If **exp-attribute="attribute-tag"** applies to the EXPRESS attribute and the data type of the EXPRESS attribute is not an entity data type, the XML data type of the accessor element shall be as specified in 8.6.4.1.

— If **exp-attribute="attribute-tag"** applies to the EXPRESS attribute and the data type of the EXPRESS attribute is an entity data type, the XML data type of the accessor element shall be as specified in 8.6.4.2.

— If **exp-attribute="double-tag"** applies to the attribute, the XML data type of the accessor element shall be as specified in 8.6.4.3.

In addition, if the configuration directive **exp-attribute="attribute-tag"** applies to the EXPRESS attribute and **content="ref"** or **content="unspecified"** also applies to the EXPRESS attribute, the derived XML schema may be required to contain a corresponding `keyref` definition , as specified in 8.6.4.4.

NOTE 4 When the configuration directive **exp-attribute="double-tag"** applies to the EXPRESS attribute, the XML `keyref` definition corresponding to the attribute is different, and is required by 8.5.9 or 8.4.6.3.

### 8.6.4.1 Attribute-tagged attribute whose data type is not an entity data type

If **exp-attribute="attribute-tag"** applies to the EXPRESS attribute, and the data type of the EXPRESS attribute is not an entity data type, the data type of the accessor element shall be declared to be the XML schema data type that corresponds to the data type of the EXPRESS attribute, as specified in 8.2.

NOTE 1 This is the default mapping of an EXPRESS attribute whose data type is not an entity.

Exceptions:

— When a **map** directive applies to the EXPRESS attribute (see 10.5.4), the data type of the accessor element shall be declared to be the XML data type specified by the **map** directive.

 

— When a **notation** directive applies to the EXPRESS attribute but *not* to the data type of the attribute (that is, when the directive was given for the attribute rather than the data type), the data type of the accessor element shall be declared as follows:

```
<xs:complexType>
 <xs:simpleContent>
  <xs:extension base = "attribute-type">
   <xs:attribute ref="xml:notation" use="fixed"
     value="configured-notation"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

where:

— ***attribute-type*** is the XML Schema data type that corresponds to the data type of the EXPRESS attribute, as specified in 8.2.

— ***configured-notation*** is the value given for the **notation** directive.

NOTE 2 When the corresponding XML data type has a name, the data type of the accessor element can be declared as: `type="name"`. Otherwise (in the case of anonymous aggregates) the data type must be declared in full, as specified in 8.2.2.

EXAMPLES 1 through 4 see EXAMPLES in 7.6.3.1

EXAMPLE 5 Example of a mapping of an `INVERSE` attribute (cf. Example 2 in 7.6.3.1):

In EXPRESS:

```
ENTITY Pt3d;
    c : ARRAY [1:3] OF REAL;
INVERSE
    in_curve: SET [0:1] OF Polyline FOR points;
END_ENTITY;
```

If the configuration directive:

```
<cnf:entity select="Pt3d">
  <cnf:attribute select="In_curve" keep="true" />
</cnf:entity>
```

is given, the derived XML schema declarations are:

```
<xs:complexType name = "Pt3d">
  <xs:complexContent>
    <xs:extension base = "exp:Entity">
      <xs:all>
        <xs:element name = "C">
          <xs:simpleType>
            <xs:restriction>
              <xs:simpleType>
                <xs:list itemType="xs:double"/>
              </xs:simpleType>
              <xs:minLength value="3"/>
              <xs:maxLength value="3"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="In_curve">
```

```
            <xs:complexType>
             <xs:sequence>
              <xs:element ref="Tns:Polyline"
                minOccurs="0" maxOccurs="1"/>
             </xs:sequence>
              <xs:attribute ref="exp:attributeType" fixed="inverse"/>
            </xs:complexType>
          </xs:element>
        </xs:all>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="Pt3d" type="Tns:Pt3d"/>
```

XML instance data:

```
  <Tns:Pt3d id = "i4">
    <C>1.0 0.0 0.0</C>
  </Tns:Pt3d>
```

#### 8.6.4.2 Attribute-tagged attributes whose data type is an entity data type

This subclause specifies the mapping of EXPRESS attributes to which **exp-attribute="attribute-tag"** applies and the data type of the EXPRESS attribute is an entity data type. The mapping depends on whether the inheritance mapping or the inheritance-free mapping (see 8.5) applies to the data type of the EXPRESS attribute, called the *associated entity data type*.

If the mapping of the associated entity data type is an inheritance mapping, the XML data type of the accessor element shall be the XML data type corresponding to the associated entity data type, as specified in 8.5.4.

If the mapping of the associated entity data type is the inheritance-free mapping, and the associated entity data type has no subtypes in the type graph, the XML data type of the accessor element shall be the XML data type corresponding to the associated entity data type, as specified in 8.5.4.

If the mapping of the associated entity data type is the inheritance-free mapping, and the associated entity data type has subtypes in the type graph, the XML data type of the accessor element shall be specified as `exp:Entity`.

NOTE 1 When the value of the attribute is an instance of a subtype, the accessor element will have an `xsi:type` XML attribute. Because `exp:Entity` is declared to be abstract, those accessor elements will always have an `xsi:type` XML attribute.

NOTE 2 The XML schema declaration allows the entity instance to be represented either by-value or by-reference in the content of the accessor element. The **exp-type** and **content** configuration directives determine which representation option is to be used in which circumstances.

EXAMPLE

In EXPRESS (this extends the Example in 8.5.4.4):

```
ENTITY measure_with_unit;
  value_component : measure_value;
  unit_component : named_unit;
END_ENTITY;

TYPE measure_value = REAL;
```

 

```
END_TYPE;

ENTITY named_unit;
   dimensions : dimensional_exponents;
END_ENTITY;

ENTITY si_unit
  SUBTYPE OF(named_unit);
    prefix     : OPTIONAL si_prefix;
    name       : si_unit_name;
  DERIVE
    SELF\named_unit.dimensions : dimensional_exponents :=
       dimensions_for_si_unit(name);
END_ENTITY;
```

In XML Schema:

For definitions of XML types and elements corresponding to NAMED_UNIT and SI_UNIT, see 8.5.4.4

```
<xs:simpleType name="Measure_value" >
     <xs:restriction base="xs:double" />
</xs:simpleType>
```

If **exp-attribute="attribute-tag"** for UNIT_COMPONENT and inheritance for NAMED_UNIT:

```
<xs:complexType name = "Measure_with_unit">
   <xs:complexContent>
      <xs:extension base = "exp:Entity">
        <xs:sequence>
          <xs:element name = "Value_component"
            type="Tns:Measure_value"/>
          <xs:element name = "Unit_component"
            type="Tns:Named_unit"/>
        </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

If **exp-attribute="attribute-tag"** for UNIT_COMPONENT and inheritance-free for NAMED_UNIT:

```
<xs:complexType name = "Measure_with_unit">
   <xs:complexContent>
      <xs:extension base = "exp:Entity">
        <xs:all>
          <xs:element name = "Value_component"
            type="Tns:Measure_value"/>
          <xs:element name = "Unit_component" type="exp:Entity"/>
        </xs:all>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

### 8.6.4.3 Double-tagged attributes

This subclause specifies the mapping of EXPRESS attributes to which **exp-attribute="double-tag"** applies.

NOTE 1 This is the default mapping of an EXPRESS attribute whose data type is an entity data type.

The XML data type of the accessor element shall be an anonymous complexType, of the form:

```
<xs:complexType>
 <xs:sequence>
     <element-or-group ref="instance-name"/>
 </xs:sequence>
</xs:complexType>
```

where:

— ***instance-name***: If the data type of the EXPRESS attribute is an entity data type, ***instance-name*** shall be the qualified name of the corresponding complexEntity group (see 8.5.6.2). If the underlying data type of the EXPRESS attribute is a SELECT data type, ***instance-name*** shall be the qualified name of the corresponding group or instance element (see 8.3.4).

— ***element-or-group*** shall be `xs:element`, if ***instance-name*** is an element, or `xs:group`, if ***instance-name*** is a group.

NOTE 2 The XML schema declaration allows the element to be either by-value or by-reference. The configuration directives determine which representation option is to be used in which circumstances.

EXAMPLE

In EXPRESS (this extends the Example in 8.5.4.4, compare with other subclauses of 8.6.2):

```
ENTITY measure_with_unit;
   value_component : measure_value;
   unit_component : named_unit;
END_ENTITY;

TYPE measure_value = REAL;
END_TYPE;

ENTITY named_unit;
   dimensions : dimensional_exponents;
END_ENTITY;

ENTITY si_unit
  SUBTYPE OF(named_unit);
    prefix     : OPTIONAL si_prefix;
    name       : si_unit_name;
  DERIVE
    SELF\named_unit.dimensions : dimensional_exponents :=
       dimensions_for_si_unit(name);
END_ENTITY;
```

In XML Schema:

If **exp-attribute="double-tag"** for `UNIT_COMPONENT` and inheritance for `NAMED_UNIT`:

```
<xs:complexType name = "Measure_with_unit">
   <xs:complexContent>
      <xs:extension base = "exp:Entity">
         <xs:sequence>
            <xs:element name = "Value_component"
              type="Tns:Measure_value" />
            <xs:element name = "Unit_component" >
              <xs:complexType>
                 <xs:sequence>
                    <xs:element ref = "Tns:Named_unit"/>
                 </xs:sequence>
              </xs:complexType>
```

 

```
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

If **exp-attribute="double-tag"** for `UNIT_COMPONENT` and inheritance-free mapping for `NAMED_UNIT`:

```
<xs:complexType name = "Measure_with_unit">
  <xs:complexContent>
    <xs:extension base = "exp:Entity">
      <xs:all>
        <xs:element name = "Value_component"
          type="Tns:Measure_value" />
        <xs:element name = "Unit_component" >
          <xs:complexType>
            <xs:sequence>
              <xs:group ref = "Tns:Named_unit-complexEntity-group"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

#### 8.6.4.4 Keyref definitions for accessor elements

If the configuration option **generate-keys="true"** applies to the derived XML schema, and the configuration directive **exp-attribute="attribute-tag"** applies to the EXPRESS attribute and **content="ref"** or **content="unspecified"** also applies to the EXPRESS attribute, the derived XML shall contain an XML `keyref` definition corresponding to the attribute. The `keyref` definition shall not appear inside any other definition or declaration. The `keyref` definition shall have the form:

```
<xs:keyref name="schema___entity__attribute-keyref" refer="keyname" >
 <xs:selector xpath=".//Tns:entity/attribute"/>
 <xs:field xpath="@ref" />
</xs:keyref>
```

where:

— ***schema*** is derived from the identifier for the EXPRESS schema as specified in 8.1.2.

— ***entity*** is the local part of the XML data type name corresponding to the owning EXPRESS entity, as specified in 8.5.3.

— ***attribute*** is the XML name corresponding to the EXPRESS attribute, as specified in 8.6.1.

— ***Tns:entity*** is the fully qualified XML name of the instance element corresponding to the owning EXPRESS entity, as specified in 8.5.5.

— ***keyname*** is the fully qualified name of the `key` component corresponding to the data type of the EXPRESS attribute as follows:

- For entity data types, if the associated entity data type has subtypes, ***keyname*** shall refer to the ***Tns:associate***-`allKey` component specified in 8.5.9, otherwise, to the ***Tns:associate***-`key` component specified in 8.5.9. In either case, ***associate*** corresponds

to the data type of the EXPRESS attribute (the associated entity data type, not the owning entity data type).

– For SELECT data types, ***keyname*** shall refer to the key component specified in 8.3.4.4.

– For all other data types, ***keyname*** shall refer to the key component specified in 8.4.6.

NOTE The xs:selector xpath expression for the keyref refers to the accessor element where it appears in any instance element for the owning entity data type anywhere in the content of the unit of serialization element. The xs:field value refers to the ref XML attribute, which appears when the accessor element contains a reference instead of a value. The keyref definition requires the value of the ref attribute to refer to an element of the type identified in the key component designated by ***keyname***.

In addition, if the above directives apply and the data type of the EXPRESS attribute is an entity data type, the derived XML shall contain an XML keyref definition corresponding to by-proxy representation of the attribute value. The keyref definition shall have the form:

```
<xs:keyref name="schema___entity___attribute-proxyKeyref"
  refer="keyname" >
 <xs:selector xpath=".//Tns:entity/attribute"/>
 <xs:field xpath="@proxy" />
</xs:keyref>
```

where:

— ***schema***, ***entity*** and ***attribute*** are as specified above.

— ***keyname*** is the fully qualified name of the corresponding key component: If the associated entity data type has subtypes, ***keyname*** shall refer to the ***Tns:associate***-proxySubtypesKey component,, otherwise to the ***associate***-proxyKey component, where in either case, ***associate*** corresponds to the data type of the attribute value (the associated entity data type, not the owning entity data type).

### 8.6.5 Type-tagged attributes

If an **exp-attribute="type-tag"** configuration directive (see 10.2.5) applies to the EXPRESS attribute, the data type of the EXPRESS attribute shall be a named data type or an anonymous aggregate whose base-type is a named data type. Exceptions: When the data type of the EXPRESS attribute has subtypes, “type-tag” shall have no effect. When the data type of the EXPRESS attribute is an anonymous aggregate, and the inheritance-free mapping applies, “type-tag” shall have no effect. In these cases the default value of exp-attribute shall apply instead.

NOTE 1 It is an error to specify exp-attribute= “type-tag” for an attribute when its data type has subtypes in the type graph, or if there is more than one attribute in the entity with the same data type, or if the attribute is an anonymous aggregate and the inheritance-free mapping applies (see 10.2.5).

For each EXPRESS attribute to which **exp-attribute="type-tag"** applies, the <sequence> or <all> group in the complexType corresponding to the entity data type shall contain one corresponding element or group declaration, referring to the instance element or group corresponding to the named data type, as follows:

— If the named data type is an entity data type, the <sequence> or <all> group shall contain the instance element corresponding to the entity data type (see 8.5.5).

 

— If the named data type is a defined data type whose fundamental type is not a SELECT type, the <sequence> or <all> group shall contain the instance element corresponding to the defined data type (see 8.4.3).

— If the named data type is a defined data type whose fundamental type is a SELECT type, the <sequence> or <all> group shall contain the resolved set of elements corresponding to the defined data type XML group determined as follows:

- The <sequence> or <all> group shall be initialized to contain all of the xs:elements and xs:groups contained in the XML group corresponding to the defined data type (see 8.3.4.2).
- Each xs:group in the <sequence> or <all> group shall be deleted from the <sequence> or <all> group, and replaced by all of the xs:elements and xs:groups contained within.
- The previous step shall be repeated until there are no xs:groups in the <sequence> or <all> group.
- Any duplicate xs:elements in the group shall be removed.

If the data type of the EXPRESS attribute is the named data type (not an anonymous aggregate):

— The maxOccurs attribute of the accessor element need not be specified in the element declaration; if it is specified, it shall have the value "1".

— If the EXPRESS attribute is mandatory, the minOccurs attribute of the accessor element need not be specified in the element declaration; if it is specified, it shall have the value "1".

— If the EXPRESS attribute is declared to be OPTIONAL, or specified in 8.6.2 to be treated as OPTIONAL, the element declaration shall contain the XML attributes minOccurs="0" and nillable="true".

If the data type of the EXPRESS attribute is an anonymous aggregate, and the inheritance mapping applies, the minOccurs and maxOccurs attributes of the accessor element shall be specified in the element declaration, and their values shall be as follows:

— The value of the minOccurs attribute shall be "0", if the base-type of the aggregate is OPTIONAL; otherwise, it shall be the lower bound on the size of the EXPRESS aggregate, represented as a decimal integer.

NOTE It is an error to use this directive if the data type of the EXPRESS attribute is an anonymous aggregate and the inheritance-free mapping applies, as the <all> group does not permit elements within to have values of maxOccurs and minOccurs other than 0 and 1.

— The value of the maxOccurs attribute shall be the upper bound on the EXPRESS aggregate, if given, represented as a decimal integer; if the upper bound of the EXPRESS aggregate is not given or indeterminate ('?'), the value of maxOccurs shall be "unbounded".

NOTE 2 If the EXPRESS aggregate is an ARRAY, the value of both the lower bound and the upper bound is: HIINDEX(A) – LOINDEX(A) + 1, where A is the aggregation data type.

— If the EXPRESS attribute is declared to be OPTIONAL, or specified in 8.6.2 to be treated as OPTIONAL, the element declaration shall contain the XML attribute nillable="true".

NOTE 3 An unset EXPRESS attribute is indicated in the instance data either by the absence of the attribute element or by an empty accessor element with XML attribute xsi:nil="true". (See 9.3.1)

NOTE 4 When an EXPRESS DERIVEd attribute is mapped to XML Schema, it is treated as if it were declared to be OPTIONAL. See 8.6.2.3.

NOTE 5 If the EXPRESS attribute is redeclared as DERIVEd or INVERSE in any subtype of this entity data type, the EXPRESS attribute is treated as being OPTIONAL. . See 8.6.2.4.

EXAMPLE 1 In EXPRESS (compare with examples in 8.6.4):

```
ENTITY measure_with_unit;
  value_component : measure_value;
  unit_component : named_unit_2;
END_ENTITY;

TYPE measure_value = REAL;
END_TYPE;

ENTITY named_unit_2;
  dimensions : dimensional_exponents;
END_ENTITY;
```

In XML Schema:

If **exp-attribute="type-tag"** for UNIT_COMPONENT and inheritance for NAMED_UNIT_2:

```
<xs:complexType name = "Measure_with_unit">
  <xs:complexContent>
    <xs:extension base = "exp:Entity">
      <xs:sequence>
        <xs:element name = "Value_component"
          type="Tns:Measure_value" />
        <xs:element ref = "Tns:Named_unit_2"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

If **exp-attribute="type-tag"** for UNIT_COMPONENT and inheritance-free mapping for NAMED_UNIT_2:

```
<xs:complexType name = "Measure_with_unit">
  <xs:complexContent>
    <xs:extension base = "exp:Entity">
      <xs:all>
        <xs:element name = "Value_component"
          type="Tns:Measure_value"/>
        <xs:element ref = "Tns:Named_unit_2"/>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

EXAMPLE 2 Example of an attribute whose data type is an aggregate of an entity data type (compare with 8.6.4.1 Example):

In EXPRESS:

```
ENTITY Pt3d
    c : ARRAY[1:3] OF REAL;
END_ENTITY;

ENTITY Polyline
```

```
    points : LIST OF Pt3d;
END_ENTITY;
```

In XML Schema:

```
<xs:complexType name = "Polyline">
  <xs:complexContent>
    <xs:extension base = "exp:Entity">
      <xs:sequence>
        <xs:element ref = "Tns:Pt3d"
          minOccurs = "0" maxOccurs = "unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

### 8.6.6 No-tag attributes

This subclause specifies the mapping for an EXPRESS attribute to which **exp-attribute ="no-tag"** applies.

NOTE 1 As specified in 10.5.1.5, the data type of the EXPRESS attribute is an entity data type that has no subtypes in the context schema. That entity data type is referred to as the *associated entity data type*.

The EXPRESS attribute itself shall have no corresponding accessor element or attribute in the XML schema. In its place, each attribute of the associated entity data type shall be mapped, as if it were an attribute of the owning entity data type, as specified in 8.6.

NOTE 2 That is, all attributes of the associated entity data type effectively become attributes of the owning entity data type. This allows the XML schema to implement a "view" that "joins" the two entity data types.

EXAMPLE

In EXPRESS:

```
ENTITY resource;
 ...
END_ENTITY;

ENTITY process;
  ...
END_ENTITY;

TYPE resource_assignment_item = SELECT(process,...);
END_TYPE;

ENTITY resource_role;
  name : label;
END_ENTITY;

ENTITY resource_assignment;
 assigned_resource: resource;
 role: resource_role;
 items: SET [1:?] OF resource_assignment_item;
END_TYPE;
```

In XML Schema, when **exp-attribute="no-tag"** is specified for ROLE, the declaration for RESOURCE_ASSIGNMENT would be:

```
<xs:complexType name = "Resource_assignment">
  <xs:complexContent>
    <xs:extension base = "exp:Entity">
      <xs:sequence>
        <xs:element name="Assigned_resource" type="Tns:Resource" />

<!-- the attribute 'role' is replaced by the attributes of entity
'resource_role' -->
        <xs:element name="Name"  type="Tns:Label" />

        <xs:element name="Items" nillable="true">
          <xs:complexType>
                <xs:group ref="Resource_assignment_item"
                  minOccurs="1" maxOccurs="unbounded"/>
                <xs:attribute name="ref" type="xs:IDREF"
                  use="optional"/>
                <xs:attribute ref="exp:arraySize"
                  use="optional"/>
                <xs:attribute ref="exp:itemType"
                  fixed="Resource_assignment_item"/>
                <xs:attribute ref="exp:cType" fixed="set"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

## 8.7 XML Schema and namespaces for EXPRESS Schema

The EXPRESS SCHEMA statement for the context schema shall map to the XML Schema component xs:schema. That is, there shall be one derived XML schema corresponding to the context EXPRESS schema and a given configuration file.

An XML namespace (the "target namespace") shall be associated with the derived XML schema. The namespace URI associated with the target namespace shall be as specified in 8.7.2. The xs:schema element shall have a targetNamespace attribute, whose value shall be the namespace URI associated with the target namespace.

The name and version of the EXPRESS schema, and the URI for the configuration file, if any, should be provided in documentation elements for the XML schema.

NOTE The name and version of the EXPRESS schema are not formally mapped to any elements of the derived XML schema. They may influence the formation of the target_namespace URI. See 8.7.2.

Additional namespace declarations for the derived XML Schema shall be as specified in 8.7.3.

The derived XML schema shall comprise exactly the definitions and declarations specified by 8.2 through 8.6 and 8.8.

If the configuration directive **embed-schema-items="true"** applies (see 10.2.24), the derived XML schema shall also contain exactly those XML data type definitions, attribute declarations and element declarations from Annex C that are used (directly or indirectly) in the definitions and declarations required by 8.2 through 8.6 and 8.8 for the context EXPRESS schema. The copied definitions and

 

declarations shall match their counterparts in Annex C verbatim. In this case, the prefix "`exp:`" that appears in those subclauses shall refer to the target namespace for the derived XML schema, and may be replaced by the prefix used for the target namespace (see 8.7.3).

If the configuration directive **embed-schema-items="false"** applies, the derived XML schema shall contain none of the XML data type definitions, attribute declarations and element declarations from Annex C. Instead, the Base XML Schema specified in Annex C shall be explicitly included in the derived XML schema by means of an `xs:import` declaration. In this case, the preprocessor shall choose a namespace prefix (see 8.7.3) to correspond to the prefix "`exp:`" that appears in subclauses 8.2 through 8.6 and 8.8, and the chosen prefix shall correspond to the namespace identified by the URN:

`urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common`.

### 8.7.1 Namespace prefixes

For each namespace to which an **alias** configuration option (see 10.2.25) applies, the namespace shall be included by a namespace declaration, and the prefix shall be as specified in the **prefix** directive. For any required namespace declarataion not specified by an **alias** configuration directive, the namespace prefix shall be as specified in 7.7.1.

### 8.7.2 URI for the target namespace of the derived XML schema

The namespace URI associated with the target namespace for the derived XML schema shall be the URI specified by the **namespace** configuration directive (see 10.2.18), if any, or else by the `schemaLocation` attribute of the `configuration` element that contains the configuration file.

If neither is provided, the URI for the target namespace shall be as specified in 7.7.2.

### 8.7.3 Namespace declarations for the derived XML schema

The `xs:schema` element shall contain (xmlns) namespace declarations for:

— the targetNamespace . The value of the namespace URI used in this namespace declaration shall be the same as that assigned to the `targetNamespace` attribute.

— the current version of the XML Schema definition.

— if the configuration directive **embed-schema-items="false"** applies (see above), the Base XML Schema (see Annex C).

In each case, the prefix value and the namespace URI may be specified by a corresponding **namespace** configuration directive (see 10.2.18). For each **namespace** configuration directive, there shall be a corresponding `xmlns` declaration in the `xs:schema` element. For any `xmlns` declaration required above for which there is not a **namespace** configuration directive, the preprocessor may choose the prefix, as specified in 8.7.1, and the appropriate URI.

If the `xs:schema` element contains the `elementFormDefault` or `attributeFormDefault` attribute, their respective values shall be set to `'unqualified'`, which is the default value specified in *XML Schema Part 1: Structures*.

See EXAMPLE in 7.7.3.

### 8.7.4 Import declarations for the derived XML schema

Import declarations shall be as specified in 7.7.4.

## 8.8 Context-schema specific unit of serialization

For the context schema there shall be a specific unit of serialization element and a corresponding `complexType` in the namespace corresponding to the derived XML schema

The unit of serialization element declaration shall have the form:

```
<xs:element type="Tns:uos" name="uos" substitutionGroup="exp:uos">
uniqueness_constraints
</xs:element>
```

where:

— ***Tns*** is the namespace prefix that corresponds to the namespace associated with the derived XML schema for the unit of serialization, as specified in 8.7.

— ***uos*** is the name specified by the **uosElement** configuration directive, if any (see 10.3.8), otherwise `uos`.

— ***uniqueness_constraints:*** `any xs:unique` schema components required by 8.5.10.

The context specific `complexType` shall be an extension of the `exp:uos complexType`, described in 5.6 (which has no content model).

The content model of the context-specific `complexType` shall be a repeating `choice` group consisting of:

— the generic entity instance element (see 8.5.3.3),

— the `exp:edokey` instance element, and

— every referenceable instance element that corresponds to a non-entity data type (see 8.4.7).

In addition, the `complexType` shall have any additional attributes specified by the **uosElement** configuration directive, if any.

That is, the `complexType` for the unit of serialization shall have the form:

```
<xs:complexType name="uos">
  <xs:complexContent>
        <xs:extension base="exp:uos">
            <xs:choice maxOccurs="unbounded" minOccurs="0">
                <xs:element ref="generic-entity" />
                <xs:element ref="exp:edokey" />
                <xs:element ref="referenceable-instance-name-1"/>
                        ...
                <xs:element ref="referenceable-instance-name-n"/>
            </xs:choice>
            added attributes
        </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

where:

— ***uos*** is the name specified by the **uosElement** configuration directive, if any (see 10.3.8), otherwise `uos`.

 

— ***generic-entity*** is the fully qualified name for the generic entity data type (see 8.5.3.3).

— ***referenceable-instance-name-i*** are the fully qualified names for the referenceable non-entity instance elements specified above.

— ***added attributes*** are the attributes added by the **uosElement** configuration directive, if any (see 10.3.8).

See EXAMPLE in 7.8.

# 9 XML document creation

An XML document based on this part of ISO 10303 may contain zero or more EXPRESS schemas, zero or more configuration files, and zero or more units of serialization, each representing a set of data described by an EXPRESS schema.

## 9.1 Preconditions

The creation of an XML instance document assumes the following preconditions:

— all EXPRESS schemas provided or referenced conform to the specifications of ISO 10303-11;

— every configuration file provided or referenced conforms to the specifications of clause 10, and describes the mapping of an EXPRESS schema that is provided or referenced;

— the context schema for each unit of serialization is an EXPRESS schema that is provided or referenced;

— the data in each unit of serialization consists of a set of entity instances that satisfy all local rules for the corresponding entity data types stated in the context schema.

## 9.2 General XML document structure

An XML document conforming to this part of ISO 10303 shall begin with an `xml` declaration with `version="1.0"`.

NOTE 1 Other requirements in this clause specify circumstances where additional XML attributes may be necessary.

An XML document conforming to this part of ISO 10303 shall be either a uos document, as specified in 9.2.2, or an iso-10303-28 document, as specified in 9.2.1.

NOTE 2 A uos document can contain only a single data set conforming to the default XML schema (see Clause 7) for the EXPRESS schema that governs it. An iso-10303-28 document can contain multiple data sets, including data sets conforming to configured XML schemas (see Clause 8), together with EXPRESS schemas and configuration files.

NOTE 3 In most cases, an XML document conforming to some configured XML schema does not conform to this Part of ISO 10303. It may conform to some other Part or standard. But the result of some configurations is a conforming uos document (see 4.1.2).

### 9.2.1 Structure of an iso-10303-28 document

An iso-10303-28 document shall include the `iso_10303_28` element. The content of the `iso_10303_28` element shall be valid with respect to the Document Schema (see Annex D).

NOTE 1 The `iso_10303_28` element need not be the root element of the document, and there is no required relationship between the document type (`doctype`) and the Document Schema.

If the XML document contains document header information, that information shall be represented using the `header` element, as specified in 5.2.1.

NOTE 2 This Part of ISO 10303 places no requirements on the use of the `header` element and its content, other than that they constitute valid XML elements under the XML schema defined in Annex C.

The `iso_10303_28` element shall include zero or more `express` elements, representing EXPRESS schemas either by reference or by value, as specified in 9.2.3.

The `iso_10303_28` element shall include zero or more `configuration` elements, as specified in 9.2.4, each representing the configuration rules for mapping an EXPRESS schema to XML schema. For every EXPRESS schema that is referenced in a configuration file or used as the context schema for a data set contained in the `iso_10303_28` element, there shall be one `express` element (see 5.4) in the content of the `iso_10303_28` element.

The `iso_10303_28` element shall include zero or more unit of serialization elements, as specified in 9.2.6 each representing a data set described by a given EXPRESS schema.

The `iso_10303_28` element shall include zero or more `schema_population` elements, as specified in 9.2.5, each defining a complete schema instance described by a given EXPRESS schema as a set of data sets represented by unit of serialization elements.

EXAMPLE An XML document could have the following structure:

```
<?xml version="1.0" encoding="utf-8"?>
<iso_10303_28 version="2.0">
  <header> ... </header>
  <express id="sch101" schemaLocation=
   "URI:www.tc184-sc4.org/schemas/Part214/cc6-lf-schema.exp"
  />
  <configuration id="cf101" schema="sch101"
   configuration-location=
   "URI:www.pdtnet.org/Abbildung/AP214cc6-cf.xml"   />
  <uos id="d101-24" configuration="cf101"
    xmlns="URN:www.pdtnet.org/Schema/AP214cc6" >
    (entities)
..</uos>
</iso_10303_28>
```

### 9.2.2 Structure of a uos document

A uos document shall consist of a single unit of serialization element as specified in 9.2.6, with the following additional requirements:

— The derived XML schema that describes it shall be the default XML schema corresponding to the EXPRESS context schema.

 

— The unit of serialization element shall, in all cases, specify the XML schema that describes the encoding used in that unit of serialization element. It may also specify the context schema and configuration for the unit of serialization element.

### 9.2.3 Encoding of EXPRESS schemas

For every EXPRESS schema that is referenced in a configuration file or used as the context schema for a data set contained in the `iso_10303_28` element, there shall be one `express` element (see 5.4) in the content of the `iso_10303_28` element. The element may provide the EXPRESS schema by value or by reference.

If the schema is provided by value, the content of the `express` element shall be a single character string value containing a representation of the EXPRESS schema that conforms to ISO 10303-11, using the character set encoding specified for the XML document.

NOTE 1 In many cases, the most convenient representation of the EXPRESS schema will be a `CDATA` section containing the "formatted" text of the EXPRESS schema.

If the EXPRESS schema is provided by reference, the content of the `express` element shall be empty, the element shall have the attribute `xsi:nil="true"`, and the `schemaLocation` attribute shall be present. The value of the `schemaLocation` attribute shall locate a resource that contains the EXPRESS schema by value.

In either case, the `express` element shall have an `id` attribute, and its value shall be a local XML identifier for the corresponding EXPRESS schema for all references within the `iso_10303_28` element.

The value of the `schema_name` attribute shall be the EXPRESS schema name, represented as specified in 8.1.1. The value of the `schema_version` attribute, if present, shall identify the EXPRESS schema version. The value of the `schema_identifier` attribute, if present, shall be the (ASN.1) schema identifier for the EXPRESS schema. At least one of `schema_name` or `schema_identifier` shall be present and have the corresponding value.

NOTE 2 Only some EXPRESS Schemas will have ASN.1 identifiers. This provides for those that do.

### 9.2.4 Encoding of configuration files

If the `iso_10303_28` element contains an EXPRESS schema that is

— used as the context schema for a data set contained in the `iso_10303_28` element, and

— mapped to XML schema using some configuration other than the default,

the `iso_10303_28` element shall contain a corresponding `configuration` element (see 10.1).

Each `configuration` element provides the definition of one configuration of one or more EXPRESS schemas, either by value or by reference.

The `configuration` element shall have an `id` attribute, and its value shall be a local XML identifier for that configuration file. The local XML identifier shall be used for all references to that configuration file within the `iso_10303_28` element.

The `configuration` element may have a `schema` attribute, and its value shall be the local XML identifier for the `express` element that provides the EXPRESS schema whose mapping this configuration file defines. If the configuration describes the mapping of a specific EXPRESS schema,

and the configuration directives contain schema-specific references, the schema attribute shall be present. If the configuration file describes the mapping of an arbitrary EXPRESS schema, and the configuration directives contain no schema-specific references, the schema attribute may be omitted.

If the configuration file is provided by value, the content of the configuration element shall be a collection of configuration directives conforming to Clause 10.2 that defines the configuration, using the character set encoding specified for the XML document.

If the configuration file is provided by reference, the content of the configuration element shall be empty, the element shall have the attribute xsi:nil="true", and the configuration-location attribute shall be present. The value of the configuration-location attribute shall locate a resource that contains the configuration file by value.

### 9.2.5 Encoding of population definitions

While the preconditions above (9.1) include the assumption that every entity instance that is represented in the unit of serialization is a valid entity instance (satisfies the local rules for that entity data type that are specified in the context schema), there is no requirement that the content of the unit of serialization as a whole be a valid *population*, as described by ISO 10303-11, and satisfy all the global RULEs in the context schema. If it is intended that the content of a given unit of serialization element, or a collection of unit of serialization elements taken together, represents a valid population, that fact is represented in the iso_10303_28 element by a schema_population element, as specified in 5.3.

For each schema_population element that appears in the content of the iso_10303_28 element, the corresponding collection of unit of serialization elements shall constitute a valid population of the context schema, as defined in ISO 10303-11. The interpretation of the schema _population element in specifying a population shall be as specified in 5.3.

### 9.2.6 Encoding of data sets – the unit of serialization

This subclause specifies the rules governing the creation of the content of the unit of serialization element that corresponds to a given schema instance.

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

— represent a collection of EXPRESS entity instances that are described by a single EXPRESS schema, designated the *context schema* for the unit of serialization,

— have been constructed from those EXPRESS entity instances according to the specifications in this clause, and

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

For each unit of serialization to be conveyed in the iso_10303_28 element, one unit of serialization element shall appear in the content of the iso_10303_28 element. The unit of serialization element shall be of the XML element type that corresponds to the context schema, as specified in 8.8.

The unit of serialization element shall specify either the XML schema that describes the encoding used in that unit of serialization element (see 9.2.6.2), or the context schema and configuration for the unit of serialization element (see 9.2.6.1). It may specify both.

 

#### 9.2.6.1 Specifying the context schema and configuration for the unit of serialization

One or both of the XML attributes `express` and `configuration` may appear in the unit of serialization element. If present, they specify the context schema and configuration used to define the XML schema for the `uos`. If neither is specified, the `schemaLocation` attribute shall be present.

If the `configuration` attribute is present in the unit of serialization element, there shall be one `configuration` element in the `iso_10303_28` element whose `id` attribute has a value identical to the value of the `configuration` attribute. That `configuration` element is referred to below as the *specified* `configuration` element.

If the `express` attribute is present in the unit of serialization element, there shall be one `express` element in the `iso_10303_28` element whose `id` attribute has a value identical to the value of the `express` attribute. That `express` element is referred to below as the *specified* `express` element.

If the `configuration` attribute is present in the unit of serialization element and the `express` attribute is not present, the `schema` attribute shall be present in the specified `configuration` element (see 9.2.4), and its value shall identify the `express` element that provides the context schema for the unit of serialization. The specified `configuration` element shall provide the configuration file for the corresponding XML schema that governs the formation of the elements in the unit of serialization.

If the `express` attribute is present in the unit of serialization element and the `configuration` attribute is not present, the specified `express` element shall provide the context schema for the unit of serialization (see 9.2.2). The default XML schema that corresponds to the context schema shall govern the formation of the elements in the unit of serialization.

If both the `configuration` and `express` XML attributes appear in the unit of serialization element, the specified `express` element shall provide the context schema for the unit of serialization (see 9.2.2). The specified `configuration` element shall provide the configuration information for the corresponding XML schema (see 9.2.4) that governs the formation of the elements in the unit of serialization. Any `express` attribute of the specified `configuration` element itself shall be ignored.

NOTE That is,

— If the unit of serialization has only a `configuration` attribute, the `configuration` element to which it refers identifies the context schema and describes the XML schema configuration.

— If the unit of serialization has only a `express` attribute, it identifies the context schema, and the default XML schema configuration is used.

— If the unit of serialization has both, the `express` attribute identifies the context schema, and the `configuration` attribute identifies the configuration to be used.

#### 9.2.6.2 Specifying the XML schema for the unit of serialization

If present, the `schemaLocation` attribute shall specify a list of resources that contain the derived XML schema that describes the encoding used in the unit of serialization. By convention, the first item in the list shall be a URI or URN that is the global unique identifier for the XML schema, if any, and the subsequent items shall be URLs at which the text of the derived XML schema can be found.

#### 9.2.6.3 Specifying the XML namespaces for the unit of serialization

The XML attribute `xmlns` shall appear in the unit of serialization element, and its value shall designate the namespace associated with the XML schema corresponding to the context schema, as determined by the rules above.

NOTE The use of a prefix for the XML names of types and instance elements declared in the derived schema is not required. However, W3C strongly encourages the use of prefixes.

The unit of serialization element shall also contain the following `xmlns` attribute, if the unit of serialization contains instance elements declared in the Base XML Schema:

```
xmlns:exp=
"urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"
```

If the representation of the unit of serialization uses XML names from any other namespaces, the unit of serialization element shall contain an additional `xmlns` attribute for each such namespace.

NOTE The configuration file may contain directives requiring the use of additional namespaces in the derived XML schema, but those namespaces are only required in the unit of serialization element if it contains element names or attribute names taken from those namespaces.

#### 9.2.6.4 Specifying the data content of the unit of serialization

The content of the unit of serialization element shall consist of a collection of XML elements representing the set of EXPRESS entity instances in the unit of serialization. Every EXPRESS entity instance in the unit of serialization shall be represented by one or more XML elements in the content of the unit of serialization element, as specified in 9.3.

NOTE Some entity instances are always represented by elements that are immediate children of the unit of serialization element. Depending on the configuration directives, all entity instances may be required to be so represented, or some entity instances may be represented within the content of the elements representing other entity instances. See 9.4.

EXAMPLE This example illustrates the unit of serialization:

In XML instance data:

```
<?xml version="1.0"?>
<!DOCTYPE iso_10303_28 SYSTEM "car_ownership.xsd">
<iso_10303_28
   xmlns="urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common">
   <express id="exp01" name="Car_ownership">
     <[CDATA[
SCHEMA car_ownership;
ENTITY person;
  name : STRING;
  owns : car;
END_ENTITY;
ENTITY car;
  make : STRING;
  car_model : STRING;
END_ENTITY;
END_SCHEMA;
]]>
   </express>
```

 

```
    <uos id="expdata01" schema="exp01"
         xmlns:own="URN:my_company.com:iso_10303_28/Car_ownership"
         xmlns:exp=
         "urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"
         express="Car_ownership" >
       <own:Person  id="id10" >
         <Name>John Q. Public</Name>
         <Owns ref="id20"/>
       </own:Person>
       <own:Car  id="id20">
         <Make>BMW</Make>
         <Car_model>Z3</Car_model>
       </own:Car>
    </uos>
  </iso_10303_28>
```

#### 9.2.6.5 Labeling the unit of serialization an enterprise data object

An *enterprise data object* is a data resource that may be referred to in business transactions and other exchange documents. An enterprise data object has a fixed global identifier by which it is known. For consistency with XML practice, we assume that the fixed global identifier is a "reference URI" that is independent of location.

If the set of EXPRESS entity instances contained in the unit of serialization represents an enterprise data object, the `edo` attribute should be present in the unit of serialization element and its value should be the fixed global identifier. Otherwise, the `edo` attribute should not be present.

NOTE 1 The context schema for the data resource should be identical to the context schema for the unit of serialization, but the configuration is not significant – more than one configuration could be used to convey the same data resource.

NOTE 2 The primary use of the `edo` identifier is to facilitate future exchanges with the recipient. But it could also be provided in order to allow the recipient to determine easily whether this data set is the same as, or different from, another data set that the recipient has otherwise obtained.

NOTE 3 The local XML identifiers for entity instances in an enterprise data object may have more than local significance. That is, specific entity instances in the data resource may have known identifiers and be externally referenced by reference URI + local id.

NOTE 4 It is also possible that the content of the unit of serialization does not itself represent an enterprise data object, but one or more of the contained entity instances (together with all the other entity instances to which it refers) constitutes an enterprise data object in its own right. In this case, the `edo` attribute should not appear on the unit of serialization element, but rather on each of those entity instances (see 9.3.1).

### 9.3 Representation of EXPRESS entity instances

Each EXPRESS entity instance in the unit of serialization shall be represented somewhere in the unit of serialization element by a by-value entity instance element, as specified in 9.3.1 or a complex entity instance element, as specified in 9.3.4. Exception: a unit of serialization can include entity instances defined in a separate unit of serialization (and represented by-value in a separate unit of serialization element), by external reference, as specified in 9.3.2.

An entity instance may be represented as an immediate child of the unit of serialization element, or in the value of an attribute of some other entity instance, or both.

An EXPRESS entity instance that is represented as an immediate child of the unit of serialization element is said to be an *independent entity instance.* Every independent entity instance shall be represented as a by-value entity instance element, as specified in 9.3.1.

An EXPRESS entity instance that is represented in the value of an attribute of some other entity instance is said to be a *dependent entity instance*. A dependent entity instance may be represented in any of several ways, as specified in 9.4 and 9.8, including the by-value representation specified in 9.3.1, the by-reference representation specified in 9.3.3, and the external representations specified in 9.3.2.

NOTE 1 The same entity instance may be represented both as a dependent entity instance and as an independent entity instance. The distinction is not in the nature of the entity instance, but rather in the place of occurrence.

NOTE 2 When an EXPRESS entity instance is the value of an attribute or a component of an aggregate value (a dependent entity instance), it may have a representation that is not an entity instance element. See 9.4 and 9.8.

When an EXPRESS entity instance is represented by an entity instance element (of any kind), it shall be represented by an entity instance element of the type corresponding to the EXPRESS entity data type that characterizes that instance (see 8.5.2). Exceptions:

— If the configuration directive **tag-source** applies to the EXPRESS entity data type that characterizes the instance, and the value of the **tag-source** attribute in the entity instance matches one of the corresponding **tag-values** names, the EXPRESS entity instance shall be represented by an entity instance element of the type corresponding to that dynamic subtype, as specified in 8.5.11. But if the value of the **tag-source** attribute in the instance does not match any of the **tag-values** names, an entity instance element that corresponds to the EXPRESS entity data type shall be used.

NOTE 3 When the value of the EXPRESS attribute designated by **tag-source** is one of the **tag-values** names, the corresponding dynamic subtype is the one that properly characterizes that entity instance.

— If no EXPRESS data type characterizes the instance, but the configuration file specified a synthetic entity data type that characterizes the instance (see 8.5.4.5), the instance shall be represented by an entity instance element of the type corresponding to the synthetic entity type.

— If no EXPRESS data type characterizes the instance, and no synthetic entity data type characterizes the instance, the instance shall be represented by a `exp:complexEntity` element, as specified in 9.3.4.

### 9.3.1 By-value representation of entity instances

The by-value instance element representing an entity instance shall have the XML element type corresponding, as specified in 8.5.5, to the EXPRESS entity data type that characterizes the instance (see 9.3 above). Exception: When the by-value instance element represents the entity instance as a value of an EXPRESS attribute of some other entity instance, and the configuration directive **exp-attribute ="attribute-tag"** applies to that EXPRESS attribute (see 10.5.1.3), the XML element type shall be as specified in 9.4.3.4.

NOTE 1 The EXPRESS entity data type that characterizes the instance may be a synthetic data type, as discussed above.

When the by-value entity instance element appears as an immediate child of the unit of serialization element, the entity instance element shall have a local XML identifier, and an `id` attribute whose value shall be that identifier.

When the by-value entity instance element (representing an *associated* entity instance) appears in the content of another by-value entity instance element (the *owning* entity instance):

 

— if the configuration directive **use-id="true"** applies to the associated entity instance (see 10.5.2), the by-value element corresponding to the associated entity instance shall have a local XML identifier and an `id` attribute whose value shall be that identifier.

— if the configuration directive **use-id="false"** applies, the `id` attribute shall not be present in the by-value element.

When the by-value entity instance element (representing a *single entity value*) appears in the content of an `exp:complexEntity` element, it shall be as specified in 9.3.4.2.

The `ref` attribute shall not be present in the by-value entity instance element.

The `proxy` attribute shall not be present in the by-value entity instance element.

The `href` attribute shall not be present in the by-value entity instance element.

The `edo` attribute may be present in the by-value entity instance element. If present, its value shall be some identifier for the entity instance that serves to identify it in some "enterprise namespace" (see 9.2.6.5).

NOTE 2 This Part of ISO 10303 does not specify the use of the `edo` attribute. It is intended to provide a universal identifier for the entity instance, as distinct from an identifier for a particular encoding of the entity instance. It may provide an identifier for a scope larger than the unit of serialization, or state the equivalence of the encoding in this unit of serialization with another encoding of the same entity instance.

The by-value entity instance element shall contain representations of the values of the EXPRESS attributes present in the entity instance, as specified in 9.3.1.1 through 9.3.1.4 below.

An EXPRESS entity instance whose by-value representation has an `id` attribute shall be representated by value only once in the unit of serialization.

NOTE 3 The `edo` attribute can be used to indicate that two by-value entity instance elements represent the same entity instance.

#### 9.3.1.1 Encoding of explicit attributes

For each EXPRESS mandatory explicit attribute of the entity data type that is not a **tag-source** attribute (see 10.2.17), and is not deleted by a **keep="false"** configuration directive (see 10.3.5 and 10.2.11), the by-value entity instance element shall contain representation of the value of that EXPRESS attribute in the instance, as specified in 9.4.

For each EXPRESS `OPTIONAL` attribute of the entity data type that is not inverted by an **inverse** configuration directive (see 10.3.5) and for which the entity instance has a value, the by-value entity instance element shall contain a corresponding representation of the value of that EXPRESS attribute in the instance, as specified in 9.4. If the entity instance has no value for that attribute, there shall be no corresponding XML element or XML attribute in the content of the entity instance element.

NOTE The handling of explicit attributes that are inverted by **inverse** directives is specified in 9.3.1.3. The handling of **tag-source** attributes is specified in 9.3.1.2.

EXAMPLE An example of an entity instance with an attribute whose value is also an entity instance:

In EXPRESS:

```
ENTITY dimensional_exponents;
  length_exponent: INTEGER;
```

```
  mass_exponent: INTEGER;
  time_exponent: INTEGER;
  electric_current_exponent: INTEGER;
  temperature_exponent: INTEGER;
  amount_of_substance_exponent: INTEGER;
  luminousIntensity_exponent: INTEGER;
END_ENTITY;

ENTITY named_unit;
  dimensions : dimensional_exponents;
  unit_type  : unit_enum;
  name       : STRING;
END_ENTITY;
```

In XML:

```
<Named_unit id="cubic_metre">
 <Dimensions>
   <Dimensional_exponents>
    <Length_exponent>3</Length_exponent>
    <Mass_exponent>0</Mass_exponent>
    <Time_exponent>0</Time_exponent>
    <Electric_current_exponent>0</Electric_current_exponent>
    <Temperature_exponent>0</Temperature_exponent>
    <Amount_of_substance_exponent>0</Amount_of_substance_exponent>
    <Luminous_intensity_exponent>0</Luminous_intensity_exponent>
   </Dimensional_exponents>
 </Dimensions>
 <Unit_type>volume_unit</Unit_type>
 <Name>cubic_metre</Name>
</Named_unit>
```

#### 9.3.1.2 Encoding of tag-source attributes

For the explicit attribute of the entity data type that is the **tag-source** attribute (see 10.2.17), if any, the explicit attribute shall be represented as follows:

— If the value of the **tag-source** attribute matches one of the names in the corresponding **tag-values** list, the by-value instance element shall not contain a representation of that attribute value, but the instance element tag shall be selected accordingly (see 9.3 above).

— If the value of the **tag-source** attribute does not match any name in the corresponding **tag-values** list, the by-value instance element shall contain a representation of that attribute value, as specified in 9.4.

#### 9.3.1.3 Encoding of INVERSE and inverted attributes

For each mandatory or `OPTIONAL` explicit attribute of the entity data type that is inverted by an **inverse** configuration directive (see 10.3.5), the explicit attribute shall be treated as if it were declared `INVERSE`, and represented as follows:

— If **keep="true"** applies to the explicit attribute (see 10.5.9), and the attribute has a value in the entity instance, the by-value instance element shall contain a representation of that attribute value, as specified in 9.4. However, every EXPRESS entity instance that appears in that attribute value shall be represented by-reference, that is, the configuration directive **content="ref"** shall always apply and override any conflicting directive.

— If **keep="false"** applies to the attribute, or the attribute has no value in the entity instance, the attribute shall not be represented in the by-value instance element.

NOTE The inverted attribute is effectively `SET [0:?] OF <X>`. Per the requirement above, when the set is empty – no other entity instance is actually related to this one in this way – the accessor does not appear as the empty set or have a nil value; it is omitted altogether.

For each `INVERSE` attribute of the entity data type that is inverted by an **inverse** configuration directive, and for each new explicit attribute of the entity data type that is created by an **inverse** configuration directive (see 10.3.5), the attribute shall be treated as if it were declared an explicit attribute of the entity data type, and the by-value entity instance element shall contain a representation of the value of the new/`INVERSE` attribute in the instance, as specified in 9.4.

Each `INVERSE` attribute of the entity data type that is not affected by an **inverse** configuration directive, shall be represented in the by-value entity instance element as follows:

— If **keep="true"** applies to the `INVERSE` attribute, and the attribute has a value in the entity instance, the by-value instance element shall contain a representation of that attribute value, as specified in 9.4. However, every EXPRESS entity instance that appears in that attribute value shall be represented by-reference, that is, the configuration directive **content="ref"** shall always apply and override any conflicting directive.

— If **keep="false"** applies to the attribute, or the attribute has no value in the entity instance, the attribute shall not be represented in the by-value instance element.

#### 9.3.1.4 Encoding of DERIVEd attributes

Each `DERIVE`d attribute of the entity data type shall be represented in the by-value entity instance element as follows:

— If the **keep="true"** applies to the `DERIVE`d attribute, and the attribute has a value in the entity instance that is available to the preprocessor, the by-value entity instance element shall contain a representation of that attribute value, as specified in 9.4. However, every EXPRESS entity instance that appears in that attribute value shall be represented by-reference, that is, the configuration directive **content="ref"** shall always apply and override any conflicting directive.

— If **keep="false"** applies to the `DERIVE`d attribute, or no value of the attribute is available to the preprocessor, the attribute shall not be represented in the by-value instance element.

### 9.3.2 External representation of EXPRESS entity instances

When the EXPRESS entity instance is described in a separate unit of serialization and represented by-value (along with other instances to which it refers) in the corresponding unit of serialization element, it may be included in the active unit of serialization element by *external reference*, as defined in this subclause. Every reference from an element in one unit of serialization to an element in another unit of serialization shall use an external reference, even when both unit of serialization elements are in the same XML document.

An external reference to an EXPRESS entity instance can take one of two forms: a by-path instance element, as specified in 9.3.2.2, or a by-proxy instance element, as specified in 9.3.2.1. No unit of serialization shall contain both forms: A unit of serialization shall either use by-proxy instance elements for all external references or use by-path instance elements for all external references.

NOTE 1 A by-path element is simpler, but when the unit of serialization will contain multiple references to the same externally represented entity instance, each by-path element will contain the complete external reference,

and each such element will have to be changed if the external entity instance is moved. As a consequence, by-path elements are most useful in referring to instance elements in different unit of serialization elements in the same complex document. By comparison, a single proxy element represents the external entity instance for all by-proxy references within the unit of serialization and supports more complex access mechanisms. This makes the by-proxy approach more useful for references into fixed external information sets. The purpose of the above exclusion rule is to avoid having a unit of serialization contain two different structures for reference to the same resource.

The term *external reference element* applies to both by-path instance elements and by-proxy instance elements, and all of the following apply to both:

An external reference element representing an entity instance shall have the XML element type corresponding, as specified in 8.5.5, to the EXPRESS entity data type that characterizes the (see 9.3 above). Exception: When the external reference element represents the entity instance as a value of an EXPRESS attribute of some other entity instance, and the configuration directive **exp-attribute="attribute-tag"** applies to that EXPRESS attribute, the XML element type shall be as specified in 9.4.3.3.

NOTE 2 The EXPRESS entity data type that characterizes the instance may be a synthetic data type, as discussed above.

The content of the external reference element shall be empty, and the external reference element shall have the attribute `xsi:nil="true"`.

The `id` attribute shall not be present in the external reference element.

The `ref` attribute shall not be present in the external reference element.

The `edo` attribute shall not be present in the external reference element.

#### 9.3.2.1 Representation by a by-proxy instance element

When the external reference element is a by-proxy instance element, this subclause shall apply.

The unit of serialization shall contain a proxy element corresponding to the EXPRESS data type of the entity instance, as specified in 8.5.8, that provides the complete reference to a by-value entity instance element (see 9.3.1) representing the entity instance in some other unit of serialization. The proxy element shall have a local XML identifier; the `id` attribute shall be present in the proxy element, and its value shall be that identifier. The `authority` attribute shall be present in the proxy element, and its value shall be a URI conforming to IETF RFC 2396 that locates a resource containing the by-value instance element. Where needed, the proxy element shall contain additional key attributes whose values are used to access the by-value instance element through that resource (see 8.5.8).

NOTE The unit of serialization to which the proxy element refers may appear in the current `iso_10303_28` element or in some other resource that is presumed to be available to the post-processor. In most cases, no additional key attributes are required.

The `edo` attribute may be present in the proxy element. If the `edo` attribute is present in the proxy element, it shall also be present in the by-value entity instance element to which the proxy element refers, and the values shall be the same.

The by-proxy instance element shall have the XML attribute `proxy`, and its value shall be the local XML identifier for the proxy element that provides the complete reference to the by-value representation of the entity instance.

The `href` attribute shall not be present in the by-proxy instance element.

 

EXAMPLE Using the EXPRESS schema in 9.3.1.1, and the by-value representation there as the by-value representation in the unit of serialization designated by `URI:starsupplier.com:units`, In XML:

```
<Named_unit_Key id="unit11"
   authority="URI:starsupplier.com:units"
   instance-id="cubic_metre" />

<Named_unit proxy="unit11" xsi:nil="true" />
```

#### 9.3.2.2 Representation by a by-path instance element

When the external reference element is a by-path instance element, this subclause shall apply.

The `proxy` attribute shall not be present in the by-path instance element.

The `href` attribute shall be present in the by-path instance element, and its value shall have the form: ***uri*#*id***, where

— ***uri*** is a URI conforming to IETF RFC 2396 that locates the resource containing the by-value instance element, and

— the fragment identifier ***id*** is an XML identifier local to that resource that identifies a by-value instance element of the same type as the by-path instance element.

EXAMPLE Using the EXPRESS schema in 9.3.1 and the by-value representation there as the by-value representation in the unit of serialization designated by `URI:starsupplier.com:units`, In XML:

```
<Named_unit href="ftp://ftp.starsupplier.com/STEP/units#cubic_metre"
  xsi:nil="true" />
```

### 9.3.3 By-reference representation of EXPRESS entity instances

The by-reference representation of an EXPRESS entity instance may be used when the entity instance is represented by-value elsewhere in the unit of serialization element. The by-reference representation shall be used only when the entity instance element is contained in a by-value representation of another entity instance.

The by-reference instance element representing an entity instance shall have the XML element type corresponding, as specified in 8.5.5, to the EXPRESS entity data type that characterizes the instance (see 9.3 above). Exception: When the configuration directive **exp-attribute="attribute-tag"** applies to the EXPRESS attribute whose value is being represented by the by-reference entity instance, the XML element shall be as specified in 9.4.3.3.

NOTE 1 The EXPRESS entity data type that characterizes the instance may be a synthetic data type, as discussed above in 9.3 above; or the `exp:complexEntity` instance element may be used, as specified in 9.3.4.1..

The content of the by-reference instance element shall be empty, and the by-reference instance element shall have the attribute `xsi:nil="true"`.

The `id` attribute shall not be present in the by-reference entity instance element.

The `proxy` attribute shall not be present in the by-reference entity instance element.

The `href` attribute shall not be present in the by-reference entity instance element.

The edo attribute shall not be present in the by-reference entity instance element.

The unit of serialization shall (somewhere) contain a by-value entity instance element that represents the entity instance. The by-value entity instance element shall have a local XML identifier; as specified in 9.3.1. The ref attribute shall be present in the by-reference entity instance element, and its value shall be that local XML identifier.

NOTE 2 No by-reference instance element can refer to a by-value instance element that is contained in another by-value instance element, unless **use-id="true"** applies to that usage (see 9.3.1).

NOTE 3 The value of the ref XML attribute always refers to a by-value instance element within the same unit of serialization. Every reference from an element in one unit of serialization to an element in another uos uses an external reference element, as specified in 9.3.2, even when both unit of serialization elements are in the same XML document.

EXAMPLE Using the EXPRESS schema in 9.3.1.1 and the by-value representation there as the by-value representation of the "cubic_metre" entity instance elsewhere in this unit of serialization, the "cubic_metre" entity instance is represented as the value of an attribute of a measure_with_unit entity:

In EXPRESS:

```
ENTITY measure_with_unit;
  measure_value : REAL;
  unit : named_unit;
END_ENTITY;
```

In XML:

```
<Measure_with_unit id="m304" >
  <Measure_value>416.78</Measure_value>
  <Unit>
   <Named_unit ref="cubic_metre" xsi:nil="true" />
  </Unit>
</Measure_with_unit>
```

### 9.3.4 Complex entity representation of EXPRESS entity instances

Complex entity representation shall be used only for an entity instances that is not characterized by any entity data type in the context schema and not characterized by any synthetic entity data type.

NOTE This feature is analogous to the "external mapping" feature of ISO 10303-21.

An uncharacterized complex entity instance shall be represented either by reference or by value, as specified below. There is no external representation for an uncharacterized entity instance.

#### 9.3.4.1 By-reference representation of uncharacterized entity instances

When the complex entity instance is represented by reference, the representation shall be as specified in 9.3.3, with the following exceptions:

— The exp:complexEntity instance element shall be used instead of the (non-existent) entity instance element that characterizes the instance.

— The corresponding by-value instance element shall be as specified in 9.3.4.2 below.

 

#### 9.3.4.2 By-value representation of uncharacterized entity instances

The by-value instance element representing an uncharacterized entity instance shall have the XML element type `exp:complexEntity` (see 8.5.3.5). Exception: When the by-value instance element represents the entity instance as a value of an EXPRESS attribute of some other entity instance, and the configuration directive **exp-attribute ="attribute-tag"** applies to that EXPRESS attribute (see 10.5.1.3), the XML element type shall be as specified in 9.4.3.4.

When the by-value `exp:complexEntity` element appears as an immediate child of the unit of serialization element, the element shall have a local XML identifier, and an `id` attribute whose value shall be that identifier.

When the by-value `exp:complexEntity` element (representing an *associated* entity instance) appears in the content of another by-value entity instance element (the *owning* entity instance):

— if the configuration directive **use-id="true"** applies to the associated entity instance (see 10.5.2), the `exp:complexEntity` element shall have a local XML identifier and an `id` attribute whose value shall be that identifier.

— if the configuration directive **use-id="false"** applies, the `id` attribute shall not be present in the `exp:complexEntity` element.

The following attributes shall not be present in the by-value `exp:complexEntity` element: `ref`, `proxy`, `href`.

The `entities` attribute shall be present in the `exp:complexEntity` element. Its value shall be a space separated list of the XML names corresponding to the *leaf* nodes in the type graph associated with the complex entity instance (see 8.5.3).

The `edo` attribute may be present in the `exp:complexEntity` element. If present, its value shall be some identifier for the entity instance that serves to identify it in some "enterprise namespace" (see 9.2.6.5).

NOTE 1 This Part of ISO 10303 does not specify the use of the `edo` attribute. It is intended to provide a universal identifier for the entity instance, as distinct from an identifier for a particular encoding of the entity instance. It may provide an identifier for a scope larger than the unit of serialization, or indicate the equivalence of the encoding in this unit of serialization with another encoding of the same entity instance.

An EXPRESS entity instance whose by-value representation has an `id` attribute shall be representated by value only once in the unit of serialization.

NOTE 2 The `edo` attribute can be used to indicate that two by-value `exp:complexEntity` elements represent the same entity instance.

The content of the by-value `exp:complexEntity` element shall be as follows:

For each entity data type instantiated in the complex entity instance that is a *root* of the associated type graph (see 8.5.3), the `exp:complexEntity` element shall contain a by-value entity instance element. The contained by-value element shall have the XML element type corresponding, as specified in 0, to the root entity data type. It shall have none of the following XML attributes: `id`, `ref`, `href`, `proxy`, `edo`. It shall contain representations of the values of those EXPRESS attributes that were declared in the EXPRESS entity declaration for the root entity and are present in the entity instance. Each such attribute shall be represented as specified in 9.3.1.1 through 9.3.1.4.

For each *subtype* instantiated in the complex entity instance, that is, each entity data type that is not a root of the associated type graph, the `exp:complexEntity` element shall contain a single entity value element. The single entity value element shall have the XML element type corresponding, as specified in 8.5.7, to the subtype. It shall contain representations of the values of those EXPRESS attributes that were declared in the EXPRESS entity declaration for the subtype and are present in the entity instance. Each such attribute shall be represented as specified in 9.3.1.1 through 9.3.1.4.

## 9.4 Representation of an EXPRESS attribute

This subclause specifies the rules for representing the value of an EXPRESS attribute in the by-value entity instance element that represents the entity instance. The representation depends on the configuration directives that apply to the EXPRESS attribute, if any, and the data type of the EXPRESS attribute.

The value of the EXPRESS attribute may be represented *by-reference* or *by-value*, according to the configuration directives that apply, as specified in 9.4.1.

If the configuration directive **exp-attribute="attribute-content"** applies to the EXPRESS attribute, the value of the EXPRESS attribute shall be encoded as specified in 9.4.2.

If the configuration directive **exp-attribute="attribute-tag"** applies to the EXPRESS attribute, the value of the EXPRESS attribute shall be encoded as specified in 9.4.3.

If the configuration directive **exp-attribute="double-tag"** applies to the EXPRESS attribute, the value of the EXPRESS attribute shall be encoded as specified in 9.4.4.

If the configuration directive **exp-attribute="type-tag"** applies to the EXPRESS attribute, the value of the EXPRESS attribute shall be represented as specified in 9.4.5.

If the configuration directive **exp-attribute="no-tag"** applies to the EXPRESS attribute, the value of the EXPRESS attribute shall be represented as specified in 9.4.6.

NOTE **exp-attribute="no-tag"** can only apply to an EXPRESS attribute whose value is an entity instance.

### 9.4.1 Determining by-reference or by-value representation

The value of the EXPRESS attribute may be represented *by-reference* or *by-value*, according to the configuration directives that apply, as follows:

— If the configuration directive **content="value"** applies to the EXPRESS attribute, or the configuration directive **content="unspecified"** applies to the EXPRESS attribute and **exp-type="value"** applies to the data type of the value of the EXPRESS attribute, the representation of the value shall be by-value.

— If the configuration directive **content="ref"** applies to the EXPRESS attribute, or the configuration directive **content="unspecified"** applies to the EXPRESS attribute and **exp-type="root"** applies to the data type of the value of the EXPRESS attribute, the representation of the value shall be by-reference.

— If the configuration directive **content="unspecified"** applies to the EXPRESS attribute and **exp-type="unspecified"** applies to the data type of the value of the EXPRESS attribute, the representation of the value may be either by-reference or by-value.

NOTE The actual data type of the value, rather than the declared data type of the attribute, determines the representation rule when content="unspecified" applies to the attribute.

 

Further requirements for by-reference and by-value representation of a given value depend on the EXPRESS data type of the value, and on the value of the configuration directive **exp-attribute** that applies (see 10.5.1), as specified in the subclauses below.

### 9.4.2 Representation of EXPRESS attribute value as accessor attribute

If the configuration directive **exp-attribute="attribute-content"** applies to the EXPRESS attribute, the EXPRESS attribute shall be represented in the owning entity instance element by the corresponding accessor attribute (see 8.6).

The value of the EXPRESS attribute shall be encoded as follows:

— If the data type of the EXPRESS attribute is an entity data type, the unit of serialization shall (somewhere) contain a by-value entity instance element; as specified in 9.3.1, or a proxy element, as specified in 9.3.2.1, that represents the associated entity instance. The by-value entity instance element or proxy element shall have a local XML identifier; and the value of the accessor attribute shall be that local XML identifier.

— If the data type of the EXPRESS attribute is *not* an entity data type and by-reference representation of the value is *required* by 9.4.1, the value shall be encoded in a by-value instance element, as specified in 9.10. The by-value instance element shall be an immediate child of the unit of serialization element and shall have a local XML identifier. The value of the accessor attribute shall be that local XML identifier.

— If the data type of the EXPRESS attribute is a primitive data type (`BINARY`, `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER`, `REAL` or `STRING`): and by-value representation is *permitted* by 9.4.1, the value shall be encoded in the value of the accessor attribute according to its data type, as specified in 9.5.

— If the data type of the EXPRESS attribute is an aggregation data type to which the list-of-values representation applies (see 8.2.2) and by-value representation is *permitted* by 9.4.1, the value shall be encoded in the value of the accessor attribute as specified in 9.8.1.

— If the data type of the EXPRESS attribute is an aggregation data type to which any representation other than list-of-values applies (see 8.2.2), including all aggregate of aggregate representations, the value shall be encoded in a by-value instance element, as specified in 9.10. The instance element shall be an immediate child of the unit of serialization element and shall have a local XML identifier. The value of the accessor attribute shall be that local XML identifier.

— If the data type of the EXPRESS attribute is an EXPRESS defined data type whose fundamental type is an `ENUMERATION` data type and by-value representation is *permitted* by 9.4.1, the value shall be encoded in the value of the accessor attribute as specified in 9.6.

— If the data type of the EXPRESS attribute is an EXPRESS defined data type whose fundamental type is a `SELECT` data type, the value shall be encoded in a by-value instance element, as specified in 9.7. The instance element shall be an immediate child of the unit of serialization element and shall have a local XML identifier. The value of the accessor attribute shall be that local XML identifier.

— If the data type of the EXPRESS attribute is a defined data type whose fundamental type is a primitive type or an aggregation data type, the value shall be encoded as specified above for the fundamental type.

NOTE 1 For accessor attributes, 9.4.1 does not affect the encoding of the attribute value when it is an entity instance, when it is a value of a `SELECT` data type, or when it is an aggregate value that does not use the list-of-values form. In those cases, the representation is always by-reference. And otherwise, whenever 9.4.1 permits the representation to be by-value, it is *required* to be by-value.

NOTE 2 When the value is represented by-reference, the value of the accessor attribute always refers to a by-value instance element within the same unit of serialization.

NOTE 3 Unless the reference is to an entity instance to which **use-id="true"** applies (see 9.3.1), the by-value instance element is always an immediate child of the unit of serialization – No accessor attribute can refer to a by-value entity instance element that is contained in another by-value instance element.

NOTE 4 A reference from an accessor attribute in one unit of serialization to an entity instance element in another unit of serialization requires a proxy element, even when both unit of serialization elements are in the same XML document. By-path representation (see 9.3.2.2) is not supported by accessor attributes.

### 9.4.3 Attribute-tag representation of EXPRESS attribute value

If the configuration directive **exp-attribute="attribute-tag"** applies to the EXPRESS attribute, the attribute value shall be represented as specified in this subclause.

If the value is an entity instance:

— If by-reference representation of the value is permitted by 9.4.1, the entity instance shall be encoded as specified in 9.4.3.3.

— If by-value representation of the value is required by 9.4.1, the entity instance shall be encoded as specified in 9.4.3.4.

If the value is not an entity instance:

— If by-reference representation of the value is required by 9.4.1, or by-reference representation of the value is permitted by 9.4.1 and the pre-processor elects to use by-reference representation, the value shall be encoded as specified in 9.4.3.1.

— If by-value representation of the value is required by 9.4.1, or by-value representation of the value is permitted by 9.4.1 and the pre-processor elects to use by-value representation, the value shall be encoded as specified in 9.4.3.2.

NOTE Except in special cases, the pre-processor should use by-value representation for values that are not entity instances.

#### 9.4.3.1 Attribute-tag representation by-reference

The value shall be encoded in a by-value instance element of the type corresponding to its actual EXPRESS data type. If the value is an entity instance, the by-value instance element shall be as specified in 9.3.1; otherwise the by-value instance element shall be as specified in 9.10.1. In either case, the instance element shall be an immediate child of the unit of serialization element and shall have a local XML identifier.

NOTE The instance element corresponds to the actual data type of the value, not necessarily to the declared data type of the EXPRESS attribute. In particular, instances of an entity data type are encoded according to their characterizing type, and instances of a `SELECT` type are encoded according to their actual data type, but values interpreted as values of a defined data type use the instance element corresponding to the defined data type.

The EXPRESS attribute shall be represented in the owning entity instance element by the corresponding accessor element (see 8.6).

The accessor element shall be empty (have no content). If the EXPRESS data type for the attribute is not a `SELECT` data type, the accessor element shall have the XML attribute `xsi:nil="true"`.

 

The accessor element shall have the XML attribute `ref`, and its value shall be the local XML identifier for the instance element that encodes the value.

The accessor element shall have no other XML attributes that describe the value. Where appropriate, such attributes shall always appear in the by-value instance element.

#### 9.4.3.2 Attribute-tag representation by-value

The EXPRESS attribute shall be represented in the owning entity instance element by the corresponding accessor element (see 8.6). The value of the EXPRESS attribute shall be encoded in the content of that accessor element, according to the EXPRESS data type of the attribute, as follows:

— If the data type is a primitive data type (`BINARY`, `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER`, `REAL` or `STRING`), the value shall be encoded as specified in 9.5.

— If the data type is an aggregation data type, the value shall be encoded as specified in 9.8.

— If the data type is a defined data type whose fundamental type is `BINARY`, `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER`, `REAL`, or `STRING`, the value shall be encoded as specified for the fundamental type in 9.5.

— If the data type is a defined data type whose fundamental type is an aggregation data type (`ARRAY`, `BAG`, `LIST`, or `SET`), the value shall be encoded as specified for the fundamental type in 9.8.

— If the data type is a defined data type whose fundamental type is an `ENUMERATION` data type, the value shall be encoded as specified in 9.6.

— If the data type is a defined data type whose fundamental type is an `SELECT` data type, the value shall be encoded in a by-value instance element corresponding to the actual data type of the value, as specified in 9.10.1.

NOTE An EXPRESS attribute whose data type is nominally a `SELECT` data type is effectively represented as **"double-tag"** when represented "by-value".

#### 9.4.3.3 Attribute-tag representation of entity instances by-reference

The EXPRESS attribute shall be represented in the owning entity instance element by an accessor element of the type corresponding to the EXPRESS attribute (see 8.6). The accessor element shall represent the associated entity instance as if it were a by-reference instance element, as specified in 9.3.3, or an external reference element, as specified in 9.3.2, in every regard except the element tag. That is, it shall have empty content and exactly the XML attribute values that are specified in 9.3.2 or 9.3.3.

If the representation is as an external reference element, and the data type that characterizes the associated entity instance does not correspond directly to the declared XML data type of the accessor element,, the element shall have the, and its value shall be the name of the XML data type that corresponds to the EXPRESS data type that characterizes the associated entity instance, as specified in 8.5.3.5.

If the representation is as if it were a by-reference element, the accessor element need not contain the `xsi:type` XML attribute.

#### 9.4.3.4 Attribute-tag representation of entity instances by value

The EXPRESS attribute shall be represented in the owning entity instance element by an accessor element of the type corresponding to the EXPRESS attribute (see 8.6). The accessor element shall represent the associated entity instance exactly as if it were a by-value instance element, as specified in 9.3.1, with the following exceptions:

— The element tag shall correspond to the owning attribute and not to the data type of the associated entity instance.

— If the data type that characterizes the associated entity instance does not correspond directly to the declared XML data type of the accessor element, the accessor element shall have the `xsi:type` XML attribute, and its value shall be the qualified name of the XML data type that corresponds to the EXPRESS data type that characterizes the associated entity instance, as specified in 8.5.3.5.

— The `id` attribute shall not be present in the element, regardless of the setting of **use-id**.

### 9.4.4 Double-tag representation of EXPRESS attribute value

If the configuration directive **exp-attribute="double-tag"** applies to the EXPRESS attribute, the EXPRESS attribute shall be represented in the owning entity instance element by the corresponding accessor element (see 8.6). The content of the accessor element shall be a single instance element that represents the value of the EXPRESS attribute.

If by-reference representation of the value is required by 9.4.1, or by-reference representation of the value is permitted by 9.4.1 and the pre-processor elects to use by-reference representation, the value shall be encoded as specified in 9.4.4.1.

If by-value representation of the value is required by 9.4.1, or by-value representation of the value is permitted by 9.4.1 and the pre-processor elects to use by-value representation, the value shall be encoded as specified in 9.4.4.2.

#### 9.4.4.1 Double-tag representation by-reference

The EXPRESS attribute shall be represented in the owning entity instance element by the corresponding accessor element (see 8.6).

If the value is an entity instance (called the associated entity instance), the content of the accessor element shall be a single by-reference instance element that represents the value as specified in 9.3.3, or an external reference instance element as specified in 9.3.2.

If the value is not an entity instance, the content of the accessor element shall be a single by-reference instance element that represents the value as specified in 9.10.2.

#### 9.4.4.2 Double-tag representation by-value

The EXPRESS attribute shall be represented in the owning entity instance element by the corresponding accessor element (see 8.6).

If the value is an entity instance, the content of the accessor element shall be a single by-value instance element that represents the associated entity instance as specified in 9.3.1.

NOTE In this case, the presence of the `id` attribute depends on the configuration directive **use-id**.

 

If the value is not an entity instance, the content of the accessor element shall be a single by-value instance element that represents the value as specified in 9.10.1.

### 9.4.5 Type tag representation of EXPRESS attribute value

If the configuration directive **exp-attribute="type-tag"** applies to the EXPRESS attribute (see 10.5.1.2), the value of the EXPRESS attribute shall be represented in the content of the entity instance element by zero or more instance elements.

NOTE 1 There is no accessor element for type-tagged values. The accessor element is replaced by zero or more instance elements representing the value of the attribute. That is, the accessor element is replaced by what would otherwise have been its content.

NOTE 2 **exp-attribute="type-tag"** only applies to EXPRESS attributes whose data types are named data types or aggregates of named data types, and the instance elements will be tagged with the corresponding XML names.

If the value of the EXPRESS attribute is a value of an anonymous aggregation data type and has no components, the value shall not be represented in the content of the entity instance element, that is, it is represented by zero instance elements.

If the value of the EXPRESS attribute is a single entity value, it shall be represented in the content of the owning entity instance element by an entity instance element that corresponds to the value. If by-reference representation of the value is required by 9.4.1, the value shall be encoded as specified in 9.3.2 or 9.3.3. If by-value representation of the value is required by 9.4.1, the value shall be encoded as specified in 9.3.1. If both are permitted, the value shall be encoded using one of the representations specified in 9.3, at the discretion of the pre-processor.

If the value of the EXPRESS attribute is a single instance of a defined data type, it shall be represented in the content of the owning entity instance element by an instance element that corresponds to the value. If by-value representation of the value is required by 9.4.1, the value shall be encoded as specified in 9.10.1. If by-reference representation of the value is required by 9.4.1, the value shall be encoded as specified in 9.10.2. If both are permitted, the value shall be encoded as specified in either 9.10.1 or 9.10.2, at the discretion of the pre-processor.

If the value of the EXPRESS attribute is an aggregate value having at least one component, it shall be represented in the content of the owning entity instance element by a sequence of instance elements, each representing one of its component values as specified in 9.8.7. For the purposes of 9.8.7, the value of configuration directive **content** that applies to the EXPRESS attribute shall be interpreted as applying to all of the component values.

NOTE 3 For type-tagged attributes, the configuration directive **content** is interpreted as applying to every component of the aggregate value instead of to the aggregate value.

NOTE 4 Except in special cases, when the pre-processor has a choice, it should use by-value representation for values that are not entity instances.

NOTE 5 Unless otherwise specified by configuration directives, there is no requirement for the sequence of instance elements to be of the same kind. That is, some could be by-value representations, some by-reference, and some external.

NOTE 6 Because entity instances that satisfy the data type of the EXPRESS attribute may belong to different subtypes, the XML tag on the entity instance element need not correspond directly to the data type of the EXPRESS attribute, and the XML tags for different components of the aggregate value may be different. Similarly, when the defined data type is a `SELECT` data type, each value will be encoded as an instance of the EXPRESS data type that it satisfies. The XML tag on the instance element will not correspond directly to the

data type of the EXPRESS attribute, but rather to one of the members of the working select list, and the XML tags for different components of the aggregate value may be different.

### 9.4.6 No-tag representation of entity instance as EXPRESS attribute value

If the configuration directive **exp-attribute="no-tag"** applies to the EXPRESS attribute (see 10.5.1.5), the value of the EXPRESS attribute shall be an entity instance, referred to as the *associated entity instance*, to distinguish it from the entity instance that owns the attribute, referred to as the *owning entity instance*. The associated entity instance shall be represented in the content of the owning entity instance element by a sequence of accessor elements or accessor attributes that correspond to the EXPRESS attributes of the associated entity instance. That is, the owning entity instance shall have an accessor for each EXPRESS attribute of the associated entity instance, and the associated entity instance shall not have a distinct representation. The value of each EXPRESS attribute of the associated instance shall be encoded, as specified in 9.3.1, as if it were the value of a corresponding attribute of the owning entity instance.

NOTE There is no accessor element for no-tagged values. The accessor element is replaced by the accessor elements corresponding to the attributes of the associated entity instance. That is, the accessor element is replaced by what would otherwise have been its content.

### 9.4.7 No-tag-simple representation of entity instance as EXPRESS attribute value

If the configuration directive **exp-attribute="no-tag-simple"** applies to the EXPRESS attribute (see 10.5.1.6), the value of the EXPRESS attribute shall be a XML text node.

## 9.5 Representation of simple values

This subclause specifies the rules for representation of values of simple data types, as either the content of an XML element or as the value of an XML attribute.

NOTE 1 The representations of values of all EXPRESS data types are constructed from the representations of simple values.

Each value shall be represented according to its nominal EXPRESS data type, as specified below.

If a **map** configuration directive applies to the value (see 10.5.4), the value shall be represented as specified in *XML Schema Part 2: Datatypes* for the XML data type specified by the **map** directive. The effects of the allowable **map** directives on the representations of values of specific EXPRESS data types are described for each EXPRESS data type below. If the actual value cannot be represented as a valid value of the specified XML data type, the value shall be encoded according to its EXPRESS data type, as specified below, but the resulting XML document will not validate under the configured XML schema.

NOTE 2 In most cases, the **map** directive affects only the XML data type specified in the XML schema and not the representation of any given value. That is, all of the XML schema data types that are permitted by 10.2.13 to represent a given EXPRESS data type use a common representation and differ only in the allowable values. So the representation of a given value will be the same for all of the XML schema data types that may be specified by the **map** directive, but not all values of the (fundamental) EXPRESS data type are valid values of all of the specifiable XML schema data types. The purpose of the **map** directive is to allow the XML schema to capture constraints on the values of the EXPRESS data type or attribute that may be phrased by `WHERE` rules in the EXPRESS schema or stated in implementor agreements.

 

### 9.5.1 Representation of BINARY values

A value of the EXPRESS `BINARY` data type shall be represented as specified for XML data type `hexBinary` in *XML Schema Part 2: Datatypes*, or as specified for XML data type `base64Binary` in *XML Schema Part 2: Datatypes*, if required by the **map** configuration directive (see 10.2.13).

If the length of the `BINARY` value, seen as a bit string, is not a multiple of eight, the value shall be so encoded that the high-order bits of the last octet(s) are significant and the low-order bits of the last octet(s) are zeros.

The `extrabits` XML attribute shall appear in the XML element whose content is the `BINARY` value. The value of the `extrabits` XML attribute shall specify the number of low-order bits (0 to 7) of the last encoded octet that are not to be interpreted as part of the `BINARY` value. Exception: The `extrabits` `its` XML attribute need not appear if its value is '0'. That is, if the `extrabits` attribute does not appear, every bit in the last encoded octet shall be interpreted as part of the `BINARY` value.

### 9.5.2 Representation of BOOLEAN values

A value of EXPRESS data type `BOOLEAN` shall be represented as specified for XML datatype `boolean` in *XML Schema Part 2: Datatypes*.

NOTE The specified representation for the value `TRUE` is the character sequence '`true`'. The specified representation for the value `FALSE` is the character sequence '`false`'.

EXAMPLE The following example illustrates the mapping of EXPRESS attributes of type `BOOLEAN` to their corresponding XML representation:

In EXPRESS:

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

XML instance data without **exp-attribute="attribute-content"**:

```
<Car id="i100">
 <Airbag>true</Airbag>
 <Sunroof>0</Sunroof>
</Car>
```

XML instance data with **exp-attribute="attribute-content"**:

```
<Car id="i100" Airbag="true" Sunroof="0" />
```

### 9.5.3 Representation of INTEGER values

A value of EXPRESS data type `INTEGER` shall be represented in the form specified for the `integer` data type in *XML Schema Part 2: Datatypes*.

NOTE the representation form consists of 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.

If the XML schema data type corresponding to the EXPRESS data type is specified by the **map** configuration directive (see 10.2.13) to be `long`, or any of the other XML schema subtypes of `integer`, the integer value shall lie within the bounds specified for that XML schema data type. If

the actual value cannot be represented as a valid value of the specified XML data type, the value shall be encoded as specified for `integer`, but the resulting XML document will not validate under the configured XML schema.

EXAMPLE 1 The following are valid literals that represent values of an EXPRESS attribute of data type `INTEGER`:

```
-1
0
12678967543233
```

EXAMPLE 2 The following example illustrates the mapping of EXPRESS attributes of type `INTEGER` into their corresponding XML data:

In EXPRESS:

```
ENTITY car;
  mileage : INTEGER;
  weight  : OPTIONAL INTEGER;
END_ENTITY;
```

XML instance data without **exp-attribute="attribute-content"**:

```
<Car id="i100">
 <Mileage>14345</Mileage>
 <Weight>2300</Weight>
</Car>
```

XML instance data with **exp-attribute="attribute-content"**:

```
<Car id="i100" Mileage="14345" Weight="2300" />
```

### 9.5.4 Representation of LOGICAL values

A value of EXPRESS data type `LOGICAL` shall be represented as follows:

— The character sequence '`false`' shall represent the `LOGICAL` value `FALSE`;

— The character sequence '`true`' shall represent the `LOGICAL` value `TRUE`;

— The character sequence '`unknown`' shall represent the `LOGICAL` value `UNKNOWN`.

EXAMPLE The following example illustrates the mapping of EXPRESS attributes of type `LOGICAL` into their corresponding XML representation:

In EXPRESS:

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

XML instance data with out **exp-attribute="attribute-content"**:

```
<Car id="i100">
 <Front_airbag>true</Front_airbag>
 <Side_airbag>unknown</Side_airbag>
```

 

```
</Car>
```

XML instance data with **exp-attribute="attribute-content"**:

```
<Car id="i100" Front_airbag="true" Side_airbag="unknown" />
```

### 9.5.5 Representation of NUMBER values

A value of EXPRESS data type `NUMBER` shall be represented as specified for datatype `decimal` in *XML Schema Part 2: Datatypes*, unless otherwise specified by a **map** configuration directive (see 10.2.13).

NOTE The general form of content data conforming to the XML schema `decimal` data type 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.

If the XML schema data type corresponding to the EXPRESS data type is specified by the **map** configuration directive to be `double` or `float`, the value shall be encoded as specified for data type `REAL` (see 9.5.6 ).

### 9.5.6 Representation of REAL values

A value of EXPRESS data type `REAL` shall be represented as specified for XML data type `double` in *XML Schema Part 2: Datatypes*, unless otherwise specified by a **map** configuration directive (see 10.2.13).

NOTE The general form of content data conforming to the XML schema `double` and `float` data types is the same as that specified for XML data type `decimal` (see the note to 9.5.5), except that the range of exponents is limited, and the `double` and `float` data types also support representations of infinite values and exceptional values. The specified representation for a positive infinite value is the character sequence '`inf`'. The specified representation for a negative infinite value is the character sequence '`-inf`'. The specified representation for an exceptional value is the character sequence '`NaN`'.

If the XML schema data type corresponding to the EXPRESS data type is specified by the **map** configuration directive to be `decimal`, the value shall be represented as specified for data type `NUMBER` in 9.5.5.

If the XML schema data type corresponding to the EXPRESS data type is `double` or `float`, the `REAL` value shall lie within the bounds specified for that XML schema data type. If the actual value cannot be represented as a valid value of the specified XML data type, the value shall be encoded as specified for `decimal`, but the resulting XML document will not validate under the configured XML schema.

For the purposes of this Part of ISO 10303, any EXPRESS data type derived from `REAL` may be considered to be extended by infinite values and exceptional values, so long as that data type is specified to be represented by `double` or `float` . In such cases, the infinite and exceptional values shall be represented as specified for the `double` or `float` data types in *XML Schema Part 1: Structures*.

EXAMPLE 1 The following are valid literals that may be assigned to an XML attribute that represents a value of EXPRESS type `REAL` or `NUMBER`:

```
-1E+4
```

```
1267.43233E+12
12.78e-2
12
inf
```

EXAMPLE 2 The following example illustrates the mapping of EXPRESS attributes of type NUMBER and REAL into their corresponding XML data:

In EXPRESS:

```
ENTITY car;
   power   : NUMBER;
   mileage : REAL;
   weight  : OPTIONAL REAL;
END_ENTITY;
```

XML instance data without **exp-attribute="attribute-content"**:

```
<Car id="i100">
 <Power>320</Power>
 <Mileage>14345.7</Mileage>
 <Weight>2.3E+3</Weight>
</Car>
```

XML instance data with **exp-attribute="attribute-content"**:

```
<Car id="i100" Power="320" Mileage="14345.7" Weight="2.3E+3" />
```

### 9.5.7 Representation of STRING values

A value of the EXPRESS data type STRING shall be represented as specified for XML data type normalizedString in *XML Schema Part 2: Datatypes*.

NOTE 1 While a STRING value may be specified by a **map** configuration directive (see 10.2.13) to use some XML schema data type that is a specialization of normalizedString, that doesn't affect the encoding of the individual STRING values. It may affect the representation of aggregate values whose base-type is, or is derived from, STRING (see 9.8)

The characters shall be encoded using whatever character encoding is specified in the XML processing instruction(s). Exception: if the **notation** configuration directive applies to the value (see 10.5.8) the value shall be encoded using the specified notation.

The representation shall be any sequence of the following:

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

— a valid XML text string in which every occurrence of the character '<' is replaced by '<' and every occurrence of the character '&' is replaced by '&'.

NOTE 2 The character sequence ']]>' cannot appear in a CDATA section.

The XML value may also contain the control codes RETURN (0D) and NEW LINE (0A) but these control codes shall not be considered part of the EXPRESS STRING value. No other control codes (00-1F) shall appear, except those that are interpreted as character code selection sequences in the specified character encoding.

 

Characters that are valid characters in an EXPRESS STRING value but not valid in XML data type normalizedString shall be mapped to a Private Use Unicode Hex value, as specified in Table 6.

**Table 6 — Representation of EXPRESS characters invalid in XML normalizedString**

| EXPRESS character | UNICODE hex representation |
|---|---|
| 08 (BACKSPACE) | F0000 |
| 0B (VERTICAL TAB) | F0001 |
| 0C (PAGE FEED) | F0002 |

EXAMPLE The following example illustrates the mapping of EXPRESS attributes of type STRING into their corresponding XML representation:

In EXPRESS:

```
ENTITY car;
  make : STRING;
  car_model : OPTIONAL STRING;
  options : LIST OF STRING;
END_ENTITY;
```

XML instance data with out **exp-attribute="attribute-content"**:

```
<Car id="i10">
  <Make>Ford</Make>
  <Car_model>14345.7</Car_model>
  <Options>
    <exp:string-wrapper>Leather upholstery</exp:string-wrapper >
    <exp:string-wrapper >Premium sound package</exp:string-wrapper >
  </Options>
</Car>
```

XML instance data with **exp-attribute="attribute-content"**:

```
<Car id="i10" Make="Ford" Car_model="Explorer" >
  <Options>
    <exp:string-wrapper >Leather upholstery</exp:string-wrapper >
    <exp:string-wrapper >Premium sound package</exp:string-wrapper >
  </Options>
</Car>
```

## 9.6 Representation of enumeration items

A value of an EXPRESS ENUMERATION data type — an enumeration item — shall be represented as the XML string value that corresponds to that enumeration item, as specified in 8.3.3.

EXAMPLE The following example illustrates the mapping of an EXPRESS attribute whose data type is an ENUMERATION type into its corresponding XML representation:

In EXPRESS:

```
TYPE b_spline_curve_form = ENUMERATION OF (
  elliptic_arc, circular_arc, parabolic_arc, hyperbolic_arc,
```

```
   polyline_form, unspecified);
END_TYPE;

ENTITY b_spline_curve;
   degree : INTEGER;
   curve_form : b_spline_curve_form;
   control_points : LIST [2:?] OF pt3d;
END_ENTITY;
```

XML instance data without **exp-attribute="attribute-content"**:

```
<B_spline_curve id="b106">
   <Degree>2</Degree>
   <Curve_form>elliptic_arc</Curve_form>
   <Control_points>
      <Pt3d>...</Pt3d>
      <Pt3d>...</Pt3d>
      <Pt3d>...</Pt3d>
   </Control_points>
</B_spline_curve>
```

XML instance data with **exp-attribute="attribute-content"**:

```
<B_spline_curve id="b106" Degree="2" Curve_form="elliptic_arc" >
   <Control_points>
      <Pt3d>...</Pt3d>
      <Pt3d>...</Pt3d>
      <Pt3d>...</Pt3d>
   </Control_points>
</B_spline_curve>
```

## 9.7 Representation of values of SELECT types

A value of an EXPRESS defined data type whose fundamental type is a `SELECT` data type shall be represented as an instance element (see 8.4) that corresponds to the EXPRESS data type of the actual value.

NOTE 1 The EXPRESS data type of the value is always a defined data type or an entity data type.

The instance element shall be a valid member of the XML schema `group` that corresponds to the fundamental `SELECT` type of the defined data type (see 8.3.4.2).

NOTE 2 If the EXPRESS defined data type that is nominally being represented is a specialization of a `SELECT` data type, its corresponding XML data type is a named data type that is a restriction of the `complexType` corresponding to the `SELECT` data type, but the content model is the same `group`.

When the actual value is an entity instance whose data type, or some supertype thereof, appears in the working select-list of the fundamental `SELECT` type (see 8.3.4.2), the value shall be encoded as an entity instance element, as specified in 9.3. But the use of by-value encoding (see 9.3.1) or by-reference encoding shall be as specified in 9.4.1 for the attribute whose value is being represented..

NOTE 3 When the inheritance mapping applies to the entity instance, the entity instance element may represent a subtype of the data type that appears in the `group` corresponding to the `SELECT` data type. However, it will be a valid member of the group, by virtue of the substitution group properties of the entity instance elements (see 8.5.5).

 

When the actual value is an instance of a defined data type that appears in the working select-list, the instance element shall be that corresponding to that defined data type, as specified in 8.4.3, and the value shall be encoded in the content of the instance element, according to the fundamental type of the value, as specified in 9.9.

NOTE 4 By the rules of EXPRESS, the actual value cannot be an instance of a specialization of a data type that appears in the working select-list, unless that specialization itself appears in the working select-list.

If the instance element represents an EXPRESS data type that appears directly in the select-list of the fundamental `SELECT` type, the `path` attribute shall not be present in the instance element.

If the instance element does not represent an EXPRESS data type that appears directly in the select-list of the fundamental `SELECT` type, the `path` attribute shall be present in the instance element. The value of the `path` attribute shall be a sequence of the XML names that correspond to the EXPRESS identifiers (see 8.1.2) for all of the generalizations of the data type that are actually instantiated in the value. The XML names shall be separated by whitespace and shall appear in order from outermost `SELECT` type to innermost `SELECT` type.

NOTE 5 The above rule deals with nested `SELECT` data types, that is, situations in which the actual data type is "more than one type declaration removed" from the required `SELECT` data type. In such cases, the `path` attribute of the instance element specifies the "path through the `SELECT` graph" by which the actual data type of that value is reached. Per ISO 10303-11, the meaning of the "path" is that the actual value is an instance of all of the named data types on the path (and possibly others).

EXAMPLE The following example illustrates the representation of an instance of an EXPRESS `SELECT` data type:

In EXPRESS:

```
TYPE Label = STRING;
END_TYPE;

TYPE Length_measure = REAL;
END_TYPE;

TYPE Traffic_light = ENUMERATION OF (red, yellow, green);
END_TYPE;

TYPE Inner = SELECT (Label, Length_measure);
END_TYPE;

ENTITY Pipe;
    Name : STRING;
END_ENTITY;

TYPE S = SELECT (Length_measure, Traffic_light, Pipe, Inner);
END_TYPE;

ENTITY A;
   sn : S;
END_ENTITY;
```

And associated instance data in a Part 21 file:

```
#10 = A(LENGTH_MEASURE(3.14));
#20 = A(LABEL('Hello, world.'));
#30 = A(TRAFFIC_LIGHT(.RED.));
#40 = A(#50);
#50 = PIPE('P1-1');
```

In XML instance data:

```
<A id = "i10" >
  <Sn><Length_measure-wrapper>3.14</Length_measure-wrapper></Sn>
</A>

<A id = "i20" >
  <Sn path="S Inner">
    <Label-wrapper>Hello, world</Label-wrapper>
  </Sn>
</A>

<A id = "i30" >
  <Sn><Traffic_light>red</Traffic_light></Sn>
</A>

<A id = "i40" >
  <Sn>
    <Pipe Name = "P1-1"/>
  </Sn>
</A>
```

In the above, **exp-attribute="attribute-content"** was assumed for entity `Pipe`. That value of **exp-attribute** does not affect the encoding of entity `A`.

## 9.8 Representation of aggregate values

An *aggregate value* is a value of an EXPRESS aggregation data type (`ARRAY`, `BAG`, `LIST` or `SET`) or a value of a defined data type whose fundamental type is an EXPRESS aggregation data type. The *component values* of an aggregate value are the values of the base-type that make up the aggregate value.

The representation of an aggregate value depends on the EXPRESS data types being represented, and on the configuration directives that apply to the occurrence of the value.

The XML representation of an aggregate value in a given occurrence shall be as specified in Table 7 for the combination of aggregate type, base-type, and **tagless** that applies to that occurrence, as follows:

— If no **map** directive applies to the component values (see 10.5.4), the base-type shall be that specified in the EXPRESS declaration for the aggregation data type. If a **map** directive applies to the component values, and the value space of the named XML data type comprises only values that contain no whitespace, the base-type shall be considered to be a *simple* data type, and the default value of **tagless** shall be **true**. If a **map** directive applies to the component values, and the the value space of the named XML data type includes values that contain whitespace, the base-type shall be considered to be a `STRING` data type, and the default value of **tagless** shall be **false**.

— *Simple* types are those whose XML representations have `simpleTypes` and contain no spaces, including EXPRESS types `BOOLEAN`, `INTEGER`, `LOGICAL`, `REAL` and `NUMBER`, named data types that are mapped to certain XML data types by the **map** configuration directive as specified above, and defined data types that are specializations of any of these.

— The value for aggregate-type shall be as specified in the EXPRESS declaration for the aggregation data type. For the purposes of this table, `ARRAY OF OPTIONAL` base-type is considered a distinct aggregate-type.

 

— The value for the directive **tagless** that applies to the aggregate value shall be as specified in 10.5.

— The applicable data type properties and the applicable values of the configuration directives should be ordered like the columns in the table, and compared one-for-one with the values in each row of the table.

— The value of the applicable property or directive named in the column matches the value in that column in a given row only if they are identical. Exception: Any applicable value of a directive matches a blank cell in the row.

— The applicable values will match all the values in exactly one row of the table. The XML encoding of the aggregate value shall be as specified in the subclause identified in the Subclause column of that row.

**Table 7 — Subclause governing XML representation of aggregate value**

| Aggregate type | Base type | tagless | Subclause |
|---|---|---|---|
| ARRAY/LIST | Simple | true | 9.8.1 |
| ARRAY/LIST | Simple | false | 9.8.2 |
| ARRAY/LIST | STRING/BINARY | true | 9.8.1 |
| ARRAY/LIST | STRING/BINARY | false | 9.8.2 |
| ARRAY/LIST | ENUMERATION | true | 9.8.1 |
| ARRAY/LIST | ENUMERATION | false | 9.8.2 |
| ARRAY/LIST | SELECT | | 9.8.2 |
| ARRAY/LIST | Defined data type | | 9.8.6 |
| ARRAY/LIST | Entity data type | true | 9.8.4 |
| ARRAY/LIST | Entity data type | false | 9.8.2 |
| ARRAY/LIST | Aggregation data type | | 9.8.5 |
| ARRAY OF OPTIONAL | Simple | | 9.8.3 |
| ARRAY OF OPTIONAL | STRING/BINARY | | 9.8.3 |
| ARRAY OF OPTIONAL | ENUMERATION | | 9.8.3 |
| ARRAY OF OPTIONAL | SELECT | | 9.8.3 |
| ARRAY OF OPTIONAL | Defined data type | | 9.8.6 |
| ARRAY OF OPTIONAL | Entity data type | | 9.8.3 |
| ARRAY OF OPTIONAL | Aggregation data type | | 9.8.5 |
| SET/BAG | Simple | true | 9.8.1 |
| SET/BAG | Simple | false | 9.8.2 |
| SET/BAG | STRING/BINARY | true | 9.8.1 |
| SET/BAG | STRING/BINARY | false | 9.8.2 |
| SET/BAG | ENUMERATION | true | 9.8.1 |
| SET/BAG | ENUMERATION | false | 9.8.2 |
| SET/BAG | SELECT | | 9.8.2 |
| SET/BAG | Defined data type | | 9.8.6 |
| SET/BAG | Entity data type | true | 9.8.4 |
| SET/BAG | Entity data type | false | 9.8.2 |
| SET/BAG | Aggregation data type | | 9.8.5 |

### 9.8.1 List-of-values representation of aggregate values

If the XML schema data type corresponding to the EXPRESS aggregation data type has the list-of-values form, as specified in 8.2.2, the aggregate value shall be encoded as specified in this subclause.

NOTE 1 If the base-type of the aggregate value is BOOLEAN, INTEGER, LOGICAL, REAL or NUMBER, or a defined data type whose fundamental type is any of these, 8.2.2.3 specifies that the XML schema data type is derived from `xs:list`.

NOTE 2 If the base-type is a defined data type whose fundamental type is an ENUMERATION type, 8.2.2.5 specifies that the XML schema data type is derived from xs:list.

NOTE 3 If the base type is, or is a specialization of, a defined data type that has been declared by the **map** configuration directive (see 10.2.13) to be represented by an XML data type whose values contain no whitespace, 8.2.2.7 specifies that the XML schema data type is derived from xs:list.

NOTE 4 Except as noted in Note 3 above, if the base-type is BINARY, STRING, a SELECT type, an entity data type, or an aggregation data type, this subclause does not apply. But this subclause may apply to the encoding of aggregate values in a nested aggregate. See 9.8.5.

The aggregate value shall be represented (in the content of the accessor element, in the value of the accessor attribute, or in the content of an instance element corresponding to the aggregation data type), by a single XML list value. Each of the component values of the aggregate value shall be represented in the list value according to its data type, as specified in 9.5, or in 9.6 when its data type is an ENUMERATION. The representations of the component values shall be separated by whitespace, as specified in *XML Schema Part 2: Datatypes*.

If the aggregate value has no component values, the XML list value shall be the empty sequence of characters.

For EXPRESS ARRAY values, the component values shall appear in order from lowest index to highest index. For EXPRESS LIST values, the component values shall appear in order of position. For EXPRESS SET or BAG values, the component values may appear in any order, but each component value shall appear exactly as many times as it appears in the aggregate value.

When the component values are values of EXPRESS data type STRING, or a defined data type whose fundamental type is STRING, no component value shall contain whitespace.

When the component values are values of EXPRESS data type BINARY, or a defined data type whose fundamental type is BINARY, each component value shall be an octet-string (that is, the length of each component value shall be a multiple of eight bits).

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

In EXPRESS:

```
TYPE color = ENUMERATION OF (red, blue, green, white, black);
END_TYPE;

ENTITY product_definition;
   package_sizes : BAG OF INTEGER;
   dimensions : ARRAY [1:3] OF REAL;
   options : LIST [1:?] OF BOOLEAN;
   colors : SET [0:?] OF color;
END_ENTITY;
```

In XML instance data with **exp-attribute="attribute-tag"**:

```
<Product_definition id = "id10" >
   <Package_sizes>1 4 12 12</Package_sizes>
   <Dimensions>1.2 2 3.4</Dimensions>
   <Options>true true false</Options>
   <Colors>red blue white black</Colors>
</Product_definition>
```

 

In XML instance data with **exp-attribute="attribute-content"**:

```
<Product_definition id = "id10"
   Package_sizes = "1 4 12 12"
   Dimensions = "1.2 2 3.4"
   Options = "true true false"
   Colors = "red blue white black" />
```

### 9.8.2 Sequence-of-elements representation of aggregate values

If the XML schema data type corresponding to the EXPRESS data type of the aggregate value has the sequence-of-elements form, as specified in 8.2.2, the aggregate value may be encoded (see Table 7) as specified in this subclause.

NOTE For aggregates whose base-type is an aggregation data type or an entity data type, this clause does not apply. See Table 7.

The aggregate value shall be represented in the content of the accessor element, or in the content of an instance element for an outer aggregate element (see 9.8.5), as a sequence of instance elements, each of which corresponds to one component value of the aggregate value. Each component value shall be encoded in the corresponding instance element, as specified in 9.8.7.

The accessor element, or the instance element for the outer aggregate, may, but is not required to, have the attribute `arraySize`. If `arraySize` is specified, the value of `arraySize` shall be the number of component values in the aggregate value.

For EXPRESS `ARRAY` values, the instance elements for the component values shall appear in order from lowest index to highest index. For EXPRESS `LIST` values, the instance elements for the component values shall appear in order of position in the list. For EXPRESS `SET` or `BAG` values, the instance elements for the component values may appear in any order, but each component value shall appear exactly as many times as it appears in the aggregation.

If the aggregate value is has no component values, the sequence of XML elements corresponding to the component values shall be empty.

In each component instance element, the `pos` attribute shall not be present.

In EXPRESS:

```
TYPE partno = STRING;
END_TYPE;

TYPE standard_part=partno;
END_TYPE;

ENTITY part;
  id : partno;
  description : STRING;
END_ENTITY;

TYPE component = SELECT(part, standard_part);
END_TYPE;

ENTITY assembly;
  components : LIST OF component;
END_ENTITY;
```

In XML instance data:

```
<Assembly id = "p1104" >
  <Components>
    <Part ref="p301" xsi:nil="true" />
    <Part ref="p315" xsi:nil="true" />
    <Standard_part>FX-7721-004A</Standard_part>
    <Part ref="p306" xsi:nil="true" />
  </Components>
</Assembly>
```

### 9.8.3 Indexed representation of aggregate values

This representation is selected for a value of an EXPRESS data type that has an EXPRESS aggregate type of ARRAY OF OPTIONAL, as specified in Table 7.

The aggregate value shall be represented in the content of the accessor element, or in the content of an instance element for an outer aggregate element (see 9.8.5), as a sequence of XML elements, each of which corresponds to one component value of the aggregate value. Each component value shall be encoded in the corresponding instance element, as specified in 9.8.7.

The accessor element, or the instance element for the outer aggregate, may, but is not required to, have the attribute arraySize. If arraySize is specified, and the aggregate value is an ARRAY, the value of arraySize shall be the declared size of the array. If arraySize is specified, and the aggregate value is a LIST, the value of arraySize shall be the number of component values in the aggregate value.

NOTE 1 When the aggregate value is a value of an ARRAY, even an ARRAY OF OPTIONAL, data type, the number of values in the aggregate value is always equal to the declared size of the array. But in the case of ARRAY OF OPTIONAL, they may not all be represented in the content of the accessor or outer instance element. See below.

NOTE 2 For aggregates whose base-type is an aggregation data type, this clause does not apply. See Table 8.

For EXPRESS ARRAY values, the instance elements for the component values shall appear in order from lowest index to highest index. For EXPRESS LIST values, the instance elements for the component values shall appear in order of position in the list.

In each component instance element, the pos attribute shall be present. For a component of an ARRAY, the value of pos shall be the array index of the component value. For a component of a LIST, the value of pos shall be the position in the list, beginning with 1.

If the aggregate value has no component values, the sequence of XML elements corresponding to the component values shall be empty.

When the component value for a given position is not available to the pre-processor, the sequence of component instance elements shall contain no instance element corresponding to that position.

NOTE 3 A component value can be missing when the primary aggregation data type is ARRAY and its base-type is declared OPTIONAL. Because the component values must be presented in order, the post-processor can infer from a break in the sequence of index values that one or more components have no value.

EXAMPLE This example illustrates the indexed representation of aggregates of STRING values. Compare with the example in 9.8.2:

In EXPRESS:

 

```
ENTITY product_definition;
  annotations: ARRAY [1:4] OF OPTIONAL STRING;
END_ENTITY;
```

In XML instance data, where annotations[3] does not exist:

```
<Product_definition id = "id10" >
  <Annotations>
    <exp:string-wrapper pos="1">Requires special process
314</exp:string-wrapper>
    <exp:string-wrapper pos="2">Ends to be finished by
hand</exp:string-wrapper>
    <exp:string-wrapper pos="4">Check for conformance of material to
S1106</exp:string-wrapper >
</Product_definition>
```

### 9.8.4 List-of-references representation of aggregate values

If the XML schema data type corresponding to the EXPRESS aggregation data type has the list-of-references form, as specified in 8.2.2, the aggregate value shall be encoded as specified in this subclause.

NOTE 1 This subclause only applies when the base-type of the aggregation type is an entity data type and the configuration directive **tagless="true"** applies to the aggregate value.

The aggregate value shall be represented (in the content of the accessor element, in the value of the accessor attribute, or in the content of an instance element corresponding to the aggregation data type) by a single XML `list` value. Each of the component entity instances in the aggregate value shall be represented in the `list` value by a single value of XML data type `IDREF`. The `IDREF` values shall be separated by whitespace, as specified in *XML Schema Part 2: Datatypes*.

For each of the component entity instances, the unit of serialization shall (somewhere) contain a by-value entity instance element that represents that entity instance; as specified in 9.3.1. The by-value entity instance element shall have a local XML identifier. In the XML `list` value, the `IDREF` value that corresponds to that entity instance shall be that local XML identifier.

NOTE 2 When this representation is used, there is no XML identity constraint (keyref) that guarantees that the `IDREF` values refer to values of the base-type.

For EXPRESS `ARRAY` values, the `IDREF` values corresponding to the entity instances shall appear in order from lowest index to highest index. For EXPRESS `LIST` values, the `IDREF` values shall appear in order of position. For EXPRESS `SET` or `BAG` values, the ordering of the `IDREF` values is arbitrary, but each entity instance shall be represented exactly as many times as it appears in the aggregate value.

If the aggregate value contains no entity instances, the XML `list` value shall be the empty sequence of characters.

EXAMPLE

In EXPRESS:

```
ENTITY Pt3d;
  -- additional attributes skipped
END_ENTITY;

ENTITY curve;
  deflection_points : LIST OF Pt3d;
```

```
END_ENTITY;
```

In XML instance data with **exp-attribute="attribute-tag"**:

```
<Curve id = "id10" >
  <Deflection_points>id001 id002 id003</Deflection_points>
</Curve>

<Pt3d id = "id001" >
  …
</Pt3d>

<Pt3d id = "id002" >
  …
</Pt3d>

<Pt3d id = "id003" >
  …
</Pt3d>
```

In XML instance data with **exp-attribute="attribute-content"**:

```
<Curve id = "id10"  Deflection_points="id001 id002 id003" />
<Pt3d id = "id001"  … />
<Pt3d id = "id002"  … />
<Pt3d id = "id003"  … />
```

### 9.8.5 Aggregates of aggregate values

This subclause specifies the representation of any EXPRESS aggregate value (ARRAY, BAG, LIST or SET) whose component values are also aggregate values.

In this subclause, the aggregate value being represented is called the *primary* aggregate value, and its component values, which are also aggregate values, are called *nested* aggregate values. The primary aggregate value itself may be nested inside a larger aggregate value, called the *outer* aggregate value.

The term *deepest underlying type* is used as defined in 8.2.2.4.2. For this subclause, let the values of the deepest underlying type within the primary aggregate value be designated *deepest values*.

The representation of the primary aggregate value shall be as specified in Table 8, depending on whether any of the nested aggregation data types is OPTIONAL, and on the value of the configuration directive **flatten** that applies, as follows:

— The applicable aggregate type and the applicable values of the configuration directive should be ordered like the columns in Table 8, and compared one-for-one with the values in each row of the table. The applicable value of the property or directive named in the column matches the value in that column in a given row only if they are identical. Exception: Any value of a property or directive matches a blank cell in the row.

— The applicable values will match all the values in exactly one row of the table. The representation of the aggregate value shall be as specified in the subclause identified in the Subclause column of that row.

NOTE There are four choices for this representation: the "sequence-of-rows" structure, as specified in 9.8.5.1, the "indexed-rows" structure, as specified in 9.8.5.2, the "multi-dimensional array" structure, as specified in 9.8.5.3, and the "serialized-array" structure, as specified in 9.8.5.4.

 

**Table 8 — Subclause governing XML representation of aggregates of aggregate values**

| Aggregate type | flatten | Subclause |
|---|---|---|
| ARRAY OF ARRAY | | 9.8.5.4 |
| ARRAY OF ARRAY OF OPTIONAL | | 9.8.5.3 |
| ARRAY OF LIST/SET/BAG | false | 9.8.5.1 |
| ARRAY OF LIST/SET/BAG | true | 9.8.5.4 |
| ARRAY OF OPTIONAL AGGREGATE | false | 9.8.5.2 |
| ARRAY OF OPTIONAL AGGREGATE | true | 9.8.5.3 |
| SET/BAG/LIST OF AGGREGATE | false | 9.8.5.1 |
| SET/BAG/LIST OF AGGREGATE | true | 9.8.5.4 |

Note to Table 8

1. AGGREGATE shall be any of ARRAY, SET, LIST, BAG, or ARRAY OF OPTIONAL.

### 9.8.5.1 Representation as a sequence-of-rows

The sequence-of-rows representation may be used (see Table 8) when the XML schema declarations for the occurrence are as specified in 8.2.2.4.1. When the sequence-of-rows representation is chosen, the primary aggregate value shall be represented as follows:

The primary aggregate value shall be represented (in the content of the accessor element, or in the content of an instance element corresponding to the primary aggregation data type) as a sequence of XML elements, each of which corresponds to one component value of the primary aggregate value, that is, one nested aggregate value.

The accessor element, or the instance element for the primary aggregate, may, but is not required to, have the attribute arraySize. If arraySize is specified, the value of arraySize shall be the number of component values in the aggregate value.

The element corresponding to each component value shall be an instance element that corresponds to the EXPRESS data type of the component value, as specified in 8.4.2, if the base-type of the primary aggregate is anonymous, or 8.4.3, if the base-type of the primary aggregate is named. Each component value shall be encoded in the corresponding instance element as specified in 9.8.7. Each component value is an aggregate value itself, and its by-value encoding shall be as specified in Table 7, using the base-type and the configuration directives that apply to the component.

NOTE 1 That is, in the content of the instance element representing the component value, the nested/component aggregate value becomes the primary aggregate value being encoded, and the current primary aggregate value becomes the outer aggregate value with respect to that component. And the encoding of the nested/component aggregate value depends on its base-type and the other directives that apply to it, as specified in Table 7. The component value itself may be an aggregate of aggregate values and may have a different choice of representation.

When the primary aggregate value is an EXPRESS ARRAY value, the instance elements for the component values shall appear in order from lowest index to highest index. For an EXPRESS LIST value, the instance elements for the component values shall appear in order of position in the list. For an EXPRESS SET value or BAG value, the instance elements for the component values may appear in any order, but each component value shall appear exactly as many times as it appears in the aggregate value.

If the primary aggregate value has no component values, the sequence of XML elements corresponding to the component values shall be empty.

In each component instance element, the pos attribute shall not be present.

When the component value for a given position in the primary aggregate value is not available to the pre-processor, the representation of the primary aggregate value shall contain an instance element in that position. The instance element shall be an instance element that corresponds, as specified in 8.4, to the base-type of the primary aggregation data type. The instance element shall have no content and instead shall have the attribute xsi:nil="true" and no other XML attributes.

NOTE 2 A component value can be missing when the primary aggregation data type is ARRAY OF OPTIONAL.

EXAMPLE Sequence-of-rows representation of a multi-dimensional array of simple types (compare with the examples in the other subclauses of 9.8.5)

In EXPRESS:

```
TYPE Matrix = ARRAY[1:3] OF ARRAY[1:2] OF REAL;
END_TYPE;
```

The value represented using the ISO 10303-21 clear text encoding [1] as:

```
( (1.0,2.0), (0.0, 4.0), (0.0,5.4) )
```

is represented in XML instance data, using the sequence-of-rows representation as:

```
<Tns:Matrix exp:itemType="Tns:Seq-double" exp:arraySize="3">
  <Tns:Seq-double >1.0 2.0</Tns:Seq-double>
  <Tns:Seq-double >0.0 4.0</Tns:Seq-double>
  <Tns:Seq-double >0.0 5.4</Tns:Seq-double>
</Tns:Matrix>
```

In this case, the XML attributes exp:itemType and exp:arraySize have fixed values and need not appear in the XML data, unless required by conventions for the use of the unit of serialization that are outside the scope of this Part of ISO 10303. They are included here for clarity.

### 9.8.5.2 Representation as indexed-rows

The indexed-rows representation may be used (see Table 8) when the XML schema declarations for the occurrence are as specified in 8.2.2.4.1. When the indexed-rows representation is chosen, the primary aggregate value shall be represented as follows:

The primary aggregate value shall be represented in the content of the accessor element, or in the content of an instance corresponding to the primary aggregation data type, as a sequence of XML elements, each of which corresponds to one component value of the aggregate value, that is, one nested aggregate value.

The accessor element, or the instance element for the primary aggregate, may, but is not required to, have the attribute arraySize. If arraySize is specified, and the primary aggregate value is an ARRAY, the value of arraySize shall be the declared size of the array. If arraySize is specified, and the primary aggregate value is a LIST, the value of arraySize shall be the number of component values actually present in the aggregate value.

NOTE 1 When the primary aggregate value is a value of an ARRAY, even an ARRAY OF OPTIONAL, data type, the number of values in the aggregate value is always equal to the declared size of the array. But in the case of ARRAY OF OPTIONAL, they may not all be represented in the content of the accessor or primary instance element. See below.

 

The element corresponding to each component value shall be an instance element that corresponds to the EXPRESS data type of the component value, as specified in 8.4.2, if the base-type of the primary aggregate is anonymous, or 8.4.3, if the base-type of the primary aggregate is named. Each component value shall be encoded in the corresponding instance element as an aggregate value itself as specified in Table 8, using the base-type and the configuration directives that apply to that component.

NOTE 2 That is, in the content of the instance element representing the component value, the nested/component aggregate value becomes the primary aggregate value being encoded, and the current primary aggregate value becomes the outer aggregate value with respect to that component. And the encoding of the nested/component aggregate value depends on its base-type and the other directives that apply to it, as specified in Table 8. The component value itself may be an aggregate of aggregate values and may have a different choice of representation.

When the primary aggregate value is an EXPRESS `ARRAY` value, the instance elements for the component values shall appear in order from lowest index to highest index. For an EXPRESS `LIST` value, the instance elements for the component values shall appear in order of position in the list. For an EXPRESS `SET` value or `BAG` value, the instance elements for the component values may appear in any order, but each component value shall appear exactly as many times as it appears in the aggregate value.

If the primary aggregate value has no component values, the sequence of XML elements corresponding to the component values shall be empty.

In each component instance element, the `pos` attribute shall be present. For a component of an `ARRAY`, the value of `pos` shall be a decimal integer representing the array index of the component value. For a component of a `LIST`, the value of `pos` shall be a decimal integer representing the position in the list, beginning with 1.

When the component value for a given position is not available to the pre-processor, the sequence of component instance elements shall contain no instance element corresponding to that position.

NOTE 3 A component value can be missing when the primary aggregation data type is `ARRAY OF OPTIONAL`. Because the component values must be presented in order, the post-processor can infer from a break in the sequence of index values that one or more components have no value.

EXAMPLE Indexed-rows representation of a multi-dimensional array of simple types (compare with the examples in the other subclauses of 9.8.5)

In EXPRESS:

```
TYPE Matrix = ARRAY[1:3] OF ARRAY[1:2] OF REAL;
END_TYPE;
```

The value represented using the ISO 10303-21 clear text encoding [1] as:

```
( (1.0,2.0), (0.0, 4.0), (0.0,5.4) )
```

is represented in XML instance data, using the sequence-of-rows representation as:

```
<Tns:Matrix exp:itemType="Tns:Seq-double" exp:arraySize="3">
  <Tns:Seq-double pos="1">1.0 2.0</Tns:Seq-double>
  <Tns:Seq-double pos="2">0.0 4.0</Tns:Seq-double>
  <Tns:Seq-double pos="3">0.0 5.4</Tns:Seq-double>
</Tns:Matrix>
```

In this case, the XML attributes `exp:itemType` and `exp:arraySize` have `fixed` values and need not appear in the XML data, unless required by conventions for the use of the unit of serialization that are outside the scope of this Part of ISO 10303. They are included here for clarity.

#### 9.8.5.3 Representation as a multi-dimensional array

The multi-dimensional array representation may be used (see Table 8) when the XML schema declarations for the occurrence are as specified in 8.2.2.4.2. When the multi-dimensional array representation is chosen, the primary aggregate value shall be represented as follows:

The primary aggregate value shall be represented in the content of the accessor element, or in the content of an instance element corresponding to the primary aggregation data type, as a sequence of instance elements, each of which corresponds to one deepest value. Each deepest value shall be encoded in the corresponding instance element, as specified in 9.8.7.

The accessor element, or the instance element for the primary aggregate, shall have the XML attribute `arraySize`. The value of `arraySize` shall be a sequence of decimal integer values, each of which represents the actual size of one dimension of the multi-dimensional array value. The sizes shall appear in order from the outermost (primary) aggregate to the deepest and shall be separated by spaces.

Exception: When any of the nested aggregation data types was declared to be a `LIST`, the value of the corresponding dimension shall be as follows: In any given value of the the primary aggregation data type, all of the component values that correspond to that `LIST` should have the same number of components, and if so, that number shall be used as the value for the corresponding dimension. In the (deprecated) case that the component values that correspond to that `LIST` do *not* all have the same number of components, the maximum number of components among those `LIST` values shall be used for the corresponding dimension in that primary aggregate value.

NOTE 1 When any of the nested aggregation data types is a `LIST`, the dimensions given in the `arraySize` attribute may be different for different instances of the primary aggregation data type. The rule for determining the appropriate value for the variable dimensions is applied separately to each instance of the primary aggregation data type.

NOTE 2 When the primary aggregate value is a value of an `ARRAY`, even an `ARRAY OF OPTIONAL`, data type, the number of values in the aggregate value is always equal to the declared size of the array. But in the case of `ARRAY OF OPTIONAL`, they may not all be represented in the content of the accessor or primary instance element. See below.

NOTE 3 If any of the nested aggregation data types is a `LIST`, and not all of the `LIST` values are of the same size, it is treated as if it were an `ARRAY OF OPTIONAL` with a dimension equal to the maximum size of the `LIST` values. The non-existent elements at the ends of the shorter `LIST` values do not appear.

For EXPRESS `ARRAY` values, the instance elements representing the component values shall appear in order from lowest index to highest index. For EXPRESS `LIST` values, the instance elements representing the component values shall appear in order of position in the list. All of the component values of each nested aggregate value shall appear before any component value of another nested aggregate value.

NOTE 4 That is, each component value is represented by a sequence of instance elements for deepest values; there are no instance elements for any of the nested aggregate values. And the "innermost index", which corresponds to the most nested aggregate, varies fastest; then the next outer index, and so on.

If any nested aggregate value has no component values, the sequence of instance elements corresponding to that nested aggregate value shall be empty. If the primary aggregate value has no

 

component values, the sequence of instance elements corresponding to the primary aggregate value shall be empty.

In each instance element, the `pos` attribute shall be present. The value of the `pos` attribute shall be a sequence of decimal integer values separated by spaces. The first integer value in the sequence shall be the index value corresponding to the component of the primary aggregate value in which this deepest value occurs. Thus, the first integer value designates a nested aggregate value. The next integer value in the sequence shall be the index value corresponding to the component of that nested aggregate value in which this deepest value occurs, and so on. The final value in the sequence shall be the index of the deepest value in the most nested aggregate value. For a component of an `ARRAY`, the integer value shall be the array index of the component value. For a component of a `LIST`, the integer value shall be the position in the list, beginning with 1.

When the component value for a given position does not exist or is not available to the pre-processor, the sequence of deepest instance elements shall contain no instance elements corresponding to that component.

NOTE 5 A component value can be unavailable when the base-type of the primary aggregation data type, or the base-type of any nested `ARRAY` data type, is declared `OPTIONAL`. A component value can be non-existent if it corresponds to a `LIST` position that is beyond the end of the actual `LIST` value. Because the component values must be presented in order, the post-processor can infer from a break in the sequence of index/position values that one or more components have no value.

EXAMPLE Multi-dimensional array representation of a multi-dimensional array of simple types (compare with the examples in the other subclauses of 9.8.5)

In EXPRESS:

```
TYPE Matrix = ARRAY[1:3] OF ARRAY[1:2] OF REAL;
END_TYPE;
```

The value represented using the ISO 10303-21 clear text encoding [1] as:

```
( (1.0,2.0), (0.0, 4.0), (0.0,5.4) )
```

is represented in XML instance data, using the multi-dimensional array representation as:

```
<Tns:Matrix exp:arraySize = "3 2">
   <exp:double pos = "0 0">1.0</exp:double>
   <exp:double pos = "0 1">2.0</exp:double>
   <exp:double pos = "1 0">0.0</exp:double>
   <exp:double pos = "1 1">4.0</exp:double>
   <exp:double pos = "2 0">0.0</exp:double>
   <exp:double pos = "2 1">5.4</exp:double>
</Tns:Matrix>
```

In this case, the XML attributes `exp:itemType` and `exp:arraySize` have `fixed` values and need not appear in the XML data, unless required by conventions for the use of the unit of serialization that are outside the scope of this Part of ISO 10303. They are included here for clarity.

#### 9.8.5.4 Representation as a serialized-array

The serialized-array representation may be used (see Table 8) when the XML schema declarations for the occurrence are as specified in 8.2.2.4.2. When the serialized-array representation is chosen, the primary aggregate value shall be represented as follows:

The primary aggregate value shall be represented in the content of the accessor element, or in the content of an instance element corresponding to the primary aggregation data type, as a sequence of instance elements, each of which corresponds to one deepest value. Each deepest value shall be encoded in the corresponding instance element, as specified in 9.8.7.

The accessor element, or the instance element for the primary aggregate, shall have the XML attribute `arraySize`. The value of `arraySize` shall be a sequence of decimal integer values, each of which represents the actual size of one dimension of the multi-dimensional array value. The sizes shall appear in order from the outermost (primary) aggregate to the deepest and shall be separated by spaces. When any of the nested aggregate datatypes was declared to be a `LIST`, in any given value of the the primary aggregation data type, all of the component values that correspond to that `LIST` shall have the same number of components, and that number shall be used as the value for the corresponding dimension.

NOTE 1 When any of the nested aggregation data types is a `LIST`, the dimensions given in the `arraySize` attribute may still be different for different instances of the primary aggregation data type. The rule above only requires that the variable dimensions be constant in any given instance of the primary aggregation data type.

For EXPRESS `ARRAY` values, the instance elements representing the component values shall appear in order from lowest index to highest index. For EXPRESS `LIST` values, the instance elements representing the component values shall appear in order of position in the list. All of the component values of each nested aggregate value shall appear before any component value of another nested aggregate value.

NOTE 2 That is, each component value is represented by a sequence of instance elements for deepest values; there are no instance elements for any of the nested aggregate values. And the "innermost index", which corresponds to the most nested aggregate, varies fastest; then the next outer index, and so on.

If the primary aggregate value has no component values, the sequence of instance elements corresponding to the primary aggregate value shall be empty.

In the instance element representing each deepest value the `pos` attribute shall not be present.

EXAMPLE Serialized-array representation of a multi-dimensional array of simple types (compare with the examples in the other subclauses of 9.8.5)

In EXPRESS:

```
TYPE Matrix = ARRAY[1:3] OF ARRAY[1:2] OF REAL;
END_TYPE;
```

The value represented using the ISO 10303-21 clear text encoding [1] as:

```
( (1.0,2.0), (0.0, 4.0), (0.0,5.4) )
```

is represented in XML instance data, using the serialized-list values representation as:

```
<Tns:Matrix exp:itemType="double" exp:arraySize="3 2">
1.0 2.0 0.0 4.0 0.0 5.4</Tns:Matrix>
```

In this case, the XML attributes `exp:itemType` and `exp:arraySize` have `fixed` values and need not appear in the XML data, unless required by conventions for the use of the `uos` that are outside the scope of this Part of ISO 10303. They are included here for clarity.

 

### 9.8.6 Aggregates of values of defined data types

A value of any EXPRESS aggregation data type whose base-type is a defined data type shall be represented as specified in Table 7 with the value of ***base-type*** chosen as follows:

— If no **map** directive applies to the component values (see 10.5.4), the ***base-type*** for Table 7, shall be the fundamental type of that defined data type. When the fundamental type of the defined data type is itself an aggregation data type, the overall "aggregate of aggregate" value shall be represented as specified in 9.8.5.

— If a **map** directive applies to the component values, and the values of the named XML data type contain no whitespace, the ***base-type*** for Table 7, or the ***deepest base-type*** for Table 8, shall be considered to be a simple data type.

— If a **map** directive applies to the component values, and the values of the named XML data type may contain whitespace, the ***base-type*** for Table 7, shall be considered to be `STRING`.

### 9.8.7 Instance elements for component values

This clause specifies the encoding of a component value of an aggregate value in an instance element that corresponds to the component value.

When the declared base-type being instantiated by the component value is a defined data type whose fundamental data type is a `SELECT` data type, the component value shall be encoded as specified in 9.7.

Otherwise, the instance element representing the component value shall be an instance element that corresponds to the EXPRESS data type of the component value.

The component value may be represented *by-reference* or *by-value*, according to the configuration directives that apply to the components of the aggregate value, as follows:

— If the configuration directive **content="value"** applies to the components (see 10.5.2), or the configuration directive **content="unspecified"** applies to the components and **exp-type="value"** applies to the data type of the component (see 10.5.3), the representation of the value shall be by-value. If the component value is an entity instance, it shall be encoded as specified in 9.3.1; otherwise, it shall be encoded as specified in 9.10.1.

— If the configuration directive **content="ref"** applies to the components (see 10.5.2), or the configuration directive **content="unspecified"** applies to the components and **exp-type="root"** applies to the data type of the component (see 10.5.3), the representation of the value shall be by-reference. If the component value is an entity instance, it shall be encoded as specified in 9.3.2 or 9.3.3; otherwise, it shall be encoded as specified in 9.10.2.

— If the configuration directive **content="unspecified"** applies to the components and **exp-type="unspecified"** applies to the data type of the components, the representation of the value may be either by-reference or by-value. If the value is an entity instance, the value shall be encoded using one of the representations specified in 9.3, at the discretion of the pre-processor. If the value is not an entity instance, it shall be encoded using one of the representations specified in 9.10, at the discretion of the pre-processor.

NOTE When the choice of by-reference or by-value is unspecified, the pre-processor may use other guidelines and directives to determine which representation to use. In general, the pre-processor should use by-value representation for values that are not entity instances.

## 9.9 Representation of values of defined data types

The representation of a value of a defined data type depends on the fundamental type of that defined data type (see 8.3) as follows:

— A value of an EXPRESS defined data type whose fundamental type is BINARY, BOOLEAN, INTEGER, LOGICAL, NUMBER, REAL, or STRING shall be encoded as specified for the fundamental type in 9.5.

— A value of an EXPRESS defined data type whose fundamental type is an ENUMERATION data type shall be encoded as specified in 9.6.

— A value of an EXPRESS defined data type whose fundamental type is a SELECT data type shall be encoded as specified in 9.7.

— A value of an EXPRESS defined data type whose fundamental type is an aggregation data type (ARRAY, BAG, LIST, or SET) shall be encoded as specified for the fundamental type in 9.8.

NOTE If a **map** configuration directive applies to the value (see 10.5.4), the representation may be affected. The effect of a **map** directive is described in each subclause, where appropriate.

EXAMPLE 1 This example illustrates the representation of a value of a defined data type whose fundamental type is primitive:

In EXPRESS:

```
TYPE Label = STRING;
END_TYPE;

TYPE Length_measure = REAL;
END_TYPE;

TYPE Plane_angle_measure = REAL;
WHERE
 principal_value: 0 <= SELF <= 2*PI;
END_TYPE;

ENTITY line_segment;
   name : OPTIONAL Label;
   origin : Pt2d;
   direction : plane_angle_measure;
   length : length_measure;
END_ENTITY;
```

XML instance data without **exp-attribute="attribute-content"**:

```
<Line_segment id="p106">
   <Name>West face</Name>
   <Origin>
      <Pt2d ref="p109" xsi:nil="true" />
   </Origin>
   <Direction>1.5708</Direction>
   <Length>8.2</Length>
</Line_segment>
```

In XML instance data with **exp-attribute="attribute-content"**:

```
<Line_segment id="p106" Name="West face"
   Direction="1.5708" Length="8.2" >
```

 

```
   <Origin><Pt2d ref="p109" xsi:nil="true" /></Origin>
</Line_segment>
```

EXAMPLE 2 The following example illustrates the representation of a defined data type (temp_vals) whose fundamental type is an aggregate type, and the representation of a defined data type (english_name) as a member of an aggregate value:

In EXPRESS:

```
TYPE english_name= STRING;
END_TYPE;

TYPE f_temperature= REAL;
END_TYPE;

TYPE temp_vals = SET OF f_temperature;
END_TYPE;

TYPE plant_list = LIST OF english_name;
END_TYPE;

ENTITY garden_info;
   temp_meas : temp_vals;
   plants : plant_list;
END_ENTITY;
```

XML instance data without **exp-attribute="attribute-content"**:

```
<Garden_info id = "id10" >
 <Temp_meas>82.7 76.6 79.8</Temp_meas>
 <Plants>
  <English_name>rose</English_name>
  <English_name>tulip</English_name>
  <English_name>lily</English_name>
 </Plants>
</Garden_info>
```

In XML instance data with **exp-attribute="attribute-content"**:

```
<Garden_info id = "id10" Temp_meas = "82.7 76.6 79.8">
 <Plants>
  <English_name>rose</English_name>
  <English_name>tulip</English_name>
  <English_name>lily</English_name>
 </Plants>
</Garden_info>
```

## 9.10 Representation of values in instance elements

As required by the subclauses of Clause 9 above, values may be represented in instance elements. This subclause specifies the details of such representations.

This subclause does not apply to the representation of values that are entity instances. Instance elements representing entity instances shall be as specified in 9.3.

### 9.10.1 By-value instance elements for non-entity data types

The instance element shall be of the type that corresponds, as specified in 8.4, to the actual data type of the value. In particular, when the value represents an instance of a SELECT data type, the instance element shall correspond to the appropriate data type in the working select list, as specified in 9.7.

NOTE The type of the instance element may not correspond to the declared data type of the EXPRESS attribute or the EXPRESS aggregation component that contains it. It could correspond to a subtype or a member of the working select list.

The XML attribute ref shall not be present in the instance element.

If the instance element is an immediate child of the unit of serialization element, the instance element shall have a local XML identifier; the XML attribute id shall be present in the instance element, and its value shall be that local XML identifier. Otherwise, the XML attribute id shall not be present in the instance element.

The value shall be encoded in the content of the instance element, according to its EXPRESS data type, as follows:

— If the data type is a primitive data type (BINARY, BOOLEAN, INTEGER, LOGICAL, NUMBER, REAL or STRING), the instance element shall be that specified in 8.4.1, and the value shall be encoded as specified in 9.5.

— If the data type is an anonymous aggregation data type, the instance element shall be that specified in 8.4.2, and the value shall be encoded as specified in 9.8.

— If the data type is a defined data type whose fundamental type is BINARY, BOOLEAN, INTEGER, LOGICAL, NUMBER, REAL, or STRING, the instance element shall be that specified in 8.4.3, and the value shall be encoded in the content of that element as specified for its fundamental type in 9.5.

— If the data type is a defined data type whose fundamental type is an aggregation data type, the instance element shall be that specified in 8.4.3, and the aggregate value shall be encoded in its content as specified in 9.8.

— If the data type is a defined data type whose fundamental type is an ENUMERATION type, the instance element shall be that specified in 8.4.3, and the value (enumeration item) shall be encoded in its content as specified in 9.6.

— If the data type is nominally a defined data type whose fundamental type is a SELECT type, the value shall be encoded as specified in 9.7.

— If the data type is an entity data type, the value shall be encoded as specified in 9.3.1.

EXAMPLE A by-value instance element corresponding to the EXPRESS data type:

```
TYPE Matrix = ARRAY[1:3] OF ARRAY[1:2] OF REAL;
END_TYPE;
```

Might be:

```
<Tns:Matrix id="v107">
   <Tns:Seq-double>1.0 2.0</Tns:Seq-double>
   <Tns:Seq-double>0.0 4.0</Tns:Seq-double>
   <Tns:Seq-double>0.0 5.4</Tns:Seq-double>
</Tns:Matrix>
```

 

And the `Tns:Seq-double` elements are by-value instance elements representing values of the EXPRESS data type `ARRAY [1:2] OF REAL`. The `Tns:Matrix` element has a local XML identifier, assuming that it appears as an immediate child of the unit of serialization. The `Tns:Seq-double` elements do not have local XML identifiers, because they appear as children of the `Tns:Matrix` element.

### 9.10.2 By-reference instance elements for non-entity data types

The value itself shall be encoded in a by-value instance element, as specified in 9.10.1 above. The by-value instance element shall be an immediate child of the unit of serialization element and shall have a local XML identifier.

NOTE If a by-value instance element representing this value already exists, it is not necessary to create a new one. There can be more than one reference to the same by-value instance element.

The by-reference instance element shall be of the same type as the by-value instance element.

The by-reference instance element shall be empty (have no content) and shall have the XML attribute `xsi:nil="true"`.

The by-reference instance element shall have the XML attribute `ref`, and its value shall be the local XML identifier for the by-value instance element.

The by-reference instance element shall not have the XML attribute `id`.

When required by usage, the by-reference instance element may have the XML attributes `pos` or `path`.

The by-reference instance element shall have no other XML attributes. Any XML attributes that describe the value itself shall appear in the by-value instance element.

EXAMPLE For the value of EXPRESS data type Matrix given in 9.10.1 above, a by-reference instance element would be:

```
<Tns:Matrix ref="v107" xsi:nil="true"/>
```

# 10 Configuration Language

Clause 7.2.2.2 specifies the mapping from an EXPRESS schema to an XML schema. Clause 9 specifies the rules for representing the data described by an EXPRESS schema as XML elements. These mappings and representation rules may be altered and further controlled by the configuration language defined in this clause. The configuration language specifies how EXPRESS schema elements are to be represented in the derived XML schema. The configuration language also specifies how EXPRESS data instances are to be rendered into XML elements.

The configuration language consists conceptually of statements called *directives*. Each directive specifies a particular option for mapping an EXPRESS element to XML Schema components, or mapping data instances described by the EXPRESS elements to XML elements, or both.

A collection of configuration directives that is to be used as a group is referred to as a *configuration file*.

This clause specifies the form of a configuration file and the form and meaning of the directives that can appear in a configuration file.

Many configuration directives can be applied to specific data types, entities and attributes that appear in a given EXPRESS schema. Other configuration directives can apply to the whole EXPRESS schema, or to data types defined in the EXPRESS language. A configuration file that contains only directives that specify global options and mapping rules for data types defined in the EXPRESS language proper is said to be *schema-independent*. A configuration file that contains directives that apply to specific data types and attributes defined in a given EXPRESS schema is said to be *schema-specific*.

Syntactically, the configuration language is XML-based: Directives are represented by XML elements and XML attributes. A configuration file is represented as a `configuration` element.

The form of the `configuration` element is specified in 10.1. The form and meaning of the each directive is specified in 10.2.

Each configuration directive is contained in a *scoping element* in the content of the `configuration` element. The scoping elements are specified in 10.3. In general, the scoping element specifies the scope of application of the directive, and the directive itself is specified by an XML attribute. Configuration options too complex to be represented as simple XML attribute values are represented by nested elements.

EXAMPLE The following is an example showing the structure of a configuration file:

```
<iso_10303_28 version= "2.0">
   <express id="exp01" name="Car_ownership">
     <[CDATA[

SCHEMA car_ownership;
TYPE currency=REAL;
END_TYPE;
ENTITY person;
  name : STRING;
  owns : SET [0:?] OF car;
END_ENTITY;
ENTITY car;
  make : STRING;
  car_model : STRING;
  price : currency;
END_ENTITY;
END_SCHEMA;

     ]]>
   </express>

   <configuration id="conf1" schema="exp01">
     <option contained="true" />

     <entity select="person" >
       <attribute select="owns" exp-attribute="type-tag" />
     </entity>

     <type select="currency" map="xs:decimal" />

   </configuration>
</iso_10303_28>
```

NOTE In general, examples of the configuration directives are found in Clauses 7.2.2.2 and 9 and in Annex G.

 

## 10.1 The `configuration` element

A configuration file is represented in the `iso_10303_28` element by a `configuration` element. The element shall contain one configuration file, either "by-value" or "by-reference". A configuration element can include other configuration files, possibly contained in other documents, by reference, using the **include** element.

The `configuration` element type declaration is:

```
<xs:element name="configuration" nillable="true">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="include">
               <xs:complexType>
                   <xs:attribute name="configuration" type="xs:IDREF"
                          use="required"/>
               </xs:complexType>
          </xs:element>
      <xs:element ref="cnf:option"/>
      <xs:element ref="cnf:type"/>
      <xs:element ref="cnf:entity"/>
      <xs:element ref="cnf:schema" maxOccurs="1"/>
      <xs:element ref="cnf:uosElement"/>
      <xs:element ref="cnf:uosEntity"/>
      <xs:element ref="cnf:rootEntity"/>
    </xs:choice>
    <xs:attribute name="id"  type="xs:ID" use="required"/>
    <xs:attribute name="targetNamespace"  type="xs:anyURI"
      use="optional"/>
    <xs:attribute name="schema"  type="xs:IDREF"  use="optional"/>
    <xs:attribute name="configuration-location"
       type="cnf:Seq-anyURI"  use="optional"/>
  </xs:complexType>
</xs:element>
```

The value of the `id` XML attribute shall be a local XML identifier for the configuration file. This value provides the reference to the configuration file for unit of serialization elements within the document, and part of the identifier for references to the configuration file from other documents.

The value of the schema XML attribute shall be the local XML identifier for the EXPRESS schema the configuration file refers to, as specified in 5.4. The requirements for use of the `schema` attribute are specified below and in 5.6. The interpretation of the `schema` attribute shall be as specified in 10.1.1 and 10.1.2.

The value of the `targetNamespace` XML attribute shall be the URN for the target namespace derived XML schema (see 8.7).

The `include` element incorporates all the configuration directives from another configuration file into the current configuration file. The value of the `configuration` attribute shall be the local XML identifier of another `configuration` element in the current `iso_10303_28` element (see 10.1). The referenced `configuration` element may be by-value or by-reference. The `schema` attribute, if any, of the referenced `configuration` element shall be ignored.

EXAMPLE

```
<configuration id="conf1" configuration-location="URI:www.tc184-
sc4.org/configurations/Part1278"/>
<configuration id="conf2" schema="exp233" >
```

```
  <include configuration="conf1"/>
..<option contained="false" />
</configuration>
```

### 10.1.1 By-reference representation of a configuration file

When the configuration file is represented *by-reference*, the `configuration-location` XML attribute shall be present and its value shall identify one or more resources that contain the configuration file by-value (see 10.1.2). A URI in the value shall be either a URI having no local part or have the form: ***uri*#*id***, where ***uri*** is a URI conforming to IETF RFC 2396 that locates the resource, and the fragment identifier ***id*** is an XML identifier local to that resource that identifies a by-value `configuration` element.

NOTE 1 By convention, the value of `configuration-location` is either a single URI, or a list of URIs of which the first is a URN that identifies the configuration file globally, and all of the others are possible locations of the text.

When the configuration file is represented by-reference, the content of the `configuration` element shall be empty.

When the configuration file is represented by-reference, the `schema` attribute is optional. If it is not present, the value of the `schema` attribute in the by-value `configuration` element, if any, is used. If the `schema` attribute is present, its value overrides the value, if any, of the `schema` attribute in the by-value `configuration` element. If neither the by-reference configuration element nor the by-value configuration element specifies the associated EXPRESS schema, every unit of serialization that uses this configuration shall identify the EXPRESS schema for that unit of serialization (see 9.2.6.1).

NOTE 2 The by-value `configuration` element could, for example, configure a STEP module schema, while the by-reference `configuration` element configures a larger EXPRESS schema that interfaces that module. In this case, both `configuration` elements would have `schema` attributes, but the value for the by-reference `configuration` element (for the interfacing EXPRESS schema) would override the value for the by-value `configuration` element (for the module schema).

### 10.1.2 By-value representation of a configuration file

When the configuration file is represented *by-value*, the content of the `configuration` element shall be non-empty, and the contained XML elements shall conform to the specifications of 10.3.

When the configuration file is represented by-value, the `configuration-location` XML attribute shall not be present.

The `schema` attribute shall be present in all by-value `configuration` elements, except those representing configuration files that are schema-independent, as defined above.

## 10.2 Configuration options

This subclause defines the options that may be specified in the configuration file. These options may appear as attributes of one or more of the **entity**, **type**, **attribute**, **inverse**, **aggregate** or **option** scoping elements (see 10.3).

Configuration options may conflict with each other. In the event that such conflicts occur, the declaration effecting the most specific declaration shall have precedence. **attribute** declarations have the highest priority; followed by **type** declarations, followed by **entity** declarations, followed by entity-level options for the supertypes, followed by global **option**s, which have the lowest precedence.

Exception: When an attribute directive conflicts with a type directive for the data type of the EXPRESS attribute, and that data type is *named* (not anonymous) in the EXPRESS schema, the type directive overrides. All non-conflicting options do have an effect.

NOTE When the type directive affects the XML Schema declaration for the data type, the accessor cannot be declared to have that data type and alter it.

For each EXPRESS entity, type and attribute, the applicable set of declarations in the configuration file shall be applied to the object. When two conflicting options are at the same level of precedence, the resulting configuration is not defined by this part of ISO 10303. Similarly, when conflicting directives for the same EXPRESS attribute are inherited in a subtype, the resulting configuration is not defined by this part of ISO 10303.

### 10.2.1 name

The **name** configuration option specifies the name for the XML element or attribute that corresponds to an EXPRESS data type or attribute. The name specified for each data type shall be unique within the namespace for the derived XML schema, and the name of each attribute must be unique within the owning entity. If the **name** option is not specified, the XML element or attribute name shall be derived from the EXPRESS definition name (see 8.1.2). The **name** attribute has no default.

The value specified by the **name** configuration option is used verbatim and overrides any specified **naming-convention**.

### 10.2.2 exp-type

The **exp-type** configuration option applies to EXPRESS named data types and EXPRESS entity data types. This option determines whether an EXPRESS data type is represented as a root instance element or as a contained element. The option shall have one the following values: **root**, **value**, or **unspecified**. The default value is **unspecified.**

— If the value is **root**, every instance of the entity data type shall be represented by an independent entity instance. Every EXPRESS attribute whose value is an instance of that named data type shall be represented by-reference, and not by-value. This option is overriden by the following attribute specific configuration options: **exp-attribute="no-tag"**, **content="value"**.

— If the value is **value**, the named data type is represented by-value in all cases. This is overriden by the attribute specific configuration option **content="ref"**.

— If the value is **unspecified**, the choice of representation of the named data type is determined by other options, or left to the preprocessor.

### 10.2.3 content

The **content** configuration option determines whether the value of an EXPRESS attribute whose data type is an entity is represented by-reference or by-value. The option must have one the following values: **value**, **ref**, or **unspecified**. The default value is **unspecified.**

— If the value is **value**, the attribute shall be represented by-value. This option conflicts with the **exp-attribute="attribute-content"** and **exp-type="root"** (of the entity referenced) options.

— If the value is **ref**, the attribute shall be represented by-reference. This option conflicts with the **exp-attribute="no-tag"**and **exp-type="value"** options.

— If the value is **unspecified**, the choice of representation of the attribute value is determined by other options, or left to the preprocessor.

NOTE It may not be possible to honor the contained option when processing circular data structures, so the resulting XML schema allows an attribute that is declared **content="value"** to contain a reference.

### 10.2.4 aggregate-content

The **aggregate-content** option determines whether the base element values of a (possibly nested) aggregate value are represented by reference or value. The option must have one the following values: **value**, **ref**, or **unspecified**. The default value is **unspecified.** This option is equivalent to a set of nested **aggregate** configuration elements where:

— The number of nested **aggregate** elements is equal to the number of dimensions the XML representation of the aggregate.

— The innermost element has a **content** configuration attribute with the same value as the value of the **aggregate-content** option.

EXAMPLE In an **entity** configuration element, and for an entity that contains an attribute named `agg` that is declared as a one-dimensional EXPRESS aggregate, the declaration:

```
<attribute select="agg" aggregate-content="ref" />
```

is exactly equivalent to the configuration:

```
<attribute select="agg" >
  <aggregate content="ref" />
</attribute>
```

### 10.2.5 exp-attribute

The **exp-attribute** configuration option determines how an EXPRESS attribute is represented in XML. The option shall have one the following values: **double-tag**, **attribute-tag**, **type-tag, no-tag, no-tag-simple, unspecified**, or **attribute-content**. The default value is **attribute-tag**, unless the datatype of the attribute is an entity datatype, in which case the default is **double-tag.**

— If the value is **double-tag**, the attribute, if it is an entity type, is represented by a pair of nested XML elements: one for the attribute, and one for the entity that populates it.

— If the value is **attribute-tag**, the attribute, if it is an entity type, is represented by a single XML element representing the attribute. If the data type of the EXPRESS attribute has subtypes, the element shall contain an xsi:type element that identified the type of the value.

— If the value is **type-tag**, the attribute, if it is an entity type, is represented by a single XML element representing the entity. It is an error to specify this option if the data type of the EXPRESS attribute has subtypes, or if there is more than one attribute in the entity with the same data type, or if the attribute is an anonymous aggregate and the inheritance-free mapping applies.

— If the value is **no-tag**, the attribute is replaced by the set of attributes of the entity that is its value. It is an error to specify this option if the value is polymorphic, or if the resulting element has an attribute name that conflicts with more than one value. This option conflicts with **exp-type="root"**, and **content="ref"**.

— If the value is **no-tag-simple**, the attribute is represented as an XML text node in the complexType of the owning entity. It can occur only once per configuration directive <entity>. All other attributes of this EXPRESS entity may be set to **exp-attribute="attribute-content"** only.

Example

```
<Translation xml:lang="de-DE" >Luftsack</Translation>
```

 

— If the value is **attribute-content**, the attribute is represented by an XML attribute. This option conflicts with **exp-type="value"** and **content="value"**.

— If the value is **unspecified**, this option has no effect, and other options are used to determine how an attribute is represented.

### 10.2.6 entity-attribute

The **entity-attribute** configuration option determines the XML representation of EXPRESS attributes that are entity data types. **entity-attribute** can only be specified on the **option** configuration element. The option shall have one the following values: **double-tag**, **attribute-tag**, or **attribute-content**. The default value is **double-tag.** Specifying **entity-attribute** provides the equivalent of specifying **exp-attribute** for all attributes that are entity data types, and can be overriden by the **exp-attribute** directive.

### 10.2.7 concrete-attribute

The **concrete-attribute** configuration option determines the XML representation of EXPRESS attributes that are non-entity data types. **concrete-attribute** can only be specified on the **option** configuration element. The option shall have one the following values: **double-tag**, **attribute-tag**, or **attribute-content**. The default value is **attribute-tag.** Specifying **concrete-attribute** provides the equivalent of specifying **exp-attribute** for all attributes that are non-entity data types, and can be overriden by the **exp-attribute** directive.

### 10.2.8 tagless

The **tagless** option is used to specify that aggregate elements are, where possible, to be represented without tags for the individual elements. This option may have the values **true, false,** or **unspecifed**. The default value is **unspecifed.**

When the value of **tagless** is **true**, the aggregate values shall be represented by the `xs:list` form (whitespace delimited list of elements).

When the value of **tagless** is **false**, the aggregate values shall be represented by the list of instance elements form (each aggregate element mapped to an XML element).

When the value is **unspecifed**, it is considered to be **true** for the base-types `BOOLEAN`, `INTEGER`, `LOGICAL`, `REAL`, `NUMBER`, `ENUMERATION`, and for defined data types that are specializations of these, and **false** for all others.

If this option is specified on an **attribute** configuration element, the data type of the attribute must be either an anonymous aggregate or a defined type whose fundamental type is an aggregate. In this case, this option specifies the representation of that attribute in all of the entity instances.

If this option is specified on an **aggregate** configuration element, it determines the representation of the values of that aggregate (which may be nested in another aggregate value).

If this option is specified on a **type** configuration element, the type must be either an anonymous aggregate or a defined type whose fundamental type is an aggregate. In this case, this option specifies a default value for the given type of aggregate regardless of where it appears.

**Tagless = "true"** can be explicitly specified for attributes whose data type is an aggregation of `STRING`, `BINARY`, or an entity data type, or a defined data type whose fundamental type is `STRING` or `BINARY`.

When the configuration directive **tagless="true"** is explicitly specified for an aggregation data type whose base-type is (or is derived from) `STRING`, the actual values of the members of the aggregate value shall contain no whitespace.

When the configuration directive **tagless="true"** is explicitly specified for an aggregation data type whose base-type is, or is derived from, `BINARY`, it is not possible to specify the `extra_bits` XML attribute (see 8.2.1.1), and the actual values of the members of the aggregate value shall be octet-strings (that is, their lengths shall be exact multiples of 8).

When the configuration directive **tagless="true"** is explicitly specified for an aggregation data type whose base-type is an entity data type, the actual values of the members of the aggregate value shall be entity references.

For an aggregation data type that is `ARRAY OF OPTIONAL`, the value of **tagless** shall be (overridden to be) false for all uses of that type.

### 10.2.9 flatten

The **flatten** option is used to represent nested aggregate values as a one-dimensional collection. The value shall be one of **true** or **false**. The default value is **true.**

If the value is **false**, each nested aggregate value is represented as a separate XML value.

If the value is **true**, all of the nested aggregate values are serialized into a single aggregate value – the content of a single XML element – by the serialization rules specified in 9.8.5. That is, **flatten="true"** shall be interpreted as applying to the target (usually outermost) aggregation data type and to every anonymous aggregation data type nested directly within it.

If this option is specified on an **attribute** configuration element, the data type of the attribute must be either an anonymous aggregation data type or a defined type whose fundamental type is an aggregation data type. In this case, this option specifies the representation of that attribute in all of the entity instances.

If this option is specified on an **aggregate** configuration element, it determines the representation of the values of that aggregation data type and those nested within it (which may all be nested in another aggregate value).

If this option is specified on a **type** configuration element, the type must be either an anonymous aggregation data type or a defined type whose fundamental type is an aggregation data type. In this case, this option specifies the value of **flatten** for all uses of the specified data type.

Whenever the value of **flatten** is **true**, the affected aggregate structures should be "rectangular", that is, the number of component values in each dimension should be consistent over the entire aggregate value.

### 10.2.10 use-id

The **use-id** option indicates whether the by-value representation of an EXPRESS attribute whose data type is an entity data type, or an aggregation data type whose base-type is an entity data type, shall have an `id` attribute. The value shall be **true** or **false.** The default value is **false.**

 

**Use-id** shall only occur in an **attribute** element or an **aggregate** element in the content of an **attribute** element. It applies only to entity instances that are, or are contained in, values of that specific attribute.

When **use-id= "true"**, the EXPRESS entity instances that are values of the EXPRESS attribute shall always be represented by-value (see 9.3.1), and the corresponding by-value instance elements shall include an **id** attribute, enabling the entity instance elements to be referenced. If **use-id="true"** applies to an EXPRESS attribute, one of **exp-attribute="double-tag"** and **exp-attribute="type-tag"** shall also apply to that attribute. The entity instance that is the value of the attribute shall not be representated by value more than one time in the unit of serialization.

When **use-id= "false"**, the representation of the value of the EXPRESS attribute shall not include an **id** XML attribute.

NOTE An EXPRESS attribute for which **use-id = "true"** could be interpreted as an alternate location in which root elements can occur.

### 10.2.11 keep

The **keep** option controls the mapping of EXPRESS `ENTITY`, `TYPE`, `ATTRIBUTE`, `INVERSE` or `DERIVE`d attributes to the derived XML schema. The value shall be **true** or **false.** The default value for `ENTITY`, `TYPE`, and `ATTRIBUTE` attributes is true. The default value for `INVERSE` and `DERIVE`d attributes is false**.**

**Keep** shall occur on an **entity, type, attribute or inverse** element. It may also apply to EXPRESS attributes that are inverted by an **invert** directive.

When **keep= "false"** applies to an EXPRESS `ENTITY`, the entity and all of its attributes are excluded from the derived XML schema.

When **keep= "false"** applies to an EXPRESS `TYPE`, the type is excluded from the derived XML schema.

When **keep= "false"** applies to an EXPRESS `ATTRIBUTE`, the attribute is excluded from the derived XML schema and cannot be used in any inverse configuration (see 10.3.5).

When **keep= "false"** applies to an EXPRESS `INVERSE` or `DERIVE`d attribute, the element is excluded from the derived XML schema.

When **keep= "true" applies**, the EXPRESS attribute shall be mapped to an accessor element or attribute in the derived XML schema.

NOTE **keep= "false"** applied to an EXPRESS `ENTITY`, `TYPE` or `ATTRIBUTE` was introduced to support the mapping of lower conformance classes within existing APs.

### 10.2.12 keep-all

The **keep-all** directive provides a method for globally assigning the '**keep**' directive to all EXPRESS `INVERSE` or `DERIVE`d attributes. The values may be **derive**, **inverse**, **derive inverse**, or the value may be unset. The default value is unset.

When the value is **derive**, the **keep= "true"** directive shall apply to all EXPRESS `DERIVE`d attributes.

When the value is **inverse**, the **keep= "true"** directive shall apply to all EXPRESS INVERSE attributes.

When the value is **derive-inverse**, the **keep= "true"** directive shall apply to all EXPRESS INVERSE or DERIVEd attributes.When the value is unset, the **keep= "true"** directive shall not apply to any EXPRESS INVERSE or DERIVEd attributes.

A **keep** directive specified explicitly for an EXPRESS attribute shall override any directive applied by the **keep-all** directive.

### 10.2.13 map

The **map** directive declares the XML data type that shall be used to represent a given EXPRESS data type. The possible values of the **map** directive depend on the EXPRESS data type, and are specified below. The **map** attribute has no default.

This option may also be used on an **entity** element that configures an EXPRESS ENTITY. In this case, the option specifies the complex type from which the current complexType is derived by extension. The possible values of the **map** may be any complexType of the generated XML schema.

This option may be used on a **type** element that configures an EXPRESS BINARY, INTEGER, NUMBER, REAL, or STRING data type, or a defined data type whose fundamental type is one of these. In this case, the option specifies the representation for all values of that EXPRESS data type wherever they occur. The **map** option shall not be used for any other EXPRESS data type.

This option may be used on an **attribute** element that configures an EXPRESS attribute whose data type is any of BINARY, INTEGER, NUMBER, REAL, or STRING, or a defined data type whose fundamental type is one of these. In this case, the option specifies the representation for all values of that EXPRESS attribute.

This option may be used on an **inverse** element that configures an EXPRESS attribute whose data type is any of BINARY, INTEGER, NUMBER, REAL, or STRING, or a defined data type whose fundamental type is one of these. In this case, the option specifies the representation for all values of that EXPRESS attribute.

#### 10.2.13.1 Mapped types for EXPRESS BINARY

For any EXPRESS data type whose fundamental type is BINARY, the following XML Schema datatypes are permitted as values of the **map** directive:

— xs:hexBinary;

— xs:base64Binary.

#### 10.2.13.2 Mapped types for EXPRESS INTEGER

For any EXPRESS data type whose fundamental type is INTEGER, the following XML Schema datatypes are permitted as values of the **map** directive:

— xs:integer;

— xs:decimal;

— xs:long;

 

— xs:int;

— xs:short;

— xs:byte;

— xs:unsignedLong;

— xs:nonNegativeInteger;

— xs:nonPositiveInteger;

— xs:positiveInteger;

— xs:negativeInteger;

— xs:year.

NOTE Except for year, this assignment affects the value space, but not the encoding. Choosing year also affects the encoding.

#### 10.2.13.3 Mapped types for EXPRESS NUMBER or REAL

For any EXPRESS data type whose fundamental type is NUMBER or REAL, the following XML Schema datatypes are permitted as values of the **map** directive:

— xs:double;

— xs:decimal;

— xs:float.

NOTE This assignment affects the value space and the encoding.

EXAMPLE The following example causes the EXPRESS REAL type to be represented by an xs:decimal value, and the positive_length_measure type (a defined type, whose ultimate underlying type is REAL) by an xs:double XML type:

```
<type select="real" map="xs:decimal"/>
<type select="positive_length_measure" map="xs:double"/>
```

#### 10.2.13.4 Mapped types for EXPRESS STRING

For an EXPRESS data type whose fundamental type is STRING, the following XML Schema datatypes are permissible as values of the **map** directive:

— xs:string;

— xs:language;

— xs:Name;

— xs:QName;

— xs:NMTOKEN;

— `xs:anyURI`.

NOTE This assignment affects the value space, but not the encoding of the value. It may also affect the encoding of `ARRAY`/`BAG`/`LIST`/`SET` OF the data type.

### 10.2.14 naming-convention

The **naming-convention** attribute declares the default naming convention to be used to represent an EXPRESS identifier. This option may have the values **initial-upper**, **camel-case** or **preserve-case**. The default value is **initial-upper.**

This option may only be specified globally on the **option** configuration element.

When the **naming-convention** option has the value **initial-upper**, the naming convention shall be: the first letter is capitalized, and the rest of the identifier shall be lowercase.

When the **naming-convention** option has the value **camel-case**, the naming convention shall be: the first letter is capitalized, the first character following an underscore is capitalized, all other characters in the identifier shall be lowercase, and underscores shall be deleted from the XML name.

When the **naming-convention** option has the value **preserve-case**, the XML name corresponding to an EXPRESS identifier shall be identical in letter case to the representation of the identifier in the text of the EXPRESS schema. A conforming pre-processor is not required to support this value for **naming convention**.

NOTE The **preserve-case** option requires that an EXPRESS processing tool must preserve the case of the identifiers. Conformance with ISO-10303-11 (EXPRESS language) does not specify this behaviour.

### 10.2.15 inheritance

The **inheritance** attribute controls the use of XML "inheritance" in the derived XML schema. The value is **true** or **false.** The default value is **false.**

**Inheritance** can only occur on an **option** or **entity** element. By setting **inheritance** at the **option** element, the global behavior is defined. This can be overridden by entries at the **entity** element. For details about when the inheritance mapping is allowed, see 8.5.

When the value of the **inheritance** attribute is **true**, XML schema inheritance shall be used where possible in the generated XML schema.

When the value of the **inheritance** attribute is **false**, the inheritance-free mechanism shall be used in generating the XML schema.

### 10.2.16 notation

The **notation** attribute specifies that the selected data type or attribute shall have the `xml:notation` XML attribute with the value given for this directive. The value of the notation attribute shall be a MIME type, as specified by *Extensible Markup Language (XML) 1.0*, that is, it shall identify an encoding registered in the IANA MIME type registry.

### 10.2.17 tag-source and tag-value

The **tag-source** attribute specifies that the name of the XML element representing each instance of the configured entity data type shall be chosen based on the value of an EXPRESS attribute of that entity instance. The value of **tag-source** shall be the name of one of the following:

 

— an EXPRESS attribute that is declared in the entity declaration for the owning entity data type;

— an EXPRESS attribute that is inherited by the owning entity data type;

— an EXPRESS attribute that is associated with the owning entity data type by an **inverse** configuration directive;

— an EXPRESS attribute that is associated with the owning entity data type by an entity-valued attribute to which the **exp-attribute="no-tag"** directive applies (see 10.5.1.5).

The **tag-values** attribute shall specify all possible values of the EXPRESS attribute designated by the value of **tag-source**.

NOTE The use of the **exp-attribute="type-tag"** and **inverse** option can be used to reference non-local EXPRESS attributes.

EXAMPLE

The following configuration

```
<entity select="product" >
  <inverse invert="cc_design_person_and_organization_assignment.items"
           exp-attribute="type-tag">
</entity>

<entity select="cc_design_person_and_organization_assignment"
  tag_source="role_label"
  tag_values="design_supplier configuration_manager contractor
      classification_officer">
    <attribute  select="role" name="role_label" exp-attribute="type-
tag"/>
</entity>

<entity select="person_and_organization" exp-attribute="type-tag"/>
```

can result in the following instance data.

```
<product id="i34" >
  {product atts}
  <design_supplier>
    <person>
       {person atts}
    <person>
    <organization>
       {org atts}
    </organization>
  </design_supplier>
</product>
```

### 10.2.18 namespace

The **namespace** option is used to specify the xmlns namespace declaration to be used for a specified dataset. If the **namespace** option is not specified, the namespace shall be determined by the pre-processor as specified in 8.7. The value of **alias** shall be a namespace URI. The value of **prefix** shall be a prefix representing the namespace URI. The **alias** and **prefix** attributes have no default.

### 10.2.19 ref

The **ref** option controls the mapping of an EXPRESS attribute to the derived XML schema. The value of the **ref** option shall be a qualified name that refers to an XML attribute or element defined in one of the associated namespaces (see 8.7).

For the EXPRESS attribute to which the **ref** option applies, the mapping specified in 8.6 shall not be used. If **exp-attribute="attribute-content"** applies to the EXPRESS attribute, the corresponding XML schema declaration shall have the form:

```
<xs:attribute ref="xxx" use="usage" />
```

Otherwise, the corresponding XML schema declaration shall have the form:

```
<xs:element ref="xxx" minOccurs="zero-or-one" />
```

where, in either case:

— ***xxx*** is the value of the **ref** option;

— ***usage*** is "`optional`" if the EXPRESS attribute is `OPTIONAL`; otherwise "`required`";

— zero-or-one is "`0`" if the EXPRESS attribute is `OPTIONAL`; otherwise "`1`".

**Ref** can only occur on an **attribute** or **inverse** element. If **ref** appears in an **attribute** or **inverse** directive, the following options shall not appear in it: **name**, **keep**, **use-id**, **map**, **implementation**.

NOTE The value of **ref** must be resolvable in the associated namespaces.

### 10.2.20 use

The **use** option controls the minimum cardinality of an EXPRESS attribute when it is mapped to the derived XML schema. **Use** can only occur on an **attribute** element.

When **use="optional"** is specified, the EXPRESS attribute shall be mapped to the derived XML schema using a minimum cardinality of zero. That is, it shall be declared `minOccurs="0"` or `use="optional"`, according to the nature of the accessor.

When **use="required"** is specified, the EXPRESS attribute shall be mapped to the derived XML schema using a mimimum cardinality of one. That is, it shall be declared `minOccurs="1"` or `use="required"`, according to the nature of the accessor.

NOTE 1 `use="optional"` and `minOccurs="1"` are the language defaults for XML schema and can therefore be omitted in the derived schema.

NOTE 2 If the basetype of the attribute to which **use="required"** is applied is an aggregate having a lower bound of 0, the lower bound of the aggregate is raised to 1

### 10.2.21 implementation

The **implementation** option controls the mapping of an EXPRESS entity data type or an EXPRESS attribute to the derived XML schema. The value of the **implementation** attribute shall be an XML schema type specification – a `simpleType` definition, or a `complexType` definition.

If the **implementation** option applies to an entity data type, its value shall be the XML data type that corresponds to that entity data type. The definition that is the value of the **implementation** option

 

shall appear verbatim in the derived XML schema *instead of* the definition of the XML data type that would otherwise correspond to the entity data type.

NOTE 1 In this case, the XML type definition will include the name by which this data type will be identified.

If the **implementation** option applies to an EXPRESS attribute, its value shall be the XML data type for the corresponding accessor element. In the derived XML schema, the XML data type specification that is the value of the **implementation** option shall appear verbatim as the data type of the accessor element.

NOTE 2 In this case, the XML type definition need not, and probably should not, include a name.

NOTE 3 It gives the user total control (and responsibility in case of errors) over how the EXPRESS element is represented in the generated XML schema by providing its final XML representation as the content of this option.

**implementation** can only occur on an **entity** or **attribute** element.

If **implementation** appears in the configuration of an entity data type, the following options shall not appear: **tag-source**, **tag-values**, **map**, **keep**, **exp-attribute**, **tagless**. If **name** appears, it shall be the name of the instance element, and the basis for other names corresponding to the selected entity data type, but not (necessarily) the name of the corresponding data type. In addition, the content of the **entity** directive shall be empty.

If **implementation** appears in the configuration of an EXPRESS attribute, the following options shall not appear: **map**, **keep**, **exp-attribute**, **tagless**, **use**, **use-id**, **notation**, **ref**.

NOTE 4 Because the XML Schema definition in the value of implementation will use XML-specific characters, the representation will be either a CDATA section or use the replacement rules for XML-specific characters that are specified in the XML specification. For example "<" is coded as "`<`"

NOTE 5 The validity of the given construct is the responsibility of the user.

### 10.2.22 facet

The **facet** option controls the mapping of an EXPRESS `ENTITY` to the derived XML schema. It provides an XML attribute, which cannot be derived by applying any of the other EXPRESS to XML mapping rules, to be added to the generated XML complexType. It may be used, for example, when it is necessary to create an abstract XML complexType from a non-abstract EXPRESS entity. **facet** can only occur on an **entity** element.

EXAMPLE

```
<entity select="simple_property_association"
 name="simple_property_value" facet="abstract=true">
 ...
</entity>
```

NOTE The value of this option must be an XML attribute valid in the context to be applied.

### 10.2.23 generate-keys

The **generate-keys** option controls the use of identity constraints and other complex XML schema constraints needed for validation of all EXPRESS constraints that are expressible in the XML Schema language. When **generate-keys="'true"**, the full validation capabilities provided by conformance class 2 of an XML schema document (see 4.2) are enabled. When **generate-keys="false"**, the complex XML schema constraints do not appear in the derived XML schema.

**generate-keys** can only occur on an **option** element.

NOTE The generation of `key` and `keyref` constraints is handled at the schema level, and therefore constraints are defined for every data type and usage in the schema, or not at all.

### 10.2.24 embed-schema-items

The **embed-schema-items** option causes elements and supporting items from the Base XML Schema (Annex C) to be embedded in the resulting schema, obviating the need to import these schemas. The value shall be true or false. The default value is false.

When the value is true, all XML data type definitions, attribute declarations and element declarations from Annex C that are used (directly or indirectly) in the definitions and declarations required for the context EXPRESS schema shall be copied verbatim into the resulting schema.

### 10.2.25 alias and prefix

The **alias** and **prefix** options are used to specify the namespace prefix to be used for a specified namespace. If the **alias** and **prefix** options are not specified, the prefix shall be determined by the pre-processor as specified in 8.7. The value of **alias** shall be a namespace URI. The value of **prefix** shall be a prefix representing the namespace URI. The **alias** and **prefix** attributes have no default.

### 10.2.26 select

The **select** configuration option specifies the collection of EXPRESS data types to which an **entity** or **type** scoping directive applies. The members of that collection are identified by their EXPRESS identifiers. The value of the select XML attribute specifies a list of patterns. The **entity** or **type** directive shall apply to every EXPRESS data type in, or interfaced into, the context schema whose identifier matches any of the patterns in the list.

The **select** attribute has no default value. The values of the select attribute shall conform to the XML Schema data type:

```
<xs:simpleType name="patterns">
 <xs:restriction>
    <xs:simpleType>
       <xs:list itemType="xs:normalizedString" />
    </xs:simpleType>
    <xs:minLength value="1"/>
 </xs:restriction>
</xs:simpleType>
```

That is, the value shall be a simple list of one or more pattern strings, each of which is composed entirely of characters from the list specified in Table 9.

 

**Table 9 — Pattern strings for select**

| Pattern element | Matching rule |
|---|---|
| any letter | Matches that letter. If **naming-convention="preserve-case"** is specified, upper case letters are distinguished from lower-case letters; in any other case they are not distinguished. |
| any digit | matches that digit. |
| _ | matches the LOW LINE character (underscore) |
| ? | matches any (one) character. |
| @ | matches any letter. |
| # | matches any digit |
| ^ | matches any upper case letter when **naming-convention="preserve-case"** is specified; matches any letter otherwise |
| ~ | matches any lower case letter when **naming-convention="preserve-case"** is specified; matches any letter otherwise |
| ! | The pattern element includes the next pattern character. The pattern element matches any character that (next) pattern character does not match. For example, !@ matches any character that is not a letter; ! matches any character that is not a LOWLINE. |
| * | The pattern element includes the preceding pattern element. It matches zero or more consecutive occurrences of that pattern element. Exception: ?* does not match the first character that matches the next pattern element, if any. For example, In ?*_, the ?* element matches every character up to, but not including the first LOW LINE, and the _ element matches that LOW LINE, if any. |
| + | The pattern element includes the preceding pattern element. It matches one or more consecutive occurrences of that pattern element. Exception: ?+ does not match the first character that matches the next pattern element, if any. |

For each EXPRESS data type identifier in, or interfaced into, the context schema, and each pattern in the list, the matching algorithm is as follows:

— Each character of the pattern string is compared to the corresponding character(s) of the identifier, as specified in Table 9. If any pair of corresponding characters does not match, the identifier does not match that pattern.

— If all characters of the pattern string match corresponding characters of the identifier, and the identifier has no unmatched characters, the identifier matches the pattern.

NOTE 1 Certain special characters in the pattern string may match more than one character in the identifier.

— If the identifier matches any pattern in the list, the corresponding EXPRESS data type is included in the scope of the directive. If the identifier matches no pattern in the list, it is not included in the scope of the directive.

NOTE 2 If the pattern string consists entirely of letters, digits and LOW LINE characters, it is an EXPRESS identifier, and it matches exactly that identifier and no other. So the "list of patterns" can in fact be a list of identifiers of the data types that are to be in the scope of the directive.

## 10.3 Scoping elements

This subclause specifies the requirements for the XML elements in the content of a by-value `configuration` element. In general, the XML element specifies the scope of application of one or more configuration directives, and each directive is specified as an XML attribute.

The **option** element specifies global options and defaults. The **schema** element controls the definition of schema and population concepts and of the derived XML schema itself. The **type** and **entity** elements control the representation of specific EXPRESS data types. **Entity** elements may contain **attribute** elements and **inverse** elements that control the representation of specific attributes of that entity. **Type** and **attribute** elements may contain **aggregate** elements that control the representation of nested aggregates within them.

### 10.3.1 Option element

The **option** element specifies global options and defaults.

There may be any number of **option** elements in a `configuration` element. Multiple **option** elements within a **configuration** element are equivalent to a single **option** element containing the collection of all the XML attributes of all of the **option** elements. It is an error for a `configuration` element to contain multiple **option** elements that specify the same XML attributes.

```
<xs:element name="option" >
  <xs:complexType>
    <xs:attribute name="inheritance" type="xs:boolean"
       default="false"/>
    <xs:attribute name="exp-type" type="exp-type"
       default="unspecified"/>
    <xs:attribute name="entity-attribute"
        type="cnf:exp-attribute-global" default="double-tag"/>
    <xs:attribute name="concrete-attribute"
        type="cnf:exp-attribute-global" default="attribute-tag"/>
    <xs:attribute name="tagless" type="cnf:boolean_or_unspecified"
        default="unspecified"/>
    <xs:attribute name="naming-convention" type="cnf:naming-convention"
        default="initial-upper"/>
    <xs:attribute name="keep-all" type="cnf:attributeType"/>
    <xs:attribute name="generate-keys" type="xs:boolean"
        default="true"/>
  </xs:complexType>
</xs:element>
```

All attributes set default values that can be overridden in other configuration elements.

### 10.3.2 Type element

The **type** element controls the representation of EXPRESS defined data types, EXPRESS primitive types, and anonymous aggregates.

```
<xs:element name="type" >
   <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element ref="cnf:aggregate"/>
     </xs:choice>
     <xs:attribute name="select" type="cnf:patterns" use="required"/>
     <xs:attribute name="name" type="xs:NMTOKEN" use="optional"/>
     <xs:attribute name="map" type="xs:NMTOKEN" use="optional"/>

     <xs:attribute name="exp-type" type="exp-type"
        use="optional"/>
     <xs:attribute name="tagless" type="boolean_or_unspecified"
        use="optional"/>
     <xs:attribute name="notation" type="xs:normalizedString"
        use="optional"/>
     <xs:attribute name="keep" type="xs:boolean" default="true"/>
```

 

```
      <xs:attribute name="flatten" type="xs:boolean" use="optional"/>
    </xs:complexType>
  </xs:element>
```

The collection of EXPRESS data types being configured is specified by the value of the **select** attribute, as follows:

— For defined data types, the value of the **select** attribute shall be as specified in 10.2.26, using the EXPRESS identifiers for the defined data types.

— For a primitive type, the value of the **select** attribute shall be the EXPRESS keyword for the primitive type (`BINARY`, `BOOLEAN`, `INTEGER`, `LOGICAL`, `NUMBER`, `REAL`, `STRING`).

— For anonymous aggregation types, the value of the **select** attribute shall consist of the characters '`Seq-`' followed by the base-type of the aggregation type.

NOTE 1 All aggregations (`ARRAY`, `BAG`, `LIST`, `SET`) of a given base type have the same configuration.

The value of the **name** attribute, if present, shall specify the name to be used for the XML data type and instance element corresponding to the EXPRESS data type. See 8.3 and 8.4.

NOTE 2 The name attribute can be used to give an XML name to EXPRESS anonymous aggregates.

The value of the **map** attribute shall be the name of the XML data type that is used to represent values of the EXPRESS data type. The permitted values for this option depend on the EXPRESS data type. See 10.2.13.

The **exp-type** attribute controls the use of by-reference and by-value instance elements for the specified defined data type. See 10.2.2.

The **tagless** and **flatten** options may only be present when the data type is an aggregation data type. See 10.2.8.

The optional **aggregate** child element shall be used to configure an aggregation data type that is nested in the selected (aggregate) data type. See 10.3.6.

The **keep** attribute controls the use of the type in the derived XML schema. See 10.2.11.

### 10.3.3 Entity element

The **entity** element controls the representation of EXPRESS entity types.

```
<xs:element name="entity" >
  <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="unbounded" >
       <xs:element ref="cnf:attribute"/>
       <xs:element ref="cnf:inverse"/>
     </xs:choice>
     <xs:attribute name="select" type="cnf:patterns" use="optional" />
     <xs:attribute name="synthetic" use="optional" >
      <xs:simpleType>
       <xs:restriction>
        <xs:simpleType>
          <xs:list itemType="xs:Name"/>
        </xs:simpleType>
        <xs:minLength value="2"/>
       </xs:restriction>
      </xs:simpleType>
```

```
        </xs:attribute>
        <xs:attribute name="inheritance" type="xs:boolean"
          default="false"/>
        <xs:attribute name="name" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="tag-source" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="tag-values" type="xs:NMTOKENS" use="optional"/>
        <xs:attribute name="map" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="exp-type" type="exp-type"
           use="optional"/>
        <xs:attribute name="content" type="cnf:content"
           use="optional"/>
        <xs:attribute name="exp-attribute" type="cnf:exp-attribute"
           default="unspecified"/>
        <xs:attribute name="tagless" type="cnf:boolean_or_unspecified"
           use="optional"/>
        <xs:attribute name="keep" type="xs:boolean" default="true"/>
        <xs:attribute name="new" type="xs:boolean" default="false"/>
        <xs:attribute name="implementation" type="xs:string"
           use="optional"/>
        <xs:attribute name="facet" type="xs:string"
           use="optional"/>
      </xs:complexType>
    </xs:element>
```

The collection of EXPRESS entity data types being configured shall be specified by either:

— the value of the **select** attribute, as specified in 10.2.26, or

— the value of the **synthetic** attribute, as specified in 10.3.3.1.

Exactly one of the **select** attribute and the **synthetic** attribute shall be specified.

The **new** attribute allows the creation of a new entity data type that is based on the (single) EXPRESS entity data type specified by the value of the **select** attribute. If **new="true"** is specified, the **name** attribute shall be present, and the value of the **select** attribute shall specify exactly one EXPRESS entity data type. If **new="true"** is specified, the entity directive shall be interpreted as specified in 10.3.3.2.

The **name** attribute, if included, shall specify the name for the XML data type that shall represent the entity data type and for the instance element that shall be used for the corresponding entity instances. If the **entity** directive specifies more than one entity data type via the **select** attribute, the **name** attribute shall not be present.

NOTE 1 If the **entity** directive defines a new data type, the **name** option is required in order to give it a name. If the **entity** directive defines a synthetic data type, the **name** attribute may be used to specify a name for the synthetic data type. If no **name** attribute appears, a name will be constructed as specified in 8.5.4.5.

The **tag-source** and **tag-values** attributes provide an alternative mechanism for naming elements corresponding to the specified entity data type. See 10.2.17.

The **map** attribute, if included, shall specify the base type (by extension) for the XML data type that shall represent the entity data type. See 10.2.13.

NOTE 2 By defining a new base-type for xs:extension the subtype-supertype-relationship of the underlying EXPRESS schema may be destroyed.

The **exp-type** attribute controls the use of by-reference and by-value instance elements for the specified entity data type. See 10.2.2.

 

The **keep** attribute controls the use of the type in the derived XML schema. See 10.2.11.

The **implementation** attribute allows the user to create an arbitrary implemementation of the selected EXPRESS entity. See 10.2.21

The **facet** attribute allows the user to add an XML attribute to the complexType generated. See 10.2.21

The other attributes provide default values for the configuration options that apply to the EXPRESS attributes of the specified entity data type (see 10.3.4).

EXAMPLE 1

In EXPRESS:

```
ENTITY Organization;
  name: STRING;
END_ENTITY;
ENTITY Supplier SUBTYPE OF (Organization);
  supplier_id: STRING;
  rating: INTEGER;
END_ENTITY;
ENTITY Customer SUBTYPE OF (Organization);
  customer_id: STRING;
  sales_office: Organization
END_ENTITY;
```

A single entity data type can be configured as follows:

```
<entity select="organization" exp-type="root" />
```

Multiple entity data types can be configured using a list:

```
<entity select="Customer Supplier" exp-type="root" />
```

Multiple entity data types can also be configured using patterns:

```
<entity select="?+er" exp-type="root" />
```

given only the above declarations, will select both Customer and Supplier. Of course, in a larger schema, the pattern `"?+er"` will select all data types ending in `"er"`.

EXAMPLE-2

In EXPRESS:

```
ENTITY simple_property_value
  ABSTRACT SUPERTYPE;
  described_element : simple_property_select;
  value_name : STRING;
  value_type : STRING;
 END_ENTITY;

ENTITY simple_string_value
  SUBTYPE OF ( simple_property_value );
  value_specification : string_select;
 END_ENTITY;

ENTITY simple_property_association;
  desribed_element : simple_property_select;
  specified_value : string_value;
```

```
END_ENTITY;
```

This example stems from the AP214-.ARM to PLM-Services mapping where the abstract entity "simple_property_value" is replaced by combination of "simple_property_association" and "simple_property_string" for the reason to come up with a condensed nested instantiation. Therefore the following mapping is used:

```
  <entity select="simple_property_association"
name="Simple_property_value"
       facet="abstract=true">
    <attribute select="specified_value" exp-attribute="no-tag"/>
  </entity>

  <entity new="true" name="Simple_string_value"
map="Simple_property_value">
    <attribute new="true" name="Value_specification"
map="Translatable_string"

  select="simple_property_association.specified_value.string_value.val
ue_specification" />
  </entity>
```

Those configuration will lead to the following representation in XML schema:

```
<xs:complexType name="Simple_property_value" abstract="true">
    <xs:complexContent>
        <xs:extension base="PLM_object">
            <xs:sequence>
                <xs:element name="Value_name" type="xs:string"/>
                <xs:element name="Value_type" type="xs:string"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="Simple_string_value">
    <xs:complexContent>
        <xs:extension base="Simple_property_value">
            <xs:sequence>
                <xs:element name="Value_specification"
type="Translatable_string"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

#### 10.3.3.1 Synthetic entity types

An EXPRESS entity instance instantiates one or more nodes in the type graph associated with its entity data type(s), as defined in 8.5.1. The subgraph containing exactly the instantiated nodes has one or more "leaf nodes" and one or more "root nodes". The entity instance is said to be *characterized* if the subgraph has exactly one leaf node; the entity instance is said to be *uncharacterized* if the subgraph has more than one leaf node.

NOTE 1 If the entity instance is characterized, the EXPRESS entity data type corresponding to the leaf node is declared in the EXPRESS schema to have all the attributes that appear in the entity instance. If the entity instance is uncharacterized, there is no single entity data type in the EXPRESS schema that has all the attributes of the instance. This situation can arise when there are ANDOR relationships among subtypes.

When the **synthetic** attribute is specified, the **entity** directive shall be interpreted as defining a *synthetic data type* (see 8.5.4.5) that characterizes a set of otherwise uncharacterized entity instances. The value of the **synthetic** attribute shall specify the set of EXPRESS entity data types that will be

 

instantiated in each entity instance characterized by the synthetic entity data type. That is, the synthetic entity data type shall be considered to be a subtype of all the EXPRESS entity data types that appear in the list (as described in 8.5.1). It is sufficient for the list to contain the EXPRESS entity data types that correspond to the leaf nodes of the subgraph.

The default XML name for a synthetic data type shall be the list of identifiers appearing in the value of the **synthetic** attribute, each converted to an XML name as specified in 8.2.1, separated by HYPHENs.

NOTE 2 If the entity directive defines a synthetic data type, the **name** attribute may be used to specify a name for the synthetic data type. If no **name** attribute appears, a name will be constructed as specified in 8.5.4.5.

EXAMPLE

In EXPRESS:

```
ENTITY Organization;
  name: STRING;
END_ENTITY;
ENTITY Supplier SUBTYPE OF (Organization);
  supplier_id: STRING;
  rating: INTEGER;
END_ENTITY;
ENTITY Customer SUBTYPE OF (Organization);
  customer_id: STRING;
  sales_office: Organization
END_ENTITY;
```

There can be instances of Organization that are both Suppliers and Customers, and have all five of the declared attributes. In order to represent these properly in the XML document, it is necessary to provide the following declaration:

```
<entity synthetic="Customer Supplier" name="Customer-and-supplier" />
```

#### 10.3.3.2 New entity types

When **new="true"**, the context schema shall be interpreted as having a new entity data type that is based on the EXPRESS entity data type specified by the value of the **select** attribute.

The identifier for the new entity shall be that specified by the value of the **name** option. The new entity data type shall not be abstract. It shall have no supertypes and no subtypes.

The new entity data type shall have exactly the attributes specified by the **attribute** directives contained within the **entity** directive. All of those attributes shall be either new or taken from the attributes of the EXPRESS entity data type on which the new entity data type is based. Unlike other cases, the attributes specified for the new entity data type may include inherited attributes of the entity data type on which it is based.

NOTE 1 Since the new data type has no supertypes, none of its attributes will be treated as inherited.

For the new entity data type, the context schema shall contain the entire set of definitions and declarations specified in 8.6. The **entity** directive and the **attribute** directives it contains shall be interpreted as configuring those definitions and declarations.

The new entity data type does not replace the entity data type on which it is based. Unless that entity data type is explicitly selected with **keep="false"** (see 10.2.11), it too shall be mapped to the derived XML schema.

NOTE 2 A given EXPRESS entity data type can be the basis for a corresponding XML data type and zero or more **new** ones..

### 10.3.4 Attribute element

The **attribute** element controls the representation of one EXPRESS attribute. The EXPRESS entity data type that owns the attribute is specified by the containing **entity** element.

```
<xs:element name="attribute" >
  <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="unbounded" >
       <xs:element ref="cnf:aggregate" />
     </xs:choice>
     <xs:attribute name="select" type="xs:NMTOKEN" use="optional"/>
     <xs:attribute name="new" type="xs:boolean" default="false"/>
     <xs:attribute name="name" type="xs:NMTOKEN" use="optional"/>
     <xs:attribute name="exp-attribute" type="cnf:exp-attribute"
        use="optional"/>
     <xs:attribute name="content" type="cnf:content"
        use="optional"/>
     <xs:attribute name="aggregate-content" type="cnf:content"
        use="optional"/>
     <xs:attribute name="map" type="xs:NMTOKEN" use="optional"/>
     <xs:attribute name="use-id" type="xs:boolean" use="optional"/>
     <xs:attribute name="keep" type="xs:boolean" default="true" />
     <xs:attribute name="tagless" type="cnf:boolean_or_unspecified"
        use="optional"/>
     <xs:attribute name="notation" type="xs:normalizedString"
        use="optional"/>
     <xs:attribute name="flatten" type="xs:boolean" use="optional"/>
     <xs:attribute name="ref" type="xs:NMTOKEN"
        use="optional" />
     <xs:attribute name="use" type="xs:NMTOKEN"
        use="optional" />
<xs:attribute name="implementation" type="xs:string"
        use="optional" />
  </xs:complexType>
</xs:element>
```

When **new="true"** is specified, a new EXPRESS explicit attribute shall be created for all of the owning entity data types selected by the **entity** directive in which the **attribute** directive appears. The **name** option shall appear in the **attribute** directive, and the identifier for the new attribute shall be as specified by the value of the **name** option. The identifier shall be unique among the names of the attributes of each of the selected entity data types. The data type of the new attribute shall be `STRING`. The new attribute shall be mapped to the derived XML schema as specified in 8.6. All of the other options on the **attribute** directive shall be interpreted as applying to the mapping of this new attribute. In particular, the **map** option or the **implementation** option (see below) may be used to specify the actual XML data type.

When **new="true"** is specified, the **select** attribute shall not appear.

When **new="false"**, the **select** attribute shall appear, and its value shall be the EXPRESS identifier for the EXPRESS attribute being configured, converted to an XML name as specified in 8.1.2. The

 

attribute name may be qualified by prepending the identifier for the EXPRESS entity data type that defines the attribute and a PERIOD character.

If the inheritance mapping applies to the owning entity data type, the value of the **select** attribute shall specify an EXPRESS attribute that is declared in the entity declaration for that entity.

If the inheritance-free mapping applies to the owning entity data type, the value of the **select** attribute shall specify an EXPRESS attribute that is declared in the entity declaration for that entity data type or any of its supertypes.

NOTE Inherited EXPRESS attributes may only be (re-)configured when the inheritance-free mapping applies to the owning entity data type: When the inheritance mapping is used, an attribute can only be configured for the entity whose declaration contains it — the configuration affects the XML Schema model of the attribute, and that model is automatically inherited.

The **name** option specifies the name of the accessor element or accessor attribute that represents the EXPRESS attribute.

The **exp-attribute** option specifies how the EXPRESS attribute is represented in the XML. See 10.2.5.

The **content** option specifies whether values of this attribute are represented by-value or by-reference in the XML data. See 10.2.3.

The **aggregate-content** option determines whether the base element values of a (possibly nested) aggregate value are represented by reference or value. See 10.2.4.

The **map** option declares the XML type that shall be used to represent a given EXPRESS data type. See 10.2.13.

The **use-id** option specifies whether the by-value representation of the EXPRESS attribute may have an **id** XML attribute. See 10.2.10.

The **tagless** option controls the representation of aggregates, when the configured attribute is an aggregate type. See 10.2.8.

The flatten option controls the representation of nested aggregates, when the configured attribute is an aggregate type. See 10.2.9.

The **notation** attribute specifies the form of representation of a `BINARY` or `STRING` value of the attribute. See 10.2.16.

The **keep** attribute controls the appearance of the EXPRESS attribute in the derived XML schema. See 10.2.11.

The **ref** attribute causes the mapping specified in 8.6 to be replaced by a reference to an XML Schema attribute-definition or element definition in one of the associated namespaces. See 10.2.19.

The **use** attribute may change the minimum cardinality of the attribute. See 10.2.20.

The **implementation** attribute specifies the XML data type of the corresponding accessor element or attribute by an explicit XML Schema construct. See 10.2.21

EXAMPLE 1

```
<entity select="axis2_placement_3d">
```

```
  <attribute select="placement.location" exp-attribute="attribute-
tag"/>
  <attribute select="ref_direction" name="dir" />
</entity>
```

EXAMPLE 2

In EXPRESS:

```
ENTITY string_with_language;
  contents : STRING;
  language_specification : language;
 INVERSE
  used_by : SET [1:?] OF multi_language_string FOR
primary_language_dependent_string;
END_ENTITY;
```

In the configuration language:

```
<entity select="string_with_language" name="translation" exp-
type="value">
  <attribute select="contents" map="xs:string"
    exp-attribute="no-tag-simple"/>
  <attribute select="language_specification" ref="xml:lang"
    exp-attribute="attribute-content"/>
</entity>
```

IN the derived XML schema:

```
<xs:complexType name="Translation">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute ref="xml:lang" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

### 10.3.5 Inverse element

In EXPRESS, when an entity, called the *owning entity*, has an explicit attribute whose data type is an entity data type or an aggregation type whose base-type is an entity data type, a relationship is created between the two entity data types. The data type, or the base-type, of the attribute is called the *associated entity*. In the context schema, the associated entity may or may not have an inverse attribute whose data type is the owning entity.

The **inverse** element creates and configures a new explicit attribute in the context schema, by inverting an existing relationship in the context schema.

The **inverse** element shall be contained within an **entity** element. The **entity** element selects the associated entity data type(s) in the context schema. For each associated entity data type, the **inverse** element creates an explicit attribute whose data type is the owning entity.

```
<xs:element name="inverse">
  <xs:complexType>
    <xs:attribute name="select" type="xs:NMTOKEN" use="optional"/>
    <xs:attribute name="name" type="xs:NMTOKEN" use="optional"/>
    <xs:attribute name="exp-attribute" type="exp-attribute"
        use="optional"/>
```

 

```
      <xs:attribute name="content" type="cnf:contained" use="optional"/>
      <xs:attribute name="tagless" type="boolean_or_unspecified"
          use="optional"/>
       <xs:attribute name="invert" type="xs:NMTOKEN" use="optional"/>
       <xs:attribute name="map" type="xs:NMTOKEN" use="optional"/>
       <xs:attribute name="keep" type="xs:boolean" default="false" />
       <xs:attribute name="minOccurs" type="xs:nonNegativeInteger"
          default="0"/>
       <xs:attribute name="maxOccurs"
          type="cnf:nonNegativeInteger_or_unbounded" default="unbounded"
          use="optional"/>
       <xs:attribute name="ref" type="xs:NMTOKEN" use="optional" />
    </xs:complexType>
  </xs:element>
```

There are two cases:

— the associated entity has an existing inverse attribute for the relationship to be inverted. In this case, the **select** option is present in the **inverse** directivet, and the **invert** option is not.

— an entirely new attribute is created for the associated entity. In this case, the **invert** option is present in the **inverse** directive, and the **select** option is not.

In any inverse directive, exactly one of **select** and **invert** shall be present.

When the **select** attribute is present, its value shall be the EXPRESS identifier for an existing inverse attribute of all the entity data types selected by the **entity** directive in which the **inverse** directive appears. In this case, the explicit attribute to be inverted is specified in the existing inverse attribute. If the **select** attribute is present, the **name** attribute is optional, and if present, it renames the inverse attribute identified by the **select** attribute.

If the **invert** attribute is present, the **inverse** directive declares a new inverse attribute for all the entity data types selected by the **entity** directive in which the **inverse** directive appears. In this case, the value of the **invert** attribute declares the explicit attribute that is to be inverted. The value of this attribute consists of the EXPRESS identifier for the owning entity followed by PERIOD followed by the identifier for the explicit attribute (of the owning entity) that creates the relationship. In this case, the **name** attribute shall be present and shall be used as the name of the new EXPRESS attribute for the associated entity data types.

NOTE 1 There can be more than one associated entity data type only when the entity directive selects a set of subtypes of the data type of the explicit attribute.

The **minOccurs** and/or **maxOccurs** configuration attributes specify corresponding lower and upper bounds for the data type of the new explicit attribute in the context schema. If the value of **maxOccurs** is 1 (the default), the data type of the new explicit attribute in the context schema shall be the owning entity data type; and if **minOccurs="0"** is specified, the explicit attribute shall be treated as `OPTIONAL`. Otherwise the data type of the new explicit attribute shall be a `SET OF` the owning entity data type, with the bounds given by **minOccurs** and **maxOccurs**. The new explicit attribute shall be mapped to the derived XML schema as specified in 8.6.

The specified values of **minOccurs** and **maxOccurs** shall not conflict with implicit or explicit constraints in the context schema, although they may narrow such constraints.

In either case, the original explicit attribute is mapped to the derived XML schema as an inverse attribute of the owning entity, and the new attribute, or the former inverse attribute, is mapped to the derived XML schema as an explicit attribute of the associated entity data type(s).

NOTE 2 That is what is meant by "inverting the relationship".

The value of the **keep** configuration attribute on the **inverse** directive applies to the explicit attribute that is to be inverted. An explicit **attribute** configuration element that selects the explicit attribute may also specify the **keep** attribute.

The other configuration options have the same value as they do in the **attribute** configuration element.

EXAMPLE 1

The following example includes a declared inverse from the EXPRESS schema in the XML schema:

```
<entity select="application_context" >
  <inverse select="context_elements"/>
</entity>
```

EXAMPLE 2

The following example creates an inverse in the context schema that does not appear in the original EXPRESS schema

```
<entity select="product">
  <inverse name="formation"
           invert="product_definition_formation.of_product"/>
</entity>
```

### 10.3.6 Aggregate element

The **aggregate** element controls the representation of nested aggregates. It may be used within a **type** or **attribute** element that corresponds to an EXPRESS type that is a nested aggregate (the top-level of nesting is handled in the **type** or **attribute** element).

```
<xs:element name="aggregate">
  <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="unbounded" >
       <xs:element ref="aggregate" />
     </xs:choice>
     <xs:attribute name="name" type="xs:NMTOKEN" use="optional"/>
     <xs:attribute name="content" type="cnf:content"
        use="optional"/>
     <xs:attribute name="tagless" type="boolean_or_unspecified"
       use="optional"/>
     <xs:attribute name="use-id" type="xs:boolean" use="optional"/>
     <xs:attribute name="flatten" type="xs:boolean" use="optional"/>
  </xs:complexType>
</xs:element>
```

The value of the **name** configuration option specifies the name for the corresponding XML element, if any.

NOTE As specified in 8.2.2, certain uses of **tagless** will cause the aggregate in question not to have a corresponding element.

The other options have the same meaning as they do on the **attribute** element as described in 10.3.4.

 

### 10.3.7 Schema element

The **schema** element controls all aspects of the element <xs:schema>, see 8.7.3. It shall occur within a configuration only once.

```
<xs:element name="schema">
    <xs:complexType>
        <xs:choice maxOccurs="unbounded">
            <xs:element ref="cnf:namespace" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element ref="cnf:include" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element ref="cnf:containerObject" minOccurs="0"
maxOccurs="1"/>
            <xs:element ref="cnf:additionalObject" minOccurs="0"
                    maxOccurs="unbounded"/>
        </xs:choice>
        <xs:attribute name="targetNamespace"    type="xs:anyURI" />
        <xs:attribute  name="elementFormDefault" type="cnf:qual"/>
        <xs:attribute name="attributeFormDefault"   type="cnf:qual" />
        <xs:attribute name="defaultRootObjectType" type="xs:string" />
        <xs:attribute name="defaultObjectType" type="xs:string" />
        <xs:attribute name="schema-version"    type="xs:string"  />
        <xs:attribute name="embed-schema-items" type="xs:boolean"
            default="false"/>
    </xs:complexType>
</xs:element>
```

The value of the **targetNamespace** configuration option specifies the name for the target namespace the generated XML schema shall be using.

The configuration options **attributeFormDefault** and **elementFormDefault** define the default form attribute value for xs:attribute and xs:element. Valid values are "qualified" and "unqualified".

The **defaultRootObjectType** (if provided) identifies the abstract type (supertype) of all independently instantiable entity data types.

The **defaultObjectType** specifies a replacement name for the exp:Entity data type and instance element. If defined as **additionalObject** as well, it may have a different attribute set from exp:Entity.

The **schemaVersion** specifies the value of the version attribute of xs:schema to be used. It identifies the version of the XML-Schame-Language as defined by W3C. Examples: 1.0, 1.0.1, 1.1, 1.2, 1.3b1, 2.0 aso.

#### 10.3.7.1 Namespace element

The **namespace** element names one namespace to be included into the definition of the generated <xs:schema> statement. It shall occurs within <schema> element, see 10.3.7. It can occur zero or more times.

```
<xs:element name="namespace">
 <xs:complexType>
    <xs:attribute name="alias" type="xs:anyURI"/>
    <xs:attribute name="prefix" type="xs:string"/>
 </xs:complexType>
</xs:element>
```

The value of the **alias** configuration option contains a valid namespace name.

The **prefix** (if provided) identifies the abbreviation of the namespace to be used. An empty prefix is allowed only once in the set of all namespaces named.

#### 10.3.7.2 Include element

The **include** element describes the content of <xs:include> statements to be included into the generated <xs:schema> statement. It shall occurs within <schema> element, see 10.3.7. It can occur zero or more times, depending on the number of includes needed.

```
<xs:element name="include">
 <xs:complexType>
    <xs:attribute name="urn" type="xs:anyURI" />
    <xs:attribute name="schema-location" type="xs:string" />
 </xs:complexType>
</xs:element>
```

The value of the **urn** configuration option contains a XML schema to be included..

The **schema-location** identifies the location of the schema to be included.

#### 10.3.7.3 ContainerObject element

The **containerObject** (if present) renames the exp:uos element and may supply additional attributes. Its content model is the same as uos.

```
<xs:element name="containerObject">
    <xs:complexType>
        <xs:attribute name="name"type="xs:NMTOKEN" use="required"/>
        <xs:attribute name="implementation" type="xs:string"
            use="optional"/>
    </xs:complexType>
</xs:element>
```

The value of the **name** configuration option contains the replacement name of the uos element.

The **implementation** contains the complete XML implementation of the replacement type, except for the enumeration of all root elements.

#### 10.3.7.4 AdditionalObject element

The **additionalObject** (if present) is used to insert XML elements or complexTypes representing high level classifications that are not part of the original EXPRESS model. The inserts are always abstract.

```
<xs:element name="additionalObject">
    <xs:complexType>
        <xs:attribute name="name"type="xs:NMTOKEN" use="required"/>
        <xs:attribute  name="implementation" type="xs:string"
            use="optional"/>
    </xs:complexType>
</xs:element>
```

The value of the **name** configuration option defines the name of the additional element inserted.

The **implementation** contains the complete XML implementation of the element inserted.

 

NOTE The inserted element can have additional XML attributes.

#### 10.3.7.5 Example of usage of <schema> configuration

This section shows one example of using the <schema> element and its subelements. It stems from the generation of the PLM-Services V1.0 out of the AP214-ARM. The example is reduced to show the main aspects of <schema> only.

```
<iso_10303_28 version="2.0">
  <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    id="ID0" version_id="1">
    . . .
    <schema
    targetNamespace="http://schema.omg.org/spec/PLM/1.0/PLMInformationalModel.xsd"
      elementFormDefault="unqualified"
      attributeFormDefault="unqualified"
      defaultRootObjectType="PLM_root_object"
      defaultObjectType="PLM_object"
      schema-version="1.0">

      <namespace namespace-urn="http://www.w3.org/2001/XMLSchema"
        namespace-prefix="xs"/>
      <namespace
      namespace-urn="http://schema.omg.org/spec/PLM/1.0/PLMInformationalModel.xsd"
        namespace-prefix=""/>
      <import namespace-urn="http://www.w3.org/XML/1998/namespace"
        schemaLocation="xml.xsd"/>

      <!--containerObject to be generated
        <xs:complexType name="PLM_container">
          <xs:choice maxOccurs="unbounded" minOccurs="0" />
          <xs:attribute name="key" type="xs:ID" use="required"/>
          <xs:attribute name="version_id" type="xs:string"
          use="required"/>
          <xs:attribute ref="xml:lang" use="optional"/>
        </xs:complexType>
      -->
      <containerObject xml-implementation=
      "&lt;xs:complexType
      name=&quot;PLM_container&quot;  &gt;
      &lt;xs:choice maxOccurs=&quot;unbounded&quot;
      minOccurs=&quot;0&quot;  /&gt;
      &lt;xs:attribute name=&quot;uid&quot; type=&quot;xs:ID&quot;
      use=&quot;required&quot; /&gt;
      &lt;xs:attribute name=&quot;version_id&quot;
      type=&quot;xs:string&quot; use=&quot;required&quot; /&gt;
      &lt;xs:attribute ref=&quot;xml:lang&quot;
      use=&quot;optional&quot; /&gt;
      &lt;/xs:complexType &gt; "/>

      <!-- additionalObject to be generated
        <xs:element name="PLM_container" type="PLM_container"/>
      -->
      <additionalObject xml-implementation="&lt;xs:element
        name=&quot;PLM_container&quot; type=&quot;PLM_container&quot;
        /&gt;"/>

      <!-- additionalObject "PLM_object" to be generated
        <xs:complexType abstract="true" name="PLM_object">
        <xs:attribute name="key" type="xs:ID" use="required"/>
```

```
            </xs:complexType>
          -->
          <additionalObject xml-implementation="&lt;xs:complexType
            name=&quot;PLM_object&quot; abstract=&quot;true&quot; &gt;
            &lt;xs:attribute name=&quot;uid&quot; type=&quot;xs:ID&quot;
            use=&quot;required&quot; /&gt;
            &lt;/xs:complexType &gt; "/>

          <!-- additionalObject "PLM_root_object" to be generated
          <xs:complexType abstract="true" name="PLM_root_object">
              <xs:complexContent>
                 <xs:extension base="PLM_object"/>
              </xs:complexContent>
            </xs:complexType>
          -->
          <additionalObject xml-implementation="&lt;xs:complexType
            name=&quot;PLM_root_object&quot; abstract=&quot;true&quot;
            &gt; lt;xs:complexContent &gt;
            &lt;xs:extension base=&quot;PLM_object&quot; /&gt;
            &lt;/xs:complexContent &gt;
            &lt;/xs:complexType &gt; "/>

        </schema>
```

The result of the configuration above is the following extract of the generated PLM-Services-1.0

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://schema.omg.org/spec/PLM/1.0/PLMInformationalModel.xsd"
          xmlns="http://www.omg.org/PLMServices1.0/XMLSchema"
          xmlns:xs="http://www.w3.org/2001/XMLSchema"
          elementFormDefault="unqualified"
attributeFormDefault="unqualified" version="1.0">
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
           schemaLocation="http://www.w3.org/2001/xml.xsd"/>

  <xs:element name="PLM_container" type="PLM_container"/>

  <xs:complexType name="PLM_container">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="Accuracy" type="Accuracy"/>
      <!--other elements removed -->
    </xs:choice>
    <xs:attribute name="uid" type="xs:ID" use="required"/>
    <xs:attribute name="version_id" type="xs:string" use="required"/>
    <xs:attribute ref="xml:lang" use="optional"/>
  </xs:complexType>

  <xs:complexType name="PLM_object" abstract="true">
    <xs:attribute name="uid" type="xs:ID" use="required"/>
  </xs:complexType>

  <xs:complexType name="PLM_root_object" abstract="true">
    <xs:complexContent>
      <xs:extension base="PLM_object"/>
    </xs:complexContent>
  </xs:complexType>

<!-- other generated parts not shown -->

</xs:schema>
```

### 10.3.8 UosElement element

The **uosElement** element controls the representation of the unit of serialization.

```
<xs:element name="uosElement" >
   <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element ref="cnf:addAttribute"/>
     </xs:choice>
     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
   </xs:complexType>
</xs:element>
```

The value of the **name** attribute, if present, shall specify the name to be used for the XML data type and instance element representing the unit of serialization. See 8.8.

Any **addAttribute** elements present specify XML attributes to be included in the definition of the unit of serialization data type.

### 10.3.9 UosEntity element

The **uosEntity** element controls the representation of the generic entity element and data type.

```
<xs:element name="uosEntity" >
   <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element ref="cnf:addAttribute"/>
     </xs:choice>
     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
   </xs:complexType>
</xs:element>
```

The value of the **name** attribute, if present, shall specify the name to be used for the XML data type and instance element representing the generic entity type. See 8.5.3.3.

Any **addAttribute** elements present specify XML attributes to be included in the definition of the generic entity data type.

### 10.3.10 RootEntity element

The **rootEntity** element controls the representation of the generic entity element and data type.

```
<xs:element name="rootEntity" >
   <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element ref="cnf:addAttribute"/>
     </xs:choice>
     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
   </xs:complexType>
</xs:element>
```

The value of the **name** attribute, if present, shall specify the name to be used for the XML data type and instance element representing the root entity type. See 8.5.3.4.

Any **addAttribute** elements present specify XML attributes to be included in the definition of the root entity data type.

## 10.4 Configuration attributes

The Configuration Schema includes the following definitions of the XML data types: `exp-type`, `content`, `exp-attribute`, `exp-attribute-global`, `boolean_or_unspecified`, `attributeType`, `naming-convention`, `patterns` and `nonNegativeInteger_or_unbounded`.

```
<xs:simpleType name="exp-type">
 <xs:restriction base="xs:string">
     <xs:enumeration value="root"/>
     <xs:enumeration value="value"/>
     <xs:enumeration value="unspecified"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="content">
 <xs:restriction base="xs:string">
     <xs:enumeration value="value"/>
     <xs:enumeration value="ref"/>
     <xs:enumeration value="unspecified"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="exp-attribute">
 <xs:restriction base="xs:string">
     <xs:enumeration value="double-tag"/>
     <xs:enumeration value="attribute-tag"/>
     <xs:enumeration value="type-tag"/>
     <xs:enumeration value="no-tag"/>
     <xs:enumeration value="no-tag-simple"/>
     <xs:enumeration value="attribute-content"/>
     <xs:enumeration value="unspecified"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="exp-attribute-global">
     <xs:restriction base="xs:string">
         <xs:enumeration value="double-tag"/>
         <xs:enumeration value="attribute-tag"/>
         <xs:enumeration value="attribute-content"/>
     </xs:restriction>
 </xs:simpleType>

<xs:simpleType name="boolean_or_unspecified">
 <xs:union>
     <xs:simpleType>
         <xs:restriction base="xs:boolean"/>
     </xs:simpleType>
     <xs:simpleType>
         <xs:restriction base="xs:string">
             <xs:enumeration value="unspecified"/>
         </xs:restriction>
     </xs:simpleType>
 </xs:union>
</xs:simpleType>
```

 

```
<xs:simpleType name="attributeType">
 <xs:restriction base="xs:normalizedString">
     <xs:enumeration value="inverse"/>
     <xs:enumeration value="derive"/>
     <xs:enumeration value="derive-inverse"/>
 </xs:restriction>
</xs:simpleType>

 <xs:simpleType name="naming-convention">
     <xs:restriction base="xs:string">
          <xs:enumeration value="initial-upper"/>
          <xs:enumeration value="camel-case"/>
          <xs:enumeration value="preserve-case"/>
     </xs:restriction>
 </xs:simpleType>

<xs:simpleType name="patterns">
 <xs:restriction>
     <xs:simpleType>
         <xs:list itemType="xs:normalizedString" />
     </xs:simpleType>
     <xs:minLength value="1"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="nonNegativeInteger_or_unbounded">
 <xs:union>
     <xs:simpleType>
         <xs:restriction base="xs:nonNegativeInteger"/>
     </xs:simpleType>
     <xs:simpleType>
         <xs:restriction base="xs:string">
             <xs:enumeration value="unbounded"/>
         </xs:restriction>
     </xs:simpleType>
 </xs:union>
</xs:simpleType>

<xs:simpleType name="qual">
 <xs:restriction base="xs:string">
     <xs:enumeration value="qualified"/>
     <xs:enumeration value="unqualified"/>
 </xs:restriction>
</xs:simpleType>
```

## 10.5 Applicability of configuration directives

In Clauses 7.2.2.2 and 9, certain requirements depend on whether a given configuration directive *applies* to a given EXPRESS element or to a given value. This subclause defines the circumstances in which each directive applies.

NOTE Conflicts that arise in applicability are resolved as specified in 10.2.

### 10.5.1 exp-attribute

The applicability of an **exp-attribute** directive depends on its value, as specified below.

#### 10.5.1.1 attribute-content

An **exp-attribute="attribute-content"** directive applies to a given EXPRESS attribute if any of the following is true:

— the EXPRESS attribute is an entity data type, the **entity-attribute="attribute-content"** directive is specified by an **option** directive (see 10.3.1), and no **exp-attribute** directive is explicitly given for the attribute;

— the EXPRESS attribute is not an entity data type, the **concrete-attribute="attribute-content** directive is specified by an **option** directive (see 10.3.1), and no **exp-attribute** directive is explicitly given for the attribute;

— the **exp-attribute="attribute-content"**directive is specified for (all attributes of) the entity data type in which the attribute is declared, or any subtype of that entity data type that is instantiated in the entity instance (see 10.3.3);

— the **exp-attribute="attribute-content"**directive is specified for the EXPRESS attribute in question (see 10.3.4), where the scoping **entity** directive selects the entity data type in which the attribute is declared, or any subtype of that entity data type that is instantiated in the entity instance.

#### 10.5.1.2 type-tag

An **exp-attribute="type-tag"** directive applies to a given EXPRESS attribute if it is specified explicitly for that attribute in an **exp-attribute** directive (see 10.3.4), where the scoping **entity** directive selects the entity data type in which the attribute is declared, or any subtype of that entity data type that is instantiated in the entity instance (see 10.3.3).

#### 10.5.1.3 attribute-tag

An **exp-attribute="attribute-tag"** directive applies to a given EXPRESS attribute if the data type of the attribute is not an entity data type, and any of the following is true:

— **concrete-attribute="attribute-tag"** is specified by an **option** directive (see 10.3.1), or no value for **concrete-attribute** is specified, and no **exp-attribute** directive is explicitly given for the attribute;

— the directive **exp-attribute="attribute-tag"** is specified for (all attributes of) the entity data type in which the attribute is declared (see 10.3.3);

— the directive **exp-attribute="attribute-tag"** is specified explicitly for the EXPRESS attribute in question (see 10.3.4), where the scoping **entity** directive selects the entity data type in which the attribute is declared.

An **exp-attribute="attribute-tag"** applies to an EXPRESS attribute whose data type is an entity data type only when the directive **exp-attribute="attribute-tag"** is specified explicitly for the EXPRESS attribute in question (see 10.3.4), where the scoping **entity** directive selects the entity data type in which the attribute is declared.

Exception: If the inheritance mapping applies to the entity data type that is the data type of the attribute, **exp-attribute="attribute-tag"** does not apply to the attribute, even when explicitly specified for it.

NOTE The **exp-attribute="attribute-tag"** directive does not apply to aggregates.

 

#### 10.5.1.4 double-tag

An **exp-attribute="double-tag"** directive applies to a given EXPRESS attribute if any of the following is true:

— the data type of the attribute is an entity data type and the **entity-attribute="double-tag"** directive is specified by an **option** directive (see 10.3.1);

— the data type of the attribute is not an entity data type and the **concrete-attribute="double-tag"** directive is specified by an **option** directive (see 10.3.1);

— the directive **exp-attribute="double-tag"** is specified for (all attributes of) the entity data type in which the attribute is declared (see 10.3.3);

— the directive **exp-attribute="double-tag"** is specified explicitly for the EXPRESS attribute in question (see 10.3.4), where the scoping **entity** directive selects the entity data type in which the attribute is declared;

— the data type of the attribute is an entity data type and no other value of **exp-attribute** applies to that EXPRESS attribute.

#### 10.5.1.5 no-tag

An **exp-attribute="no-tag"** directive applies to a given EXPRESS attribute if all of the following are true:

— the data type of the attribute is an entity data type that has no subtypes in the context schema;

— the attribute is not declared `OPTIONAL`;

— the directive is specified explicitly for the attribute in question (see 10.3.4), where the scoping **entity** directive (see 10.3.3) selects the entity data type in which the attribute is declared.

NOTE **exp-attribute="no-tag"** applies to the entity in which the attribute is declared and to all subtypes of it. It cannot be selectively applied to subtypes of the entity data type in which the attribute is declared.

#### 10.5.1.6 no-tag-simple

An **exp-attribute="no-tag-simple"** directive applies to a given EXPRESS attribute if all of the following are true:

— the data type of the attribute is a simple data type;

— the directive is specified explicitly for the attribute in question (see 10.3.4), where the scoping **entity** directive (see 10.3.3) selects the entity data type in which the attribute is declared;

— all other attributes of the same entity, as far as they are mapped into the same complexType, are declared **exp-attribute="attribute-content".**

### 10.5.2 content and use-id

A **content** or **use-id** directive applies to a given EXPRESS attribute if any of the following is true:

— the directive is specified for (all attributes of) the entity data type in which the attribute is declared, or any subtype of that entity data type that is instantiated in the owning entity instance (see 10.3.3);

— the directive is specified for the EXPRESS attribute in question (see 10.3.4), where the scoping **entity** directive (see 10.3.3) selects the entity data type in which the attribute is declared, or any subtype of that entity data type that is instantiated in the owning entity instance.

**Use-id** shall only apply to an attribute whose data type is an entity data type. If **use-id** applies, **content= "value"** applies as well.

A **content** directive applies to the encoding of a given value if either:

— the directive applies to the attribute with that value, or

— the value is a component of an aggregate value, and the directive is specified for that aggregate (see 10.3.6).

### 10.5.3 exp-type

The **exp-type** directive applies to an EXPRESS data type if either:

— the directive is specified for all EXPRESS data types in the unit of serialization by an **option** directive (see 10.3.1),

— the directive is specified for the EXPRESS data type in question (see 10.3.2 and 10.3.3), or

— the directive is specified for some data type of which the data type is a subtype, or a specialization.

An **exp-type** directive applies to an entity instance if the directive applies to any entity data type that is instantiated in the entity instance (see 10.3.3).

An **exp-type** directive applies to a value of a non-entity data type if the directive applies to the data type of the value (see 10.3.2).

### 10.5.4 map

A **map** configuration directive (see 10.2.13) applies to an EXPRESS attribute if any of the following is true:

— the **map** directive is specified explicitly for the attribute (see 10.3.4);

— the **map** directive is specified for the data type of the attribute (see 10.3.2);

— the **map** directive is specified for some data type of which the data type of the attribute is a specialization.

A **map** configuration directive applies to a given value if any of the following is true:

— the **map** directive is specified for the data type of the value (see 10.3.2);

— the **map** directive is specified for some data type of which the data type of the value is a specialization;

— the **map** directive is specified explicitly for the place where the value is to be encoded – as the value of an attribute (see 9.4.3), or as a component of an aggregate value (see 9.8).

 

### 10.5.5 tagless

A **tagless** configuration directive (see 10.2.8) applies to an EXPRESS aggregation data type if any of the following is true:

— the **tagless** directive is specified explicitly for the data type in a **type** directive (see 10.3.2);

— the **tagless** directive is specified for some data type of which the data type is a specialization;

— the **tagless** directive is specified in an **aggregate** element (see 10.3.6) for the data type as a nested aggregation data type occurring as the base-type of another aggregation data type.

A **tagless** configuration directive applies to an EXPRESS attribute if either of the following is true:

— the **tagless** directive is specified explicitly for the attribute (see 10.2.5);

— the **tagless** directive applies to the data type of the attribute, as specified above.

Exception: The directive **tagless="true"** applies only to aggregation data types whose deepest base-type (see 8.2.2.4) is one of the following:

— EXPRESS types `BOOLEAN`, `INTEGER`, `LOGICAL`, `REAL` or `NUMBER`;

— a `STRING` data type whose values will not contain whitespace;

— a `BINARY` data type whose values will be octet-strings (multiples of 8 bits);

— a defined data type whose fundamental data type is an `ENUMERATION`;

— a named data type that is mapped, by the **map** configuration directive (see 10.2.13), to an XML simple type whose values contain no whitespace;

— a defined data type that is a specialization of any of the above;

— an entity data type that uses the inheritance-free mapping (see 8.5) and whose values are specified by the configuration directives **exp-type="root"** or **exp-type="unspecified"** to allow by-reference representation (see 9.3).

NOTE **tagless="true"** specifies that the `xs:list` form should be used to represent the aggregate value. Therefore, it only applies when the XML representations of the component have `simpleTypes` and contain no whitespace.

A **tagless** configuration directive applies to an aggregate value that is the value of an EXPRESS attribute if it applies to that EXPRESS attribute, as specified above.

### 10.5.6 flatten

A **flatten** configuration directive (see 10.2.9) applies to an EXPRESS aggregation data type if any of the following is true:

— the **flatten** directive is specified explicitly for the data type in a **type** directive (see 10.3.2);

— the **flatten** directive is specified for some data type of which the data type is a specialization;

— the **flatten** directive is specified in an **aggregate** element (see 10.3.6) for the data type as a nested aggregate occurring in the base-type of another aggregation data type.

A **flatten** configuration directive applies to an EXPRESS attribute if either of the following is true:

— the **flatten** directive is specified explicitly for the attribute (see 10.3.4);

— the **flatten** directive applies to the data type of the attribute, as specified above.

A **flatten** configuration directive applies to a value if it applies to the EXPRESS attribute of which it is the value.

### 10.5.7 inheritance

An **inheritance** configuration directive applies to a given EXPRESS entity data type if the directive is specified in a scoping **entity** directive that selects the entity data type.

An **inheritance** configuration directive applies to a given EXPRESS entity data type if the directive is specified in a scoping **option** directive, and an **inheritance** directive is not specified in a scoping **entity** directive that selects the entity data type.

### 10.5.8 notation

The **notation** configuration directive applies to a data type when that data type is selected in the declaration or that data type is a specialization of the data type selected in the declaration.

The **notation** configuration directive applies to an EXPRESS attribute when the attribute is selected in the declaration or the directive applies to the data type of that EXPRESS attribute, as specified above.

The **notation** configuration directive applies to a value if it is a value of a data type or an attribute to which the directive applies as specified above.

### 10.5.9 keep

A **keep** configuration directive (see 10.2.11) applies to an EXPRESS attribute if it is specified explicitly for that attribute (see 10.3.4).

The **keep** configuration directive applies to an EXPRESS `INVERSE` or `DERIVE`d attribute, or to an EXPRESS attribute inverted by the **invert** directive, when the **keep-all** option is specifed for the EXPRESS attribute type, and no **keep** directive is specified explicitly for the attribute.

The **keep** configuration directive applies to a data type when that data type is selected in the declaration.

The **keep** configuration directive applies to an EXPRESS entity if it is specified explicitly for that entity (see 10.3.3).

### 10.5.10 ref

The **ref** configuration directive (see 10.2.19) applies to an EXPRESS attribute if it is specified explicitly for that attribute (see 10.3.4).

### 10.5.11 use

The **use** configuration directive (see 10.2.20) applies to an EXPRESS attribute if it is specified explicitly for that attribute (see 10.3.4).

 

### 10.5.12 implementation

The **implementation** configuration directive applies to an EXPRESS entity if it is specified explicitly for that entity (see 10.3.3).

The **implementation** configuration directive (see 10.2.21) applies to an EXPRESS attribute if it is specified explicitly for that attribute (see 10.3.4).

### 10.5.13 facet

The **facet** configuration directive applies to an EXPRESS entity if it is specified explicitly for that entity (see 10.3.3).

# Annex A
(normative)

# Universal Resource Names for bindings of EXPRESS schemas

This annex specifies the form of a URN to designate a binding of an EXPRESS schema published in an International Standard, where the binding conforms to this part of ISO 10303. Such URNs shall be used in corresponding XML Namespace declarations required by this part of ISO 10303.

## A.1 URN for Base XML Schema

The document descriptive elements specified in the Base XML Schema (Annex C) shall belong to the namespace identified by the URN:

```
urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common
```

## A.2 URN for Configuration Language Schema

The document descriptive elements specified in the Configuration Language schema (Annex B) shall belong to the namespace identified by the URN:

```
urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:configuration_language
```

## A.3 URN for Document Schema

The document descriptive elements specified in the Document Schema (Annex D) shall belong to the namespace identified by the URN:

```
urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:document
```

## A.4 URNs for EXPRESS schemas with registered object-identifiers

For an EXPRESS schema that has a URN specified in an international standard, the specified URN shall be used.

For an EXPRESS schema that has no specified URN, but has an existing ASN.1 object-identifier, the corresponding URN shall be that specified in IETF RFC 3061 (which specifies the URN that corresponds to an ASN.1 object-identifier).

NOTE For those EXPRESS schema whose ASN.1 object-identifiers are registered in ISO standards, the following structure applies: iso(1) standard(0) <number> [ part(<part>) } version(<version>) followed by other identifier elements defined in the standard. And the corresponding URN is: urn:oid:1.0.<number>[.<part>].<version>.<other elements>

EXAMPLE

The URN for the XML namespace corresponding to the geometry EXPRESS schema registered in ISO 10303-42:2000 as: object(1) geometry(6) would be: urn:oid:1.0.10303.42.2.1.6

# Annex B
(normative)

# XML Schema for the configuration language

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:cnf=
        "urn:iso:std:iso:10303:-28:ed-
                2:tech:XMLschema:configuration_language"
    targetNamespace=
        "urn:iso:std:iso:10303:-28:ed-
                2:tech:XMLschema:configuration_language"
    elementFormDefault="unqualified" attributeFormDefault="unqualified">


    <xs:element name="configuration" nillable="true">
        <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="include">
                    <xs:complexType>
                        <xs:attribute name="configuration" type="xs:IDREF"
                                use="required"/>
                    </xs:complexType>
                </xs:element>
                <xs:element ref="cnf:option"/>
                <xs:element ref="cnf:type"/>
                <xs:element ref="cnf:entity"/>
                <xs:element ref="cnf:schema" maxOccurs="1"/>
                <xs:element ref="cnf:uosElement"/>
                <xs:element ref="cnf:uosEntity"/>
                <xs:element ref="cnf:rootEntity"/>
            </xs:choice>
            <xs:attribute name="id" type="xs:ID" use="required"/>
            <xs:attribute name="targetNamespace" type="xs:anyURI"
use="optional"/>
            <xs:attribute name="schema" type="xs:IDREF" use="optional"/>
            <xs:attribute name="configuration-location"
                type="cnf:Seq-anyURI" use="optional"/>
        </xs:complexType>
    </xs:element>

    <xs:element name="option">
        <xs:complexType>
            <xs:attribute name="inheritance" type="xs:boolean"
                default="false"/>
            <xs:attribute name="exp-type" type="cnf:exp-type"
                default="unspecified"/>
            <xs:attribute name="entity-attribute"
                    type="cnf:exp-attribute-global" default="double-tag"/>
            <xs:attribute name="concrete-attribute"
                    type="cnf:exp-attribute-global" default="attribute-tag"/>
            <xs:attribute name="tagless" type="cnf:boolean_or_unspecified"
                default="unspecified"/>
            <xs:attribute name="naming-convention"
                type="cnf:naming-convention" default="initial-upper"/>
            <xs:attribute name="keep-all" type="cnf:attributeType"/>
```

```
        <xs:attribute name="generate-keys" type="xs:boolean"
            default="true"/>
    </xs:complexType>
</xs:element>

<xs:element name="type">
    <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="1">
         <xs:element ref="cnf:aggregate"/>
        </xs:choice>
        <xs:attribute name="select" type="cnf:patterns" use="required"/>
        <xs:attribute name="name" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="map" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="exp-type" type="cnf:exp-type"
        use="optional"/>
        <xs:attribute name="tagless" type="cnf:boolean_or_unspecified"
          use="optional"/>
        <xs:attribute name="notation" type="xs:normalizedString"
          use="optional"/>
        <xs:attribute name="keep" type="xs:boolean" default="true"/>
        <xs:attribute name="flatten" type="xs:boolean" use="optional"/>
    </xs:complexType>
</xs:element>

<xs:element name="entity">
    <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element ref="cnf:attribute"/>
         <xs:element ref="cnf:inverse"/>
        </xs:choice>
        <xs:attribute name="select" type="cnf:patterns" use="optional"/>
        <xs:attribute name="synthetic" use="optional" >
          <xs:simpleType>
            <xs:restriction>
              <xs:simpleType>
                 <xs:list itemType="xs:Name"/>
              </xs:simpleType>
              <xs:minLength value="2"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="inheritance" type="xs:boolean"
          default="false"/>
        <xs:attribute name="name" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="tag-source" type="xs:NMTOKEN"
          use="optional"/>
        <xs:attribute name="tag-values" type="xs:NMTOKENS"
          use="optional"/>
        <xs:attribute name="map" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="exp-type" type="cnf:exp-type"
          use="optional"/>
        <xs:attribute name="content" type="cnf:content" use="optional"/>
        <xs:attribute name="exp-attribute" type="cnf:exp-attribute"
          default="unspecified"/>
        <xs:attribute name="tagless" type="cnf:boolean_or_unspecified"
          use="optional"/>
```

```
        <xs:attribute name="keep" type="xs:boolean"  default="true"/>
        <xs:attribute name="new" type="xs:boolean"
          default="false"/>
        <xs:attribute name="implementation" type="xs:string"
          use="optional"/>
        <xs:attribute name="facet" type="xs:string" use="optional"/>
    </xs:complexType>
</xs:element>

<xs:element name="attribute">
    <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element ref="cnf:aggregate"/>
        </xs:choice>
        <xs:attribute name="select" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="new" type="xs:boolean" default="false"/>
        <xs:attribute name="name" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="exp-attribute" type="cnf:exp-attribute"
           use="optional"/>
        <xs:attribute name="content" type="cnf:content" use="optional"/>
        <xs:attribute name="aggregate-content" type="cnf:content"
    use="optional"/>
        <xs:attribute name="map" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="use-id" type="xs:boolean" use="optional"/>
        <xs:attribute name="keep" type="xs:boolean" default="true"/>
        <xs:attribute name="tagless" type="cnf:boolean_or_unspecified"
           use="optional"/>
        <xs:attribute name="notation" type="xs:normalizedString"
           use="optional"/>
        <xs:attribute name="flatten" type="xs:boolean" use="optional"/>
        <xs:attribute name="ref" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="use" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="implementation" type="xs:string"
           use="optional"/>
    </xs:complexType>
</xs:element>

<xs:element name="inverse">
    <xs:complexType>
        <xs:attribute name="select" type="xs:NMTOKEN"
            use="optional"/>
        <xs:attribute name="name" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="exp-attribute" type="cnf:exp-attribute"
            use="optional"/>
        <xs:attribute name="content" type="cnf:content" use="optional"/>
        <xs:attribute name="tagless" type="cnf:boolean_or_unspecified"
            use="optional"/>
        <xs:attribute name="invert" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="map" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="keep" type="xs:boolean" default="false"/>
        <xs:attribute name="minOccurs" type="xs:nonNegativeInteger"
            default="0" use="optional"/>
        <xs:attribute name="maxOccurs"
            type="cnf:nonNegativeInteger_or_unbounded"
            default="unbounded"  use="optional"/>
        <xs:attribute name="ref" type="xs:NMTOKEN" use="optional" />
    </xs:complexType>
</xs:element>

<xs:element name="aggregate">
    <xs:complexType>
```

```
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="cnf:aggregate"/>
        </xs:choice>
        <xs:attribute name="name" type="xs:NMTOKEN" use="optional"/>
        <xs:attribute name="content" type="cnf:content"
            use="optional"/>
        <xs:attribute name="tagless" type="cnf:boolean_or_unspecified"
            use="optional"/>
        <xs:attribute name="use-id" type="xs:boolean" use="optional"/>
        <xs:attribute name="flatten" type="xs:boolean" use="optional"/>
    </xs:complexType>
</xs:element>

<xs:element name="schema">
    <xs:complexType>
        <xs:choice maxOccurs="unbounded">
            <xs:element ref="cnf:namespace" minOccurs="0"
                   maxOccurs="unbounded"/>
            <xs:element ref="cnf:include" minOccurs="0"
                   maxOccurs="unbounded"/>
            <xs:element ref="cnf:containerObject" minOccurs="0"
                   maxOccurs="1"/>
            <xs:element ref="cnf:additionalObject" minOccurs="0"
                   maxOccurs="unbounded"/>
        </xs:choice>
        <xs:attribute name="targetNamespace" type="xs:anyURI" />
        <xs:attribute name="elementFormDefault" type="cnf:qual" />
        <xs:attribute name="attributeFormDefault"  type="cnf:qual" />
        <xs:attribute name="defaultRootObjectType" type="xs:string" />
        <xs:attribute name="defaultObjectType" type="xs:string" />
        <xs:attribute name="schema-version"    type="xs:string" />
        <xs:attribute name="embed-schema-items" type="xs:boolean"
            default="false"/>
    </xs:complexType>
</xs:element>

<xs:element name="containerObject">
    <xs:complexType>
        <xs:attribute name="name"type="xs:NMTOKEN" use="required"/>
        <xs:attribute  name="implementation" type="xs:string"
                   use="optional"/>
    </xs:complexType>
</xs:element>

<xs:element name="additionalObject">
    <xs:complexType>
        <xs:attribute name="name"type="xs:NMTOKEN" use="required"/>
        <xs:attribute  name="implementation" type="xs:string"
                   use="optional"/>
    </xs:complexType>
</xs:element>

<xs:element name="namespace">
    <xs:complexType>
        <xs:attribute name="alias"   type="xs:anyURI"/>
        <xs:attribute  name="prefix" type="xs:string"/>
    </xs:complexType>
</xs:element>

<xs:element name="include">
    <xs:complexType>
```

 

```
            <xs:attribute name="urn" type="xs:anyURI" />
            <xs:attribute  name="schema-location" type="xs:string" />
        </xs:complexType>
    </xs:element>

<xs:element name="addAttribute" >
   <xs:complexType>
     <xs:attribute name="name" type="xs:QName" use="required" />
     <xs:attribute name="ref" type="xs:boolean" default="false" />
     <xs:attribute name="type" type="xs:QName" use="optional" />
     <xs:attribute name="usage" default="optional" >
       <xs:simpleType>
         <xs:restriction base="xs:string">
           <xs:enumeration value="optional" />
           <xs:enumeration value="required" />
           <xs:enumeration value="fixed" />
           <xs:enumeration value="default" />
         </xs:restriction>
       </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="value" type="xs:string" use="optional" />
   </xs:complexType>
</xs:element>

    <xs:simpleType name="exp-type">
        <xs:restriction base="xs:string">
            <xs:enumeration value="root"/>
            <xs:enumeration value="value"/>
            <xs:enumeration value="unspecified"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="exp-attribute">
        <xs:restriction base="xs:string">
            <xs:enumeration value="double-tag"/>
            <xs:enumeration value="attribute-tag"/>
            <xs:enumeration value="type-tag"/>
            <xs:enumeration value="no-tag"/>
            <xs:enumeration value="no-tag-simple"/>
            <xs:enumeration value="attribute-content"/>
            <xs:enumeration value="unspecified"/>
        </xs:restriction>
    </xs:simpleType>

<xs:simpleType name="exp-attribute-global">
        <xs:restriction base="xs:string">
            <xs:enumeration value="double-tag"/>
            <xs:enumeration value="attribute-tag"/>
            <xs:enumeration value="attribute-content"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="content">
        <xs:restriction base="xs:string">
            <xs:enumeration value="value"/>
            <xs:enumeration value="ref"/>
            <xs:enumeration value="unspecified"/>
        </xs:restriction>
    </xs:simpleType>
```

```
    <xs:simpleType name="naming-convention">
        <xs:restriction base="xs:string">
             <xs:enumeration value="initial-upper"/>
             <xs:enumeration value="camel-case"/>
             <xs:enumeration value="preserve-case"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="boolean_or_unspecified">
        <xs:union>
            <xs:simpleType>
             <xs:restriction base="xs:boolean"/>
            </xs:simpleType>
            <xs:simpleType>
             <xs:restriction base="xs:string">
               <xs:enumeration value="unspecified"/>
             </xs:restriction>
            </xs:simpleType>
        </xs:union>
    </xs:simpleType>

<xs:simpleType name="attributeType">
    <xs:restriction base="xs:normalizedString">
        <xs:enumeration value="inverse"/>
        <xs:enumeration value="derive"/>
        <xs:enumeration value="derive-inverse"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="nonNegativeInteger_or_unbounded">
    <xs:union>
        <xs:simpleType>
            <xs:restriction base="xs:nonNegativeInteger"/>
        </xs:simpleType>
        <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="unbounded"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:union>
</xs:simpleType>

<xs:simpleType name="patterns">
    <xs:restriction>
        <xs:simpleType>
            <xs:list itemType="xs:normalizedString" />
        </xs:simpleType>
        <xs:minLength value="1"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="qual">
    <xs:restriction base="xs:string">
        <xs:enumeration value="qualified"/>
        <xs:enumeration value="unqualified"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="Seq-anyURI">
    <xs:list itemType="xs:anyURI" />
</xs:simpleType>
```

 

```
<xs:element name="uosElement">
   <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element ref="cnf:addAttribute"/>
     </xs:choice>
     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
   </xs:complexType>
</xs:element>

<xs:element name="uosEntity" >
   <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element ref="cnf:addAttribute"/>
     </xs:choice>
     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
   </xs:complexType>
</xs:element>

<xs:element name="rootEntity">
   <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element ref="cnf:addAttribute"/>
     </xs:choice>
     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
   </xs:complexType>
</xs:element>

</xs:schema>
```

# Annex C
(normative)

# Base XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace=
   "urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"
  xmlns:exp=
   "urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">


    <!-- ROOT ELEMENT  UOS -->
<xs:complexType name="uos">
    <xs:sequence>
        <xs:element ref="exp:header" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID"  use="optional"/>
    <xs:attribute name="express" type="exp:Seq-anyURI"
     use="optional"/>
    <xs:attribute name="configuration" type="exp:Seq-anyURI"
     use="optional"/>
    <xs:attribute name="schemaLocation"  type="exp:Seq-anyURI"
     use="optional"/>
    <xs:attribute name="edo"  type="xs:anyURI" use="optional"/>
    <xs:attribute name="defaultLanguage" type="xs:language"
      use="optional"/>
</xs:complexType>

<xs:element name="uos" type="exp:uos">
    <xs:key name="base64Binary-key">
        <xs:selector xpath="exp:base64Binary-wrapper"/>
        <xs:field xpath="@id"/>
    </xs:key>
    <xs:key name="hexBinary-key">
        <xs:selector xpath="exp:hexBinary-wrapper"/>
        <xs:field xpath="@id"/>
    </xs:key>
    <xs:key name="logical-key">
        <xs:selector xpath="exp:logical-wrapper"/>
        <xs:field xpath="@id"/>
    </xs:key>
    <xs:key name="integer-key">
        <xs:selector xpath="exp:integer-wrapper"/>
        <xs:field xpath="@id"/>
    </xs:key>
    <xs:key name="boolean-key">
        <xs:selector xpath="exp:boolean-wrapper"/>
        <xs:field xpath="@id"/>
    </xs:key>
    <xs:key name="double-key">
        <xs:selector xpath="exp:double-wrapper"/>
        <xs:field xpath="@id"/>
    </xs:key>
    <xs:key name="decimal-key">
        <xs:selector xpath="exp:decimal-wrapper"/>
```

```
        <xs:field xpath="@id"/>
    </xs:key>
    <xs:key name="long-key">
        <xs:selector xpath="exp:long-wrapper"/>
        <xs:field xpath="@id"/>
    </xs:key>
    <xs:key name="string-key">
        <xs:selector xpath="exp:string-wrapper"/>
        <xs:field xpath="@id"/>
    </xs:key>
</xs:element>

    <!-- UOS and DOCUMENT HEADER -->
<xs:complexType name="name_and_address">
  <xs:sequence>
    <xs:element name="name" type="xs:string" />
    <xs:element name="address">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="address_line" type="xs:string"
              minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:element name = "header">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="name" type="xs:string" minOccurs="0"/>
   <xs:element name="time_stamp" type="xs:dateTime" minOccurs="0"/>
   <xs:element name="author" type="exp:name_and_address" minOccurs="0"/>
   <xs:element name="organization" type="exp:name_and_address"
       minOccurs="0"/>
   <xs:element name="preprocessor_version" type="xs:string" minOccurs="0"/>
   <xs:element name="originating_system" type="xs:string" minOccurs="0"/>
   <xs:element name="authorization" type="xs:string" minOccurs="0"/>
   <xs:element name="documentation" type="xs:string" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:attribute name="authority" type="xs:anyURI"/>

<xs:complexType name = "edokey" abstract="true">
       <xs:attribute name="id" type="xs:ID" use="required" />
       <xs:attribute name="edo" type="xs:anyURI" use="optional"/>
       <xs:attribute ref = "exp:authority"  use = "required"/>
</xs:complexType>

<xs:attribute name="edokeyType" type="xs:QName" />

<xs:element name="edokey" type="exp:edokey" abstract="true" />

<!--  EXP ENTITY DEFINTIONS and DECLARATIONS -->

<xs:complexType name = "Entity" abstract="true">
       <xs:annotation>
          <xs:documentation>The Entity type is the supertype for every
         EXPRESS entity. It provides the exp id attribute that is the local
```

```
          identifier for each instance as well as the other exp descriptive
          attributes.</xs:documentation>
        </xs:annotation>
 <xs:attribute name = "href" type = "xs:anyURI" use = "optional"/>
 <xs:attribute name = "ref" type = "xs:IDREF" use = "optional"/>
 <xs:attribute name = "proxy" type = "xs:IDREF" use = "optional"/>
 <xs:attribute name = "edo" type = "xs:anyURI" use = "optional"/>
 <xs:attributeGroup ref = "exp:instanceAttributes"/>
</xs:complexType>

<xs:element name="Entity" type = "exp:Entity" abstract = "true" nillable =
"true"/>

<xs:element name="complexEntity" substitutionGroup="exp:Entity" nillable =
"true">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="exp:Entity">
                <xs:sequence>
                    <xs:any namespace="##any"
                        processContents="skip" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="entities" type="xs:NMTOKENS"
                    use="optional"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

<xs:element name="Single-Entity" type="exp:Single-Entity"
           abstract="true"/>

<xs:complexType name="Single-Entity">
</xs:complexType>

<!-- UNRESTRICTED AGGREGATION TYPE DEFINITIONS -->

<xs:complexType name="Seq-IDREF">
    <xs:simpleContent>
        <xs:extension base="xs:IDREFS">
            <xs:attribute name="ref" type="xs:IDREF" use="optional"/>
            <xs:attribute ref="exp:itemType"  use="optional" />
            <xs:attribute ref="exp:cType"     use="optional" />
            <xs:attribute ref="exp:arraySize" use="optional"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="Seq-IDREF-wrapper">
    <xs:simpleContent>
        <xs:extension base="exp:Seq-IDREF">
            <xs:attributeGroup ref="exp:instanceAttributes"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

    <!-- GLOBAL ATTRIBUTE DECLARATIONS  -->

    <!--  EXP ARRAY -->
<xs:attribute name="arraySize">
        <xs:simpleType>
```

 

```
     <xs:restriction>
            <xs:simpleType>
              <xs:list itemType="xs:integer"/>
            </xs:simpleType>
            <xs:minLength value="1"/>
     </xs:restriction>
       </xs:simpleType>
</xs:attribute>

<xs:attribute name="itemType">
    <xs:simpleType>
      <xs:list itemType="xs:QName"/>
    </xs:simpleType>
</xs:attribute>

<xs:attribute name="cType">
    <xs:simpleType>
      <xs:list itemType="exp:aggregateType"/>
    </xs:simpleType>
</xs:attribute>

    <xs:simpleType name="aggregateType">
        <xs:restriction base="xs:normalizedString">
            <xs:enumeration value="array"/>
            <xs:enumeration value="list"/>
            <xs:enumeration value="set"/>
            <xs:enumeration value="bag"/>
     <xs:enumeration value="array-unique"/>
            <xs:enumeration value="array-optional"/>
            <xs:enumeration value="array-optional-unique"/>
     <xs:enumeration value="list-unique"/>
        </xs:restriction>
    </xs:simpleType>

<xs:attributeGroup name="instanceAttributes">
    <xs:attribute name="id"  type="xs:ID" use="optional"/>
    <xs:attribute name="path" type="xs:NMTOKENS" use="optional"/>
    <xs:attribute name="pos" use="optional">
        <xs:simpleType>
            <xs:restriction>
                <xs:simpleType>
                       <xs:list itemType="xs:integer"/>
                </xs:simpleType>
                <xs:minLength value="1"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
</xs:attributeGroup>

<xs:attribute name="attributeType">
    <xs:simpleType>
        <xs:restriction base="xs:normalizedString">
            <xs:enumeration value="explicit"/>
            <xs:enumeration value="inverse"/>
            <xs:enumeration value="derived"/>
            <xs:enumeration value="renamed"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
```

```
<!--  SIMPLE TYPE DEFINITIONS -->

    <xs:complexType name="base64Binary">
        <xs:simpleContent>
            <xs:extension base="xs:base64Binary" >
                <xs:attribute name="extraBits" type="xs:integer"
use="optional"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>

<xs:complexType name="hexBinary">
    <xs:simpleContent>
        <xs:extension base="xs:hexBinary" >
                <xs:attribute name="extraBits" type="xs:integer"
use="optional"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<xs:simpleType name = "logical">
    <xs:restriction base = "xs:string">
        <xs:enumeration value = "false"/>
        <xs:enumeration value = "true"/>
        <xs:enumeration value = "unknown"/>
    </xs:restriction>
</xs:simpleType>


<!-- OTHER NEEDED TYPE DEFINITIONS -->

<xs:simpleType name="Seq-anyURI">
    <xs:list itemType="xs:anyURI" />
</xs:simpleType>


        <!-- WRAPPER FOR ATOMIC SIMPLE TYPES  -->
<xs:element name="base64Binary-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="exp:base64Binary">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="hexBinary-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="exp:hexBinary">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="integer-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:integer">
```

 

```
            <xs:attributeGroup ref="exp:instanceAttributes"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
</xs:element>

<xs:element name="logical-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="exp:logical">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>


        <!-- WRAPPER FOR ATOMIC SIMPLE TYPES  -->
<xs:element name="boolean-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:boolean">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="double-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:double">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="decimal-wrapper" nillable="true">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:decimal">
                    <xs:attributeGroup ref="exp:instanceAttributes"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
</xs:element>

<xs:element name="long-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:long">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="string-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
```

```
            <xs:extension base="xs:normalizedString">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="generalString-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="language-wrapper" nillable="true" >
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:language">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="Name-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:Name">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="QName-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:QName">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="NMTOKEN-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:NMTOKEN">
                <xs:attributeGroup ref="exp:instanceAttributes"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="anyURI-wrapper" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:anyURI">
```

 

```
            <xs:attributeGroup ref="exp:instanceAttributes"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>


</xs:schema>
```

# Annex D
(normative)

# Document Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:doc=
        "urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:document"
    targetNamespace=
        "urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:document"
        xmlns:cnf="urn:iso:std:iso:10303:-28:ed-
2:tech:XMLschema:configuration_language"
        xmlns:exp="urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"
    elementFormDefault="unqualified" attributeFormDefault="unqualified">

<!--The value for exp.xsd below provides the location of the Base XML
Schema (see Annex C). The value for cnf.xsd below provides the location of
the configuration language XML schema (see Annex B).-->

<xs:import schemaLocation="cnf.xsd" namespace="urn:iso:std:iso:10303:-
28:ed-2:tech:XMLschema:configuration_language"/>
<xs:import schemaLocation="exp.xsd" namespace="urn:iso:std:iso:10303:-
28:ed-2:tech:XMLschema:common"/>

<xs:element name="iso_10303_28">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="exp:header" minOccurs="0"/>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                   <xs:element ref="doc:schema_population"/>
                   <xs:element ref="exp:uos"/>
                   <xs:element ref="doc:express"/>
                   <xs:element ref="doc:schema"/>
                   <xs:element ref="cnf:configuration"/>
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="version" type="xs:string" use="required"/>
        <xs:attribute name="edo" type="xs:anyURI" use="optional"/>
    </xs:complexType>
</xs:element>

<xs:element name="schema_population">
 <xs:complexType>
  <xs:attribute name="governing_schema" type="xs:IDREF" use="required"/>
  <xs:attribute name="governed_sections" type="xs:IDREFS"
           use="optional"/>
  <xs:attribute name="determination_method"
           type="xs:normalizedString" default="section_boundary"/>
 </xs:complexType>
</xs:element>

<xs:element name="express" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
```

 

```
        <xs:attribute name="schemaLocation" type="exp:Seq-anyURI"
          use="optional"/>
        <xs:attribute name="id" type="xs:ID" use="required"/>
        <xs:attribute name="schema_identifier"
          type="xs:normalizedString" use="optional"/>
        <xs:attribute name="schema_name" type="xs:normalizedString"
          use="optional"/>
        <xs:attribute name="schema_version" type="xs:normalizedString"
          use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<!-- EXPRESS MODEL SCHEMA -->

<!-- This set of declarations defines a model of the EXPRESS language
     as specified by ISO 10303-11.2:2004.
-->

<!-- This specification was developed as a DTD by Eurostep
     for the STEP Modules Repository Project,
     and is here incorporated by permission.
-->


<!--                    data type                                    -->
<!-- represents the structure of an EXPRESS data type referenece -->
<xs:group name="datatype" >
  <xs:sequence>
    <xs:element ref="doc:aggregate" minOccurs="0" maxOccurs="unbounded"/>
    <xs:choice>
      <xs:element ref="doc:typename" />
      <xs:element ref="doc:builtintype" />
    </xs:choice>
  </xs:sequence>
</xs:group>


<!--                    identifiers                                  -->
<!-- identifier = an EXPRESS identifier =the name of a named element -->
<!-- identifiers = a list of EXPRESS identifiers                     -->
<xs:simpleType name="identifier">
  <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:simpleType name="identifiers">
  <xs:list itemType="doc:identifier" />
</xs:simpleType>


<!--                    express.text                                 -->
<!-- represents a body of unparsed EXPRESS text                      -->
<xs:simpleType name="express.text">
  <xs:restriction base="xs:string"/>
</xs:simpleType>


<!--                    description                                -->
<!-- The original DTD version of this used an XML parameter entity -->
<!-- to allow the structure of the description element to be       -->
```

```
<!-- externally specified.                                          -->
<xs:simpleType name="description.content">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:element name="description" type="doc:description.content"/>


<!--                 graphic.element                                -->
<!-- represents an EXPRESS-G reference symbol                       -->
<xs:element name="graphic.element">
  <xs:complexType>
    <xs:attribute name="image" type="xs:string" use="optional" />
    <xs:attribute name="page" type="xs:string" use="required" />
    <xs:attribute name="xcoord" type="xs:string" use="optional" />
    <xs:attribute name="ycoord" type="xs:string" use="optional" />
  </xs:complexType>
</xs:element>


<!--                     express_model                              -->
<!-- This is the document element, containing one or more schemas. -->
<xs:element name="express_model" >
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:element ref="doc:application" minOccurs="0"/>
      <xs:element ref="doc:schema"
          minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="language_version" default="1">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="1" />
          <xs:enumeration value="2" />
          <xs:enumeration value="3" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="reference" type="xs:string" use="optional" />
<!--    reference is the source of the schema, for example ISO 10303-41
-->
    <xs:attribute name="description.file" type="xs:string" use="optional"
/>
<!-- The rcs attributes are keywords for RCS source control systems     -->
<!--    rcs.date      $Date: 2003/07/18 21:50:20 $                      -->
<!--    rcs.revision  $Revision: 1.14 $                                 -->
    <xs:attribute name="rcs.date" type="xs:string" use="required" />
    <xs:attribute name="rcs.revision" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>

<xs:element name="application">
  <xs:complexType>
    <xs:attribute name="name"  type="xs:string"   use="optional" />
    <xs:attribute name="owner" type="xs:string"   use="optional" />
    <xs:attribute name="url"   type="xs:string"   use="optional" />
    <xs:attribute name="version" type="xs:string" use="optional" />
    <xs:attribute name="source"  type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
```

 

```
<!--                 EXPRESS language elements                      -->

<!--                 SCHEMA declaration                             -->
<xs:element name="schema">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:element ref="doc:interface" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="doc:constant" minOccurs="0" maxOccurs="unbounded"/>
      <xs:choice  minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="doc:entity"/>
        <xs:element ref="doc:function"/>
        <xs:element ref="doc:procedure"/>
        <xs:element ref="doc:rule"/>
        <xs:element ref="doc:subtype.constraint"/>
        <xs:element ref="doc:type"/>
      </xs:choice>
      <xs:element ref="doc:graphic.element" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required"/>
    <xs:attribute name="schemaLocation" type="exp:Seq-anyURI"
        use="optional"/>
    <xs:attribute name="name" type="doc:identifier"  use="optional"/>
    <xs:attribute name="version" type="doc:identifier"  use="optional"/>
    <xs:attribute name="schemaIdentifier"
        type="xs:normalizedString" use="optional"/>
  </xs:complexType>
</xs:element>


<!--              interface (USE and REFERENCE)                     -->
<xs:element name="interface">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:interfaced.item"
         minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="doc:described.item"
         minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="kind" default="USE">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="USE"/>
          <xs:enumeration value="REFERENCE"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="schema" type="doc:identifier" use="required" />
  </xs:complexType>
</xs:element>

<!-- interfaced.item
    REFERENCE FROM schema (interfaced.item .name AS .alias)
-->
<xs:element name="interfaced.item">
  <xs:complexType>
    <xs:attribute name="name" type="doc:identifier" use="required" />
    <xs:attribute name="alias" type="doc:identifier" use="optional" />
  </xs:complexType>
</xs:element>
```

```
<!-- described.item
  A description of an interfaced.item.
  The described.item specifies what kind of language element the
  interfaced.item is, and may include other documentation.

  item is the EXPRESS identifier for the interfaced.item
  attribute is used to describe attributes of an interfaced ENTITY
    item is the name of the entity
    attribute is the name of the attribute
 -->
<xs:element name="described.item">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="item" type="doc:identifier" use="required" />
    <xs:attribute name="attribute" type="doc:identifier" use="optional" />
    <xs:attribute name="kind" default="ENTITY">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="ENTITY"/>
          <xs:enumeration value="TYPE"/>
          <xs:enumeration value="ATTRIBUTE"/>
          <xs:enumeration value="FUNCTION"/>
          <xs:enumeration value="PROCEDURE"/>
          <xs:enumeration value="CONSTANT"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>


<!--              CONSTANT declaration                      -->
<xs:element name="constant">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:group ref="doc:datatype"/>
    </xs:sequence>
    <xs:attribute name="name" type="doc:identifier" use="required" />
    <xs:attribute name="expression" type="doc:express.text" use="required"
/>
  </xs:complexType>
</xs:element>


<!--              TYPE declaration                         -->
<xs:element name="type">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:element ref="doc:aggregate" minOccurs="0" maxOccurs="unbounded"/>
      <xs:choice>
        <xs:element ref="doc:typename"/>
        <xs:element ref="doc:builtintype"/>
        <xs:element ref="doc:enumeration"/>
        <xs:element ref="doc:select"/>
      </xs:choice>
      <xs:element ref="doc:where" minOccurs="0" maxOccurs="unbounded"/>
```

 

```
      <xs:element ref="doc:graphic.element" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="name" type="doc:identifier" use="required" />
  </xs:complexType>
</xs:element>


<!-- typename = reference to a defined data type by name -->
<xs:element name="typename">
  <xs:complexType>
    <xs:attribute name="name" type="doc:identifier" use="required" />
  </xs:complexType>
</xs:element>


<!-- builtintype = a data type identified by an EXPRESS keyword -->
<xs:element name="builtintype">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:graphic.element" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="type"  use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="BINARY"/>
          <xs:enumeration value="BOOLEAN"/>
          <xs:enumeration value="GENERIC"/>
          <xs:enumeration value="GENERICENTITY"/>
          <xs:enumeration value="INTEGER"/>
          <xs:enumeration value="LOGICAL"/>
          <xs:enumeration value="NUMBER"/>
          <xs:enumeration value="REAL"/>
          <xs:enumeration value="STRING"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="width" type="doc:express.text" use="optional" />
<!--      Width specification for BINARY or STRING -->
    <xs:attribute name="fixed" type="xs:boolean" default="false" />
<!--      Fixed specification for BINARY or STRING -->
    <xs:attribute name="precision" type="doc:express.text" use="optional"
/>
<!--      Precision specification for REAL -->
    <xs:attribute name="typelabel" type="doc:identifier" use="optional" />
<!--      Type label on GENERIC or GENERIC_ENTITY -->
  </xs:complexType>
</xs:element>


<!-- aggregate = an aggregation type constructor   -->
<xs:element name="aggregate">
  <xs:complexType>
    <xs:attribute name="type" default="SET" >
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="AGGREGATE"/>
          <xs:enumeration value="ARRAY"/>
          <xs:enumeration value="BAG"/>
          <xs:enumeration value="LIST"/>
          <xs:enumeration value="SET"/>
        </xs:restriction>
```

```
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="optional" type="xs:boolean" default="false" />
<!--      OPTIONAL specification for ARRAY -->
    <xs:attribute name="unique" type="xs:boolean" default="false" />
<!--      UNIQUE specification for ARRAY or LIST or AGGREGATE  -->
    <xs:attribute name="lower" type="doc:express.text" use="optional" />
<!--      lower-bound specification, or LoIndex for ARRAY -->
    <xs:attribute name="upper" type="doc:express.text" use="optional" />
<!--      upper-bound specification, or HiIndex for ARRAY -->
    <xs:attribute name="typelabel" type="doc:identifier" use="optional" />
<!--      Type label on AGGREGATE -->
  </xs:complexType>
</xs:element>


<!-- select = a select type constructor   -->
<xs:element name="select">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:graphic.element" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="extensible"    type="xs:boolean" default="false" />
    <xs:attribute name="genericentity" type="xs:boolean" default="false" />
    <xs:attribute name="basedon"     type="doc:identifier" use="optional"
/>
    <xs:attribute name="selectitems" type="doc:identifiers" use="optional"
/>
  </xs:complexType>
</xs:element>


<!-- enumeration = an enumeration type constructor   -->
<xs:element name="enumeration">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:graphic.element" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="extensible" type="xs:boolean" default="false" />
    <xs:attribute name="basedon"    type="doc:identifier" use="optional" />
    <xs:attribute name="items"    type="doc:identifiers" use="optional" />
  </xs:complexType>
</xs:element>


<!--              ENTITY declaration                        -->
<xs:element name="entity">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:element ref="doc:explicit" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="doc:derived" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="doc:inverse" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="doc:unique" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="doc:where" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="doc:graphic.element" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="name" type="doc:identifier" use="required" />
    <xs:attribute name="abstract.entity" type="xs:boolean" default="false"
/>
    <xs:attribute name="abstract.supertype" type="xs:boolean"
```

```
        default="false" />
    <xs:attribute name="supertypes" type="doc:identifiers" use="optional"
/>
    <xs:attribute name="super.expression" type="doc:express.text"
        use="optional" />
  </xs:complexType>
</xs:element>


<!-- explicit = an explicit attribute declaration   -->
<xs:element name="explicit">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:group ref="doc:datatype"/>
      <xs:element ref="doc:redeclaration" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="name" type="doc:identifier" use="required" />
    <xs:attribute name="optional" type="xs:boolean" default="false" />
  </xs:complexType>
</xs:element>


<!-- derived = a derived attribute declaration   -->
<xs:element name="derived">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:group ref="doc:datatype"/>
      <xs:element ref="doc:redeclaration" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="name" type="doc:identifier" use="required" />
    <xs:attribute name="expression" type="doc:express.text" use="required"
/>
  </xs:complexType>
</xs:element>


<!-- inverse = an inverse attribute declaration   -->
<xs:element name="inverse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:element ref="doc:inverse.aggregate" minOccurs="0"/>
      <xs:element ref="doc:redeclaration" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="name" type="doc:identifier" use="required" />
    <xs:attribute name="entity" type="doc:identifier" use="required" />
    <xs:attribute name="attribute" type="doc:identifier" use="required" />
  </xs:complexType>
</xs:element>

<!-- inverse.aggregate = multiplicity specification for inverse attribute
     type is SET or BAG
     lower is lower-bound specification
     upper is upper-bound specification
  -->
<xs:element name="inverse.aggregate">
  <xs:complexType>
    <xs:attribute name="type" default="SET" >
      <xs:simpleType>
```

```
        <xs:restriction base="xs:string">
          <xs:enumeration value="BAG"/>
          <xs:enumeration value="SET"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="lower" type="doc:express.text" use="optional" />
    <xs:attribute name="upper" type="doc:express.text" use="optional" />
  </xs:complexType>
</xs:element>

<!-- redeclaration = the attribute being redeclared by this attribute -->
<xs:element name="redeclaration">
  <xs:complexType>
    <xs:attribute name="entity-ref" type="doc:identifier" use="required" />
    <xs:attribute name="old_name" type="doc:identifier" use="optional" />
  </xs:complexType>
</xs:element>


<!-- where = a rule declaration                                    -->
<!-- used for both local rules and global rules   -->
<xs:element name="where">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="label" type="doc:identifier" use="optional" />
    <xs:attribute name="expression" type="doc:express.text" use="optional"
/>
  </xs:complexType>
</xs:element>


<!-- unique = a UNIQUE rule declaration   -->
<xs:element name="unique">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:element ref="doc:unique.attribute"
          minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="label" type="doc:identifier" use="optional" />
  </xs:complexType>
</xs:element>

<!-- unique.attribute - a key attribute in a UNIQUE rule -->
<xs:element name="unique.attribute">
  <xs:complexType>
    <xs:attribute name="entity-ref" type="doc:identifier" use="optional" />
    <xs:attribute name="attribute" type="doc:identifier" use="required" />
  </xs:complexType>
</xs:element>


<!--              SUBTYPE_CONSTRAINT declaration                    -->
<xs:element name="subtype.constraint">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:element ref="doc:graphic.element" minOccurs="0"/>
```

```
    </xs:sequence>
    <xs:attribute name="name" type="doc:identifier" use="required" />
    <xs:attribute name="entity" type="doc:identifier" use="required" />
    <xs:attribute name="abstract.supertype" type="xs:boolean"
        default="false" />
    <xs:attribute name="totalover" type="doc:identifiers" use="optional" />
    <xs:attribute name="super.expression" type="doc:express.text"
        use="optional" />
  </xs:complexType>
</xs:element>


<!--              algorithms                              -->

<!--              FUNCTION declaration                    -->
<xs:element name="function">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:element ref="doc:parameter" minOccurs="0" maxOccurs="unbounded"/>
      <xs:group ref="doc:datatype"/>
<!-- represents the return type of the function -->
      <xs:element ref="doc:algorithm" />
    </xs:sequence>
    <xs:attribute name="name" type="doc:identifier" use="required" />
  </xs:complexType>
</xs:element>


<!--              PROCEDURE declaration                    -->
<xs:element name="procedure">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:element ref="doc:parameter" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="doc:algorithm" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="name" type="doc:identifier" use="required" />
  </xs:complexType>
</xs:element>

<!--              parameter declaration                    -->
<xs:element name="parameter">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:group ref="doc:datatype"/>
    </xs:sequence>
    <xs:attribute name="name" type="doc:identifier" use="required" />
  </xs:complexType>
</xs:element>

<!-- algorithm represents the function/procedure body,
     including local variable declarations, as a string -->
<xs:element name="algorithm" type="doc:express.text" />


<!--              RULE declaration                    -->
<xs:element name="rule">
  <xs:complexType>
    <xs:sequence>
```

```
      <xs:element ref="doc:description" minOccurs="0"/>
      <xs:element ref="doc:algorithm" minOccurs="0"/>
      <xs:element ref="doc:where" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="doc:identifier" use="required" />
    <xs:attribute name="appliesto" type="doc:identifiers" use="required" />
  </xs:complexType>
</xs:element>

</xs:schema>
```

 

# Annex E
# (normative)

# Valid populations of EXPRESS entity instances

A collection of EXPRESS entity instances that, taken as a group, satisfies all the RULEs and other constraints stated in an EXPRESS schema is said to be a *valid population* for that schema. ISO 10303-11 specifies what constitutes the validity of such a population. This Annex specifies mechanisms for determining valid populations of EXPRESS entity instances represented by XML elements specified in this Part of ISO 10303.

One of the requirements for a valid population is that every EXPRESS entity instance that is referenced by an attribute of any entity instance in the population is itself in the population. The collection of EXPRESS entity instances represented in a unit of serialization element is not necessarily a valid population as defined by ISO 10303-11. The element may contain references to EXPRESS entity instances represented in other unit of serialization elements in the same document, to EXPRESS entity instances represented in unit of serialization elements in another document, or to EXPRESS entity instances having other representations.

NOTE 1 All of the above are forms of "external reference", as described in 9.3.2.

As a consequence, a valid population can consist of EXPRESS entity instances represented by XML elements from more than one unit of serialization element, possibly contained in more than one document. There is no requirement for the several unit of serialization elements (or other representations) so related to be governed by the same context schema. Subclause E.1 of this Annex specifies rules that may be used to determine the validity of references to EXPRESS entity instances represented outside of the referencing unit of serialization element.

NOTE 2 A "data section", as defined in ISO 10303-21:2001 [1], may contain representations of some EXPRESS entity instances that are (externally) referenced by EXPRESS entity instance representations in unit of serialization elements. The data section also conforms to some context schema, which may or may not be the same as the context schema for the referencing unit of serialization element.

A pre-processor that creates a document that conforms to this Part of ISO 10303 may specify the collections of entity instances that are intended to represent valid populations using the `schema_population` element specified in 5.3. In that element, the identification of a *determination method* – the algorithm for identifying the EXPRESS entity instances that constitute the population – is required. Subclause E.2 of this annex defines algorithms that may be used as determination methods, and their corresponding identifiers.

The mechanisms specified in this annex are expected to be commonly used, but they are not intended to be exclusive. Other mechanisms for determining valid populations and valid entity references may be specified elsewhere, may be identified in `schema_population` elements in conforming documents, and may be used by conforming postprocessors.

NOTE 3 The remaining text of this annex is intended to be functionally equivalent to the text of Annex G of ISO-10303-21:2001. But the text refers to objects defined in this part of ISO 10303 where appropriate and some of the defaults are different.

## E.1 Reference validity

This subclause describes three methods of determining what references between entity instances are to be considered valid for determining validity of a population. Other methods are possible but are not specified in this Part of ISO 10303.

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

When determining validity of a population, an implementation must refer to the document that defines the schema for the complete EXPRESS specification and other requirements and constraints not stated in EXPRESS. If the schema is to be used in combination with others, that document must also define what references are to be considered valid.

It is expected that this determination will be done once and will govern all uses of that set of EXPRESS schemas that conform to that document. It is also expected that such a document will designate a particular EXPRESS schema as the governing schema under which validation of a population will be determined. That is, that document will explicitly or implicitly specify the reference validity mechanism to be used for all populations to be validated under the designated schema (as governing schema).

### E.1.1 Schema identity method

If this method is used, references between EXPRESS entity instances defined in different unit of serialization elements (or other representations of schema instances) shall be valid only when the referencing unit of serialization element and the unit of serialization element containing the referenced entity instance are governed by the same context schema.

For this purpose, the term "same context schema" shall refer to EXPRESS schemas sharing a common EXPRESS identifier and intent, but it may be broadened to include different versions of the schema. The mechanism by which a postprocessor can determine whether two EXPRESS schemas are the same when the form of their identification is not identical (for example, when one is by name and another by URI) is not specified in this Part of ISO 10303.

EXAMPLE Consider the following two EXPRESS schemas:

```
SCHEMA mr_smiths_garden;

ENTITY garden;
  has_beds : SET [1:?] OF bed;
  neighbours_garden : OPTIONAL garden;
END_ENTITY;

ENTITY bed;
  name : STRING;
  holds_plants : SET [1:?] OF plant;
END_ENTITY;

ENTITY plant
  name : STRING;
END_ENTITY;

END_SCHEMA (* mr_smiths_garden *);


SCHEMA mr_jones_garden;

USE FROM mr_smiths_garden (plant);
```

 

```
ENTITY garden;
   has_beds : SET [1:?] OF bed;
   neighbours_garden : OPTIONAL garden;
END_ENTITY;

ENTITY bed;
  name : STRING;
  holds_plants : SET [1:?] OF plant;
END_ENTITY;

ENTITY outdoor_plant
  SUBTYPE OF (plant);
  survival_ph_range : ph_range;
END_ENTITY;

TYPE ph_value = REAL; END_TYPE;
ENTITY ph_range;
  min : ph_value;
  max : ph_value;
END_ENTITY;

END_SCHEMA;
```

And the related `iso_10303_28` element:

```
<xs:schema
  targetNamespace="urn:mrsmith.co.uk:exp/Mr_smiths_garden"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:exp=
   "urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"
  xmlns:msg="urn:mrsmith.co.uk:exp/Mr_smiths_garden">

 <xs:complexType name="Garden">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
      <xs:sequence>

        <xs:element name = "Has_beds">
          <xs:complexType>
            <xs:complexContent>
              <xs:restriction base="exp:array">
                <xs:sequence>
                  <xs:element ref="msg:Bed"
                    minOccurs="1" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="id"  type="xs:ID"
                  use="optional"/>
                <xs:attribute name="ref" type="xs:IDREF"
                  use="optional"/>
                <xs:attribute ref="exp:arraySize" use="optional"/>
                <xs:attribute ref="exp:itemType" fixed="msg:Bed"/>
                <xs:attribute ref="exp:cType" fixed="set"/>
              </xs:restriction>
            </xs:complexContent>
          </xs:complexType>
        </xs:element>

        <xs:element name = "Neighbours_garden"
            minOccurs="0" nillable="true">
          <xs:complexType>
```

```
                <xs:choice>
                  <xs:element ref = "msg:Garden"/>
                </xs:choice>
              </xs:complexType>
            </xs:element>

          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="Garden" type="msg:Garden" nillable="true"
      block="extension restriction" substitutionGroup="exp:Entity"/>

    <xs:complexType name="Bed">
      <xs:complexContent>
        <xs:extension base="exp:Entity">

          <xs:sequence>
          <xs:element name = "Name" type = "xs:normalizedString"/>
            <xs:element name = "Holds_plants">
              <xs:complexType>
                <xs:complexContent>
                  <xs:restriction base="exp:array">
                    <xs:sequence>
                      <xs:element ref="msg:Plant"
                        minOccurs="1" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="id"  type="xs:ID"
                      use="optional"/>
                    <xs:attribute name="ref" type="xs:IDREF"
                      use="optional"/>
                    <xs:attribute ref="exp:arraySize" use="optional"/>
                    <xs:attribute ref="exp:itemType"
fixed="msg:Plant"/>
                    <xs:attribute ref="exp:cType" fixed="set"/>
                  </xs:restriction>
                </xs:complexContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>

        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:element name="Bed" type="msg:Bed" nillable="true"
      block="extension restriction" substitutionGroup="exp:Entity"/>

    <xs:complexType name="Plant">
      <xs:complexContent>
        <xs:extension base="exp:Entity">

          <xs:sequence>
          <xs:element name = "Name" type = "xs:normalizedString"/>

          </xs:sequence>

        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
```

 

```
 <xs:element name="Plant" type="msg:Plant" nillable="true"
  block="extension restriction" substitutionGroup="exp:Entity"/>

</xs:schema>


<xs:schema
  targetNamespace="urn:mrjones.co.uk:xs/Mr_jones_garden"
  xmlns:mjg="urn:mrjones.co.uk:xs/Mr_jones_garden"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:p28=
   "urn:iso:std:iso:10303:-28:ed-2:tech:XMLschema:common"
  xmlns:msg="urn:mrsmith.co.uk:exp/Mr_smiths_garden">

<xs:import namespace="urn:mrsmith.co.uk:exp/Mr_smiths_garden"
    schemaLocation="Mr_smiths_garden.xs"/>

<xs:complexType name="Garden">
  <xs:complexContent>
   <xs:extension base="exp:Entity">
       <xs:sequence>

         <xs:element name = "Has_beds">
           <xs:complexType>
             <xs:complexContent>
               <xs:restriction base="exp:array">
                 <xs:sequence>
                   <xs:element ref="mjg:Bed"
                     minOccurs="1" maxOccurs="unbounded"/>
                 </xs:sequence>
                 <xs:attribute name="id"  type="xs:ID"
                   use="optional"/>
                 <xs:attribute name="ref" type="xs:IDREF"
                   use="optional"/>
                 <xs:attribute ref="exp:arraySize" use="optional"/>
                 <xs:attribute ref="exp:itemType" fixed="mjg:Bed"/>
                 <xs:attribute ref="exp:cType" fixed="set"/>
               </xs:restriction>
             </xs:complexContent>
           </xs:complexType>
         </xs:element>

         <xs:element name = "Neighbours_garden"
              minOccurs="0" nillable="true">
           <xs:complexType>
             <xs:choice>
               <xs:element ref = "mjg:Garden"/>
             </xs:choice>
           </xs:complexType>
         </xs:element>

       </xs:sequence>
     </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="Garden" type="mjg:Garden" nillable="true"
  block="extension restriction" substitutionGroup="exp:Entity"/>

<xs:complexType name="Bed">
```

```
  <xs:complexContent>
    <xs:extension base="exp:Entity">

      <xs:sequence>
      <xs:element name = "Name" type = "xs:normalizedString"/>
        <xs:element name = "Holds_plants">
          <xs:complexType>

                <xs:sequence>
                  <xs:element ref="msg:Plant"
                    minOccurs="1" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="id"  type="xs:ID"
                  use="optional"/>
                <xs:attribute name="ref" type="xs:IDREF"
                  use="optional"/>
                <xs:attribute ref="exp:arraySize" use="optional"/>
                <xs:attribute ref="exp:itemType"
fixed="msg:Plant"/>
                <xs:attribute ref="exp:cType" fixed="set"/>

          </xs:complexType>
        </xs:element>
      </xs:sequence>

    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="Bed" type="mjg:Bed" nillable="true"
  block="extension restriction" substitutionGroup="exp:Entity"/>

<xs:complexType name="Outdoor_plant">
  <xs:complexContent>
    <xs:extension base="msg:Plant">

      <xs:sequence>
        <xs:element name = "Survival_ph_range">
          <xs:complexType>
            <xs:choice>
              <xs:element ref = "mjg:Ph_range"/>
            </xs:choice>
          </xs:complexType>
        </xs:element>
      </xs:sequence>

    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="Outdoor_plant" type="mjg:Outdoor_plant"
  nillable="true"
  block="extension restriction" substitutionGroup="msg:Plant"/>

<xs:simpleType name = "Ph_value">
  <xs:restriction base = "xs:double"/>
</xs:simpleType>

 <xs:complexType name="Ph_range">
  <xs:complexContent>
    <xs:extension base="exp:Entity">
```

 

```
        <xs:sequence>
          <xs:element name = "Min" type = "mjg:Ph_value"/>
          <xs:element name = "Max" type = "mjg:Ph_value"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

Using the schema identity method:

— in the "TomJones" unit of serialization element, both osb:edokey references, to "g1" and "plant1", are invalid. They refer to elements in an unit of serialization element whose context schema is mr_smiths_garden, which differs from the context schema for the references — the mr_jones_garden schema.

— in the "GeorgeJones" unit of serialization element, both osb:edokey references, to "g2" and "plant2", are valid. They refer to elements in an unit of serialization element whose context schema is mr_jones_garden, the same as the context schema for the references.

### E.1.2 EXPRESS interface specification method

If this method is used, references between EXPRESS entity instances defined in different unit of serialization elements (or other representations of schema instances) shall be valid when the unit of serialization elements have the same context schema (as in E.1.1) or have different context schemas that are related by EXPRESS interface specifications, as specified in Clause 11 of ISO 10303-11.

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

EXAMPLE Consider the two EXPRESS schemas and the iso_10303_28 element in E.1.1:

Using the EXPRESS interface specification method:

— in the "TomJones" unit of serialization element, the osb:edokey reference to "g1" is invalid. It refers to an EXPRESS entity instance represented in an unit of serialization element whose context schema is mr_smiths_garden, and the EXPRESS entity data type garden is not interfaced into the context schema for the references, the mr_jones_garden schema.

— in the "TomJones" unit of serialization element, the osb:edokey reference to "plant1" is valid. It refers to an EXPRESS entity instance represented in an unit of serialization element whose context schema is mr_smiths_garden, but the EXPRESS entity data type plant is interfaced into the context schema for the references, the mr_jones_garden schema.

— in the "GeorgeJones" unit of serialization element, both osb:edokey references, to "g2" and "plant2", are valid. They refer to elements in an unit of serialization element whose context schema is mr_jones_garden, the same as the context schema for the references.

### E.1.3 SDAI domain equivalence method

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

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

EXAMPLE Consider the two EXPRESS schemas and the `iso_10303_28` element in E.1.1:

For the purpose of this example, assume that we have domain equivalence information, determined using one of the techniques described in ISO 10303-22, stating that EXPRESS entity data types `garden` and `bed` are domain equivalent in both schemas. That is:

```
mr_jones_garden.garden is DEQ to mr_smiths_garden.garden
mr_jones_garden.bed is DEQ to mr_smiths_garden.bed
```

And the USE statement in mr_jones_garden schema implies that the EXPRESS entity data type plant is domain equivalent in both schemas.

Then, using the SDAI domain equivalence method:

— in the `"TomJones"` unit of serialization element, the `osb:edokey` references to `"g1"` and `"plant1"` are valid. They refer to EXPRESS entity instances represented in an unit of serialization element whose context schema is `mr_smiths_garden`, but the EXPRESS entity data types `garden` and `plant` in the `mr_smiths_garden` schema are domain equivalent to their counterparts in the context schema for the references, the `mr_jones_garden` schema.

## E.2 Determining population of a schema

The `schema_population` element (see 5.3) associates an EXPRESS schema – the "governing schema" – with a collection of EXPRESS entity instances. The XML attribute `determination_method` identifies an algorithm for selecting a collection of entity instances from the given set of unit of serialization elements – the "governed sections". This subclause describes three determination methods and specifies the corresponding values of the `determination_method` attribute. Other methods are possible but are not specified in this Part of ISO 10303.

### E.2.1 Section boundary method

The `determination_method` attribute shall have the value `"SECTION_BOUNDARY"` when the section boundary method is to be used. The collection of entity instances shall consist of the following:

— all EXPRESS entity instances represented in the given unit of serialization elements.

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

NOTE There is no requirement for the governing schema for the population to be the same as the context schema for any of the unit of serialization elements, although clearly they must be related. For example, they can be related by one of the mechanisms specified in E.1 above.

EXAMPLE Consider the two EXPRESS schemas and the `iso_10303_28` element in E.1.1. This `iso_10303_28` does not contain any `schema_population` elements, and therefore no schema validation of the instance data represented in this `iso_10303_28` element is defined by this Part of ISO 10303:

 

For this example, suppose the following modification to the iso_10303_28 element: Before the first unit of serialization element, insert:

```
<schema_population governing_schema="mr_smiths_garden"
  governed_sections="Smith" />
<schema_population governing_schema="mr_jones_garden"
  governed_sections="mjg TomJonesGarden" />
```

For both of the above, the default value of the determination_method attribute – "SECTION_BOUNDARY" – is implied.

When determining the validity of the populations, the following must be considered:

— The first schema_population element defines a collection of instances that is governed by schema mr_smiths_garden. All instances in unit of serialization element "Smith" must satisfy the requirements and constraints of schema mr_smiths_garden. In this example, the population contains no references to instances outside the "Smith" unit of serialization element and satisfies all constraints of schema mr_smiths_garden.

— The second schema_population element defines a collection of instances that is governed by schema mr_jones_garden. All instances in unit of serialization elements "TomJones" and "GeorgeJones", as a single population, must satisfy the requirements and constraints of schema mr_jones_garden. In this population:

— in the "GeorgeJones" unit of serialization element, both osb:edokey references, to "g2" and "plant2", are to instances represented in the named unit of serialization elements, that is, contained in the population.

— in the "TomJones" unit of serialization element, both osb:edokey references, to "g1" and "plant1", are to instances outside of the named unit of serialization elements, that is, not contained in the population. Therefore, the value of the EXPRESS attribute garden.neighbours_garden for EXPRESS entity instance "g2" is treated as unset. This is valid, because the EXPRESS attribute is declared to be OPTIONAL. But the value of the "explant1" instance in the value of EXPRESS attribute bed.holds_plants for EXPRESS entity instance bed2 is also treated as unset. Since unset is not a valid member of a SET, the population is not valid under the schema.

### E.2.2 Include all compatible method

The determination_method attribute shall have the value "INCLUDE_ALL_COMPATIBLE" when the include all compatible method is to be used. The collection of EXPRESS entity instances shall consist of the following:

— all EXPRESS entity instances represented in the given unit of serialization elements; and

— all EXPRESS entity instances represented in any other unit of serialization element contained within the iso_10303_28 element, such that the EXPRESS entity data type of the instance is compatible with an the data type of an attribute of an EXPRESS entity data type defined by the governing schema for the population.

For this purpose, compatibility of entity data types shall be determined by one of the methods specified in E.1, or some other reference validity method, and shall be the subject of separate agreement. If unspecified, compatibility shall be determined by the schema identity method (see E.1.1).

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

EXAMPLE Consider the two EXPRESS schemas and the `iso_10303_28` element in E.1.1. For this example, suppose the following modification to the `iso_10303_28` element: Before the first unit of serialization element, insert:

```
<schema_population governing_schema="mr_jones_garden"
  governed_sections="TomJones"
  determination_method="INCLUDE_ALL_COMPATIBLE" />
```

When determining the validity of the population, the following must be considered:

— The `schema_population` element defines a collection of instances that is governed by schema `mr_jones_garden`. All instances in unit of serialization element `"TomJones"` are included in the population.

— All instances in unit of serialization element `"GeorgeJones"` are also included in the population, because each of those instances (of entity data types `garden` and `bed`) could be a valid value of an EXPRESS attribute of a `garden` instance in the `"TomJones"` population, no matter which of the methods in E.1 is used to determine reference validity.

— The collection of instances in unit of serialization element `"Smith"` that are included in the population depends on which reference validity method in E.1 is used:

— if the schema identity method is used, none of the instances in `"Smith"` is included in the population. The population is not valid, because the instance represented by `osb:edokey "explant1"` is treated as unset, which is not a valid member of a `SET`.

— if the EXPRESS interface specification method is used, the `plant` instance `"plant1"` in `"Smith"` is included in the population, because its data type `plant` is the same data type as the data type `plant` in the `mr_jones_garden` schema and it could be a value of `bed.holds_plants` for some bed instance represented in the unit of serialization element `"TomJones"`. No other instance in the unit of serialization element `"Smith"` is contained in the population. The population is valid, because only the instance represented by `osb:edokey "exg1"` is treated as unset, and the affected EXPRESS attribute is `OPTIONAL`.

— if the SDAI domain equivalence method is used, and the following domain equivalence declarations are present:

```
mr_jones_garden.garden is DEQ to mr_smiths_garden.garden
mr_jones_garden.bed is DEQ to mr_smiths_garden.bed
```

then all of the instances of entity data types `garden` and `bed`, as well as all instances of `plant`, in the unit of serialization element `"Smith"` are included in the population and the population is valid, that is, there are no unsatisfied references.

### E.2.3 Include referenced instance method

The `determination_method` attribute shall have the value `"INCLUDE_REFERENCED"` when the include referenced instance method is to be used. The collection of entity instances shall consist of the following:

— all EXPRESS entity instances represented in the given unit of serialization elements; and

 

— all EXPRESS entity instances that are referenced by external reference elements in the given unit of serialization elements, wherever they may be represented.

The validity of each external reference may be determined by one of the methods specified in E.1, or some other reference validity method, and shall be the subject of separate agreement. If unspecified, reference validity shall be determined by the schema identity method (see E.1.1). When determining the satisfaction of requirements and constraints of a schema, invalid references shall behave as "unset" values.

EXAMPLE Consider the two EXPRESS schemas and the `iso_10303_28` element in E.1.1. For this example, suppose the following modification to the `iso_10303_28` element: Before the first unit of serialization element, insert:

```
<schema_population governing_schema="mr_jones_garden"
  governed_sections="TomJones"
  determination_method="INCLUDE_REFERENCED" />
```

When determining the validity of the population, the following must be considered:

— The `schema_population` element defines a collection of instances that is governed by schema `mr_jones_garden`. All instances in unit of serialization element `"TomJones"` are included in the population.

— No instance in unit of serialization element `"GeorgeJones"` is included in the population, because none of them is referenced by any instance in unit of serialization element `"TomJones"`.

— The collection of instances in unit of serialization element `"Smith"` that are included in the population depends on which reference validity method in E.1 is used:

— if the schema identity method is used, none of the instances in `"Smith"` is included in the population. The referenced instances `"g1"` and `"plant1"` are not included, because their data types are not defined in the `mr_jones_garden` schema (the context schema of the reference) and therefore they do not satisfy the reference validity requirement. And the population is not valid, because the instance represented by `osb:edokey "explant1"` is treated as unset, which is not a valid member of a `SET`.

— if the EXPRESS interface specification method is used, the `plant` instance `"plant1"` in `"Smith"` is included in the population, because it is referenced by `osb:edokey "explant1"` and its data type `plant` is interfaced into the `mr_jones_garden` schema, thus satisfying the reference validity requirement. No other instance in the unit of serialization element `"Smith"` is contained in the population. The `garden` instance `"g1"` is not included in the population, even though it is referenced, because its data type `mr_smiths_garden.garden` is not interfaced into the `mr_jones_garden` schema, and it does not satisfy the reference validity requirement.. The population is valid, because only the instance represented by `osb:edokey "exg1"` is treated as unset, and the affected EXPRESS attribute is `OPTIONAL`.

— if the SDAI domain equivalence method is used, and the following domain equivalence declarations are present:

```
mr_jones_garden.garden is DEQ to mr_smiths_garden.garden
mr_jones_garden.bed is DEQ to mr_smiths_garden.bed
```

then the referenced instances `"g1"` and `"plant1"` are included in the population, because their data types are domain equivalent to their counterparts in the `mr_jones_garden` schema (the context schema of the reference) and therefore they satisfy the reference validity requirement. But the `"bed1"` instance is not included in the population, because it is not referenced by any instance in unit of serialization element `"TomJones"`. And the population is not valid — the `SET` value of EXPRESS attribute `has_beds` for the `garden` instance `"g1"` contains an unset member.

# Annex F
(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 (2)}
```

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.

## F.1 Object identifier for Base XML Schema

The object identifier

```
{iso standard 10303 part (28) version (2) Base-XML-Schema(1)}
```

is assigned to the Base XML Schema defined in Annex C of this part of ISO 10303.

## F.2 Object identifier for Configuration Language Schema

The object identifier

```
{iso standard 10303 part (28) version (2) Configuration-Language-
Schema(2)}
```

is assigned to the Configuration Language schema defined in Annex B of this part of ISO 10303.

## F.3 Object identifier for Document Schema

The object identifier

```
{iso standard 10303 part (28) version (2) Document-Schema(3)}
```

is assigned to the Document Schema defined in Annex D of this part of ISO 10303.

# Annex G
(informative)
# Configuration language examples

This Annex provides examples of configuration language directives and their effects on the derived XML schema and on the XML representation of the data instances.

The following EXPRESS Schema will be used for these examples:

```
ENTITY person;
id            : identifier;
last_name     : OPTIONAL label;
first_name    : OPTIONAL label;
END_ENTITY; -- person

ENTITY person_and_organization;
the_person       : person;
the_organization : organization;
END_ENTITY; -- person_and_organization

ENTITY organization;
id          : OPTIONAL identifier;
name        : label;
END_ENTITY; -- organization;
```

The following instance data will be used for these examples:

```
#7 = PERSON('s32', 'Gamgee', 'Sam');
#8 = PERSON('s77', 'Baggins', 'Bilbo');
#9 = PERSON_AND_ORGANIZATION(#7,#11);
#10 = PERSON_AND_ORGANIZATION(#8,#11);
#11 = ORGANIZATION('sci', 'Shyre Crafts, Inc.');
```

In these examples, only the portion of the XML instance data for instance `#9` will be shown. The other entity instances, however, may still affect the XML encoding.

EXAMPLE 1 **exp-attribute="double-tag"**, everything by-reference

Configuration:

```
<option exp-type="root" entity-attribute="double-tag">
```

Instance Data:

```
<person_and_organization id="i9">
  <the_person>
    <person ref="i7"/>
  </the_person>
  <the_organization>
    <organization ref="i11" />
  </the_organization>
</person_and_organization>

<person id="i7">
  <id>s32</id>
  <last_name>Gamgee</last_name>
  <first_name>Sam</first_name>
```

```
</person>

<person id="i8">
  <id>s77</id>
  <last_name>Baggins</last_name>
  <first_name>Bilbo</first_name>
</person>

<organization id="i11">
  <id>sci</id>
  <name>Shyre Crafts, Inc.</name>
</organization>
```

EXAMPLE 2 **exp-attribute="double-tag"**, by-value data, person_and_organization as root

Configuration:

```
<option exp-type="value" entity-attribute="double-tag">
<entity select="person_and_organization" exp-type="root"/>
```

Instance Data:

```
<person_and_organization id="i9">
  <the_person>
    <person>
      <id>s32</id>
      <last_name>Gamgee</last_name>
      <first_name>Sam</first_name>
    </person>
  </the_person>
  <the_organization>
    <organization>
      <id>sci</id>
      <name>Shyre Crafts, Inc.</name>
    </organization>
  </the_organization>
</person_and_organization>
```

EXAMPLE 3 **exp-attribute="attribute-tag"**, by-reference data

Configuration:

```
<option exp-type="root" entity-attribute="attribute-tag">
```

Instance Data:

```
<person_and_organization id="i9">
  <the_person person ref="i7"/>
  <the_organization ref="i11" />
</person_and_organization>
```

EXAMPLE 4 **exp-attribute="attribute-tag"**, by-value data

Configuration:

```
<option exp-type="value" entity-attribute="attribute-tag">
<entity select="person_and_organization" exp-type="root"/>
```

Instance Data:

 

```
<person_and_organization id="i9">
  <the_person>
    <id>s32</id>
    <last_name>Gamgee</last_name>
    <first_name>Sam</first_name>
  </the_person>
  <the_organization>
    <id>sci</id>
    <name>Shyre Crafts, Inc.</name>
  </the_organization>
</person_and_organization>
```

NOTE An `xsi:type` attribute is required on the <the_person> or <the_organization> elements when the entity type of the attribute has any subtypes. In this example, there are no subtypes, so the data can be simplified as shown above.

EXAMPLE 5 **exp-attribute="type-tag"**, by-reference data

NOTE The type-tag option can only be specified on a per-attribute basis.

Configuration:

```
<entity select="person_and_organization" exp-type="root">
  <attribute select="the_person" exp-attribute="type-tag"
       content="ref"/>
  <attribute select="the_organization" exp-attribute="type-tag"
       content="ref"/>
</entity>
```

Instance Data:

```
<person_and_organization id="i9">
  <the_person person ref="i7"/>
  <the_organization ref="i11" />
</person_and_organization>
```

EXAMPLE 6 **exp-attribute="no-tag"**

NOTE The no-tag option can only be specified on a per-attribute basis.

Configuration:

```
<entity select="person_and_organization" exp-type="root">
  <attribute select="the_person" exp-attribute="no-tag"/>
  <attribute select="the_organization" exp-attribute="no-tag"/>
</entity>
```

Instance Data:

```
<person_and_organization id="i9">
  <per-id>s32</per-id>
  <last_name>Gamgee</last_name>
  <first_name>Sam</first_name>
  <id>sci</id>
  <name>Shyre Crafts, Inc.</name>
</person_and_organization>
```

In this case, because both `THE_PERSON` and `THE_ORGANIZATION` have an attribute called '`id`', achieving the above effect requires renaming one of the `id` attributes as follows:

```
<entity select="person">
```

```
    <attribute select="id" name="per-id" />
</entity>
```

EXAMPLE 7 **exp-attribute="attribute-content"**

Configuration:

```
<option exp-type="root" entity-attribute="attribute-content" concrete-
attribute="attribute-content">
```

Instance Data:

```
<person_and_organization id="i9" the_person="i7"
          the_organization="i11" />
<person id="i7" Id="s32" last_name="Gamgee" first_name="Sam" />
<organization id="i11" Id="sci" name="Shyre Crafts, Inc." />
```

  

# Bibliography

[1] ISO 10303-21:2002, *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure*

[2] ISO 10303-203:1994, *Industrial automation systems and integration – Product data representation and exchange – Part 203: Application protocol: Configuration controlled 3D design of mechanical parts and assemblies*

[3] *URN Syntax,* Internet Engineering Task Force RFC 2141, May 1997 [cited 2004-03-15]. Available from World Wide Web: <http://www.ietf.org/rfc/rfc2141.txt>

[4] *Uniform Resource Names (URN) Namespace Definition Mechanism,.* Internet Engineering Task Force RFC 3406, October 2002. [cited 2004-03-15]. Available from World Wide Web: <http://www.ietf.org/rfc/rfc3406.txt>

[5] *XML Path Language (Xpath) Version 1.0,* World Wide Web Consortium Recommendation 16 November 1999 [cited 2004-03-15]. Available from World Wide Web: <http://www.w3.org/TR/xpath/>

[6] *XML Schema Part 0: Primer Second Edition*, World Wide Web Consortium Recommendation 28 October 2004. Available from World Wide Web: <http://www.w3.org/TR/xmlschema-0/>

[7] Megginson, David. *Structuring XML Documents*, Chapters 10 and 11, Prentice Hall, 1998, ISBN 0-13-642299-3

# Index

**ICS 25.040.40**

Price based on 309 pages