ISO TC184/SC4/WG10 N285

Date: 1999-10-24

ISO TC184/SC4/WG11 N090

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

ABSTRACT

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

KEYWORDS:

Implementation methods XML representation

COMMENTS TO READER:

This document is the first rough draft of part 28.

It has been produced with support of the BSI CDS scheme.

In a number of areas there are comments concerning the draft status of the part (given in italic.)

Project Leader:	Nigel Shaw	**Project Editor**:	Robin La Fontaine
Address:	Eurostep Limited Castell, Bodfari, Denbigh LL16 4HT UK	**Address**:	Monsell EDM, Monsell House, Monsell Lane, Upton-on-Severn UK WR8 0QN
Telephone:	44 (0) 1745 710677	**Telephone**:	44 (0) 1684 592144
Facsimile:	44 (0) 1745 710688	**Facsimile**:	44 (0) 1684 594504
E-mail:	nigel.shaw@eurostep.com	**E-mail**:	robin@monsell.co.uk

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.

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.

International Standard ISO 10303-28 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration,* Subcommittee SC4, *Industrial data*.

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

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

This part of ISO 10303 is a member of the implementation methods series. The implementation methods specify <???>.

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

Introduction

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

This part of ISO 10303 specifies means by which data and schemas specified using the EXPRESS language (ISO 10303-11) can be encoded using XML.

XML provides a basic syntax that can be used in many different ways to encode information. In this part of ISO 10303, the following uses of XML are specified:

a) A late bound XML architectural Document Type Declaration (DTD) that enables any EXPRESS schema to be encoded;

b) An extension to the late bound DTD to enable data corresponding to any EXPRESS schema to be encoded as XML;

c) A canonical form for the late bound DTD that is derived from the architectural DTD;

d) The use of SGML architectures to enable early binding XML forms to be defined that are compatible with the late binding.

The use of architectures allows for different early bindings to be defined that are compatible with each other and can be processed using the architectural DTD.

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

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

1 Scope

This part of ISO 10303 specifies use of the Extensible Markup Language (XML) to enable the transfer of both schemas and data specified using the EXPRESS information specification language (ISO 10303-11).

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

a) The specification of an architectural XML DTD that enables any EXPRESS schema and/or data conforming to that schema to be encoded as XML.

   Note 1 This generic DTD is referred to as a late-bound DTD in that it uses an approach that is independent of the schema. It allows for a number of choices in how EXPRESS-driven data is encoded.

b) The specification of a canonical XML DTD that enables any EXPRESS schema and/or data conforming to that schema to be encoded as XML.

   Note 2 This canonical DTD is also a late-bound DTD and is based on the architectural DTD. However it eliminates the flexibility allowed for in that DTD.

c) The means by which to define other XML DTD's that are wholly or partly based on an EXPRESS schema such that the correspondence between the XML elements and the architectural DTD can be identified and used.

   Note 3 There are many ways in which a given EXPRESS schema can be used to define an XML DTD that can be used to encode the data described by the schema. The approach used here is not to specify a single early-bound DTD but instead to specify how architectures as defined in ISO 10744 (HyTime) can be applied to enable multiple XML DTD's.

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

a) The specification of any specific mapping to XML from the EXPRESS language where the form of the mapping is dependent on the specific EXPRESS schema.

b) The specification of any specific XML DTD corresponding to a specific EXPRESS schema.

Note 3 Given an EXPRESS schema, it is feasible to produce a schema specific XML DTD. Such DTD's may make use of the attribution capabilities defined in this part of ISO 10303.

c) The specification of a mapping from an XML to an EXPRESS schema.

Note 4 Given an XML DTD and one or more data sets corresponding to it, it is feasible to define an EXPRESS schema describing the data. However, this requires an understanding of the semantics of the data that may not be captured by the XML DTD.

2 Normative references

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

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

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

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

ISO 10303-11: ????, *Industrial automation systems and integration - product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual. Amendment 1* ???

ISO 8879:1986, *Information processing – Text and office systems – Standard Generalized Markup Language.*

W3C XML *What is the most authoritative source?*

ISO/IEC 10744:????, *Information processing – Hypermedia/Time-based structuring language (HyTime).*

3 Terms and definitions

3.1 Terms defined in ISO 10303-1

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

— data;

— information.

3.2 Terms defined in ISO 10303-11

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

— entity instance;

— ???.

3.3 Terms defined in ISO 8879

For the purposes of this part of ISO 10303, the following terms defined in ISO 8879 apply.

— element;

— empty element;

— tag.

3.4 Terms defined in ISO/IEC 10744

For the purposes of this part of ISO 10303, the following terms defined in ISO/IEC 10744 apply.

— Architectural Engine;

— Architectural forms;

— Base architecture

— Client DTD.

3.5 Other terms and definitions

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

3.3.1
EXPRESS-driven data
Data that is known to correspond to an identified EXPRESS schema.

Note Such data can always be transformed into a set of entity instances according to one of the ISO 10303 implementation methods.

3.3.2
???
placeholder.

3.6 Abbreviations

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

— DTD Document Type Declaration;

— HyTime Hypermedia/Time-based structuring language

— SGML Standard Generalized Markup Language;

— XML Extensible Markup Language.

3.7 Terminology

EXPRESS and XML use similar or identical words for different concepts. Where there is scope for confusion the prefixes of XML- and EXPRESS- are used to distinguish between the different cases. These prefixes are used for the following terms:

— Attribute.

4 Fundamental concepts and assumptions

The EXPRESS language is used to specify information. Such a specification is given as an EXPRESS schema. ISO 10303 provides multiple implementation methods that can be used for data described by means of an EXPRESS schema.

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

This part of ISO 10303 specifies how XML can be applied to both EXPRESS schemas and data corresponding to EXPRESS schemas. It is assumed that an EXPRESS schema is available.

4.1 Early and Late binding

Given an EXPRESS schema specifying some information, it is possible to use two different approaches in defining an XML DTD for the same information. These two approaches are: Late Binding and early Binding.

— A Late Bound DTD can be used in the same manner for any EXPRESS schema. It does not define any constructs that are specific to the schema.

— An Early Bound DTD is based on the specific DTD and embeds specific aspects, such as names or structures, from the schema in the DTD.

There are many possible DTD's that can be constructed for both the late and early bindings. In this part of ISO 10303 two DTD’s are specified for the late bound case. The first is specified as a base architecture for both the second (canonical) late-bound DTD and for defining early bound DTD’s. By means of using the architectural forms provided by ISO 10744, A number of examples are provided in annex F, including the definition of one possible mapping from EXPRESS to XML.

4.2 Use of Architectural forms

This part of ISO 10303 makes use of the SGML/HyTime (ISO 10744) concept of architectural forms. Given a document in XML that corresponds with a particular DTD, architectural forms provide a standard mechanism for processing it as if it were consistent with another DTD (the meta-DTD or base architecture).

This is achieved by identifying, by means of XML-attributes, the relationship between the two DTD's such that an application can recognise an element defined in one DTD as equivalent to an element in the meta-DTD and process the data according to the meta-DTD.

SGML/HyTime architectures place constraints on the extent of differences between the two DTD's. The allowed flexibility is as follows:

— Choice of names for element types;

— Choice of names for attributes;

— Choice of using attributes or content for data items;

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

— Ability to define extra data which does not appear at all in the base architecture.

This part of ISO 10303 defines a meta-DTD that is used as the base architecture for a canonical late-bound DTD and that can also be used to define multiple early-bound DTD's. This will allow early-bound data sets to be viewed as if they were defined in terms of either the meta-DTD or the canonical late-bound DTD. Thus software written against the meta-DTD can, without modification, process data that complies with any compliant early-bound DTD. Any data set defined against a compliant early-bound DTD can also be automatically translated into a form compliant to the canonical late-bound DTDF.

An early-bound DTD is compliant if it has the late-bound DTD as its base architecture.

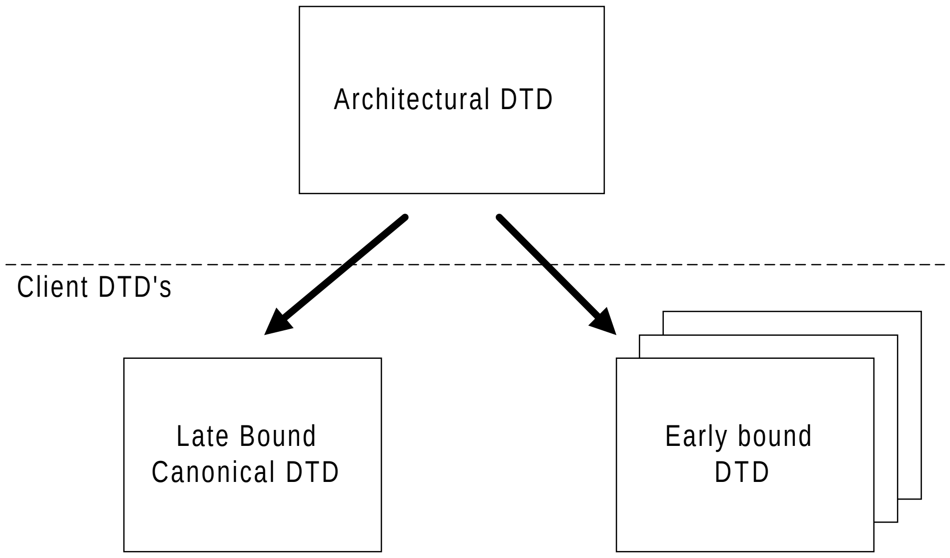

Figure 1 Relationship between DTD's

Note: Figure 1 shows the relationship between the different DTD's included and enabled by this part of ISO 10303. This approach gives flexibility in defining early-bound DTD's that can be optimised for different purposes, e.g., for display, for data exchange, for compactness.

5 Late bound XML representation of EXPRESS and EXPRESS-driven data.

Two XML DTD’s are specified that can be used to encode the following:

- One or more EXPRESS schemas;
- One or more data sets, each corresponding to an EXPRESS schema;
- A combination of EXPRESS schemas and corresponding data.

For each data sets the EXPRESS schema shall always be identified although it need not be provided encoded as XML.

The two XML DTD's are related as follows:

— The Architectural DTD is designed to act as a base architecture for both a canonical late-bound DTD and for the definition of early bound DTD's that can then be processed according to the architectural DTD. The design of this DTD allows for options that facilitate the specification of early-bound DTD's.

— The Canonical DTD is based on the architectural DTD but eliminates the options within that DTD.

The principle of the canonical late-bound format is a single DTD for any Express model. Multiple data sections can be put into a single file and each will have data for a particular schema. All entities and attributes are referenced explicitly: The model may or may not be in the same file. The various '_ref' elements are used for these references. The '_literal' values are defined in a way that is consistent with the language definition.

5.1 The meta-DTD

This clause specifies a late-bound DTD for EXPRESS-driven data.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE ISO-10303-data [
<!ELEMENT ISO_10303_data (documentation?, (schema_decl | data)*)>
]>
```

The details of the elements used to encode an EXPRESS schema (schema_decl) and a data set corresponding to an EXPRESS schema (data) are specified in annex A and annex B respectively. The documentation element allows for the inclusion of character data as annotation of the EXPRESS and data. The documentation element is used instead of providing an XML mapping from the comment syntax of EXPRESS.

Annex C specifies the correspondence between the syntax of EXPRESS (as specified in ISO 10303-11) to elements used in the late bound DTD.

> Note: The DTD presented in annex B allows for two different ways to handle instances of EXPRESS entities that are part of a supertype/subtype hierarchy: these use the nested_complex_entity_instance and flat_complex_entity_instance.

5.2 The canonical late-bound DTD

This clause specifies a late-bound DTD for EXPRESS-driven data. This DTD is derived from the meta-DTD defined in clause 5.1. It eliminates the flexibility allowed in the meta-DTD to support early-bound DTD's.

This DTD has yet to be completed. It will be specified as another normative annex. The DTD will be defined as a client of the architectural DTD specified in the previous clause. It will only allow the use of flat_complex_entity_instance and not nested_complex_entity_instance.

6 Attribution of early bound XML to use the architectural DTD

See supporting document WG11/N??? that provides a tutorial-style introduction to how early-bound DTD's can be defined based on the architectural DTD.

By definition (?) early-bound DTD's will not include the schema_decl element.

How will they point to the schema used? Is a standard mechanism required? --Yes and should be the same as for externally defined schemas for the early bound form.

Annex A
(normative)

Late Bound DTD elements for EXPRESS schema

The following XML element declarations are based on the syntax of the EXPRESS language (ISO 10303-11). They are specified in alphabetical order. The XML element express_driven_data defines the root of the structure.

Annex C provides a table identifying the correspondence between the syntax of EXPRESS and the elements specified here.

```
<!ENTITY % actual_parameter_list ' arg* '>


<!ENTITY % add_like_op ' add |
   subtract |
   or |
   xor  '>


<!ENTITY % aggregation_types '  array_type |
   bag_type |
   list_type |
   set_type '>


<!ENTITY % attribute_decl ' attribute_id |
   qualified_attribute '>


<!ENTITY % attribute_qualifier ' attribute_ref '>


<!ENTITY % built_in_constant ' const_e |
   pi |
   self |
   unknown '>


<!ENTITY % built_in_procedure ' insert |
   remove '>
```

```
<!ENTITY % constructed_types ' enumeration |
   select  '>


<!ENTITY % doc ' documentation? '>


<!ENTITY % entity_body ' explicit_attr_block?,
  derive_clause?,
  inverse_clause?,
  unique_clause?,
  where_clause? '>


<!ENTITY % function_head ' function_id,
 formal_parameter_block?,
 function_return_type '>


<!ENTITY % general_aggregation_types ' general_array_type |
  general_bag_type |
  general_list_type |
  general_set_type '>


<!ENTITY % general_ref ' parameter_ref |
   variable_ref '>


<!ENTITY % group_qualifier ' entity_ref '>


<!ENTITY % literal ' binary_literal |
  integer_literal |
  logical_literal |
  real_literal |
  string_literal '>


<!ENTITY % multiplication_like_op ' multiply |
   real_divide |
   integer_divide |
   mod |
   and |
   complex_entity_constructor '>


<!ENTITY % named_types ' entity_ref |
  type_ref  '>
```

```
<!ENTITY % procedure_head ' procedure_id,
procedure_formal_parameter_block? '>


<!ENTITY % referenced_attribute ' attribute_ref |
  qualified_attribute '>


<!ENTITY % rel_op ' less_than |
  greater_than |
  less_than_or_equal |
  greater_than_or_equal |
  not_equal |
  equal |
  instance_not_equal |
  instance_equal '>


<!ENTITY % rule_head ' rule_id,
  applies_to_entities '>


<!ENTITY % selector ' expression '>


<!ENTITY % simple_types ' binary | boolean | integer | logical | number |
real | string  '>


<!ENTITY % stmt ' alias_stmt |
  assignment_stmt |
  case_stmt |
  compound_stmt |
  escape_stmt |
  if_stmt |
  null_stmt |
  procedure_call_stmt |
  repeat_stmt |
  return_stmt |
  skip_stmt '>


<!ENTITY % supertype_constraint ' abstract_supertype_of |
   supertype_of  '>


<!ENTITY % supertype_expression ' entity_ref |
   supertype_one_of  |
   supertype_and_or  |
   supertype_and   '>
```

```
<!ENTITY % type_label ' type_label_id |
  type_label_ref '>


<!ENTITY % constant_factor '  const_e |
   pi |
   self |
   unknown  |
   constant_ref '>


<!ENTITY % declaration ' entity_decl |
  function_decl |
  procedure_decl |
  type_decl '>


<!ENTITY % qualifier_content '  attribute_ref  |
    entity_ref  |
   index_qualifier '>


<!ENTITY % rel_op_extended '  less_than |
  greater_than |
  less_than_or_equal |
  greater_than_or_equal |
  not_equal |
  equal |
  instance_not_equal |
  instance_equal  |
  in |
  like '>


<!ENTITY % schema_body ' interface_specification_block?,
   constant_block?,
   ( entity_decl |
  function_decl |
  procedure_decl |
  type_decl  |
    rule_decl)* '>


<!ENTITY % subsuper ' ( abstract_supertype_of |
   supertype_of  )?,
  subtype_of? '>


<!ENTITY % entity_head ' entity_id,
   ( abstract_supertype_of |
   supertype_of  )?,
  subtype_of?  '>
```

```
<!ENTITY % generalized_types ' aggregate_type |
   general_array_type |
  general_bag_type |
  general_list_type |
  general_set_type  |
  generic_type '>


<!ENTITY % parameter_type '  aggregate_type |
   general_array_type |
  general_bag_type |
  general_list_type |
  general_set_type  |
  generic_type  |
   entity_ref |
  type_ref   |
   binary | boolean | integer | logical | number | real | string   '>


<!ENTITY % qualifiable_factor '   const_e |
   pi |
   self |
   unknown  |
   constant_ref  |
    built_in_function |
    function_call |
    qualified_attribute_ref |
    attribute_ref |
    partial_entity_ref |
    population |
    parameter_ref |
    variable_ref |
    index_qualified_item '>
```

```
<!ENTITY % primary '  binary_literal |
  integer_literal |
  logical_literal |
  real_literal |
  string_literal  |
     const_e |
   pi |
   self |
   unknown  |
   constant_ref  |
    built_in_function |
    function_call |
    qualified_attribute_ref |
    attribute_ref |
    partial_entity_ref |
    population |
    parameter_ref |
    variable_ref |
    index_qualified_item  '>


<!ENTITY % simple_factor ' aggregate_initializer |
  entity_constructor |
  enumeration_reference |
  interval |
  query |
  not |
    binary_literal |
  integer_literal |
  logical_literal |
  real_literal |
  string_literal  |
     const_e |
   pi |
   self |
   unknown  |
   constant_ref  |
    built_in_function |
    function_call |
    qualified_attribute_ref |
    attribute_ref |
    partial_entity_ref |
    population |
    parameter_ref |
    variable_ref |
    index_qualified_item   '>
```

```
<!ENTITY % factor '  aggregate_initializer |
  entity_constructor |
  enumeration_reference |
  interval |
  query |
  not |
    binary_literal |
  integer_literal |
  logical_literal |
  real_literal |
  string_literal  |
     const_e |
   pi |
   self |
   unknown  |
   constant_ref  |
    built_in_function |
    function_call |
    qualified_attribute_ref |
    attribute_ref |
    partial_entity_ref |
    population |
    parameter_ref |
    variable_ref |
    index_qualified_item    |
  raise_to_power '>
```

```
<!ENTITY % term '   aggregate_initializer |
  entity_constructor |
  enumeration_reference |
  interval |
  query |
  not |
    binary_literal |
  integer_literal |
  logical_literal |
  real_literal |
  string_literal  |
     const_e |
   pi |
   self |
   unknown  |
   constant_ref  |
    built_in_function |
    function_call |
    qualified_attribute_ref |
    attribute_ref |
    partial_entity_ref |
    population |
    parameter_ref |
    variable_ref |
    index_qualified_item    |
  raise_to_power  |
   multiply |
   real_divide |
   integer_divide |
   mod |
   and |
   complex_entity_constructor  '>
```

```
<!ENTITY % simple_expression '    aggregate_initializer |
  entity_constructor |
  enumeration_reference |
  interval |
  query |
  not |
    binary_literal |
  integer_literal |
  logical_literal |
  real_literal |
  string_literal  |
     const_e |
   pi |
   self |
   unknown  |
   constant_ref  |
    built_in_function |
    function_call |
    qualified_attribute_ref |
    attribute_ref |
    partial_entity_ref |
    population |
    parameter_ref |
    variable_ref |
    index_qualified_item    |
  raise_to_power  |
   multiply |
   real_divide |
   integer_divide |
   mod |
   and |
   complex_entity_constructor   |
   add |
   subtract |
   or |
   xor   '>


<!ENTITY % numeric_expression_top ' integer_literal |
 numeric_expression '>


<!ENTITY % index '  integer_literal |
 numeric_expression  '>


<!ENTITY % width '  integer_literal |
 numeric_expression  '>


<!ELEMENT abs (documentation?, arg)>
            <!-- Used by: (built_in_function) -->
```

```
<!ELEMENT abstract_supertype_of (documentation?, (entity_ref |
supertype_one_of | supertype_and_or | supertype_and))>
            <!-- Used by: (entity_decl) -->


<!ELEMENT acos (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT add (documentation?, arg+)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT aggregate_initializer (documentation?, element_list)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT aggregate_source (documentation?, (aggregate_initializer |
entity_constructor | enumeration_reference | interval | query | not |
binary_literal | integer_literal | logical_literal | real_literal |
string_literal | const_e | pi | self | unknown | constant_ref |
built_in_function | function_call | qualified_attribute_ref | attribute_ref
| partial_entity_ref | population | parameter_ref | variable_ref |
index_qualified_item | raise_to_power | multiply | real_divide |
integer_divide | mod | and | complex_entity_constructor | add | subtract |
or | xor))>
            <!-- Used by: (query) -->


<!ELEMENT aggregate_type (documentation?, (aggregate_type |
general_array_type | general_bag_type | general_list_type |
general_set_type | generic_type | entity_ref | type_ref | binary | boolean
| integer | logical | number | real | string), (type_label_id |
type_label_ref)?)>
            <!-- Used by: (aggregate_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable var_formal_parameter) -->


<!ELEMENT algorithm_head (documentation?, declaration_block?,
constant_block?, local_variable_block?)>
            <!-- Used by: (function_decl procedure_decl) -->


<!ELEMENT alias_stmt (documentation?, variable_id, (parameter_ref |
variable_ref), qualifier?, statement_block)>
            <!-- Used by: (case_action otherwise statement_block) -->
```

```
<!ELEMENT and (documentation?, arg+)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT applies_to_entities (documentation?, entity_ref+)>
            <!-- Used by: (rule_decl) -->


<!ELEMENT arg (documentation?, (binary_literal | integer_literal |
logical_literal | real_literal | string_literal | expression))>
            <!-- Used by: (abs acos add and asin atan bLength
complex_entity_constructor cos entity_constructor equal exists exp format
function_call greater_than greater_than_or_equal hiBound hiIndex in insert
instance_equal instance_not_equal integer_divide length less_than
less_than_or_equal like loBound log log10 log2 loIndex mod multiply not
not_equal nvl odd or procedure_call_stmt raise_to_power real_divide remove
rolesOf sin sizeOf sqrt subtract tan typeOf usedIn value value_in
value_unique xor) -->


<!ELEMENT array_type (documentation?, index_spec, base_type, optional?,
unique?)>
            <!-- Used by: (base_type underlying_type) -->


<!ELEMENT asin (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT assignment_stmt (documentation?, (parameter_ref | variable_ref),
qualifier?, expression)>
            <!-- Used by: (case_action otherwise statement_block) -->


<!ELEMENT atan (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT attribute_id (#PCDATA)>
            <!-- Used by: (derived_attr explicit_attr inverse_attr) -->


<!ELEMENT attribute_ref (#PCDATA)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive inverse_attr
numeric_expression qualified_attribute qualified_attribute_ref qualifier
unique_rule) -->
```

```
<!ELEMENT bag_type (documentation?, bound_spec?, base_type)>
            <!-- Used by: (base_type underlying_type) -->


<!ELEMENT base_type (documentation?, (array_type | bag_type | list_type |
set_type | binary | boolean | integer | logical | number | real | string |
entity_ref | type_ref))>
            <!-- Used by: (array_type bag_type constant_decl derived_attr
explicit_attr list_type set_type) -->


<!ELEMENT binary (documentation?, width_spec?)>
            <!-- Used by: (aggregate_type base_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable underlying_type var_formal_parameter) -->


<!ELEMENT binary_literal (#PCDATA)>
            <!-- Used by: (aggregate_source arg expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT bLength (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT boolean EMPTY>
            <!-- Used by: (aggregate_type base_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable underlying_type var_formal_parameter) -->


<!ELEMENT bound_spec (documentation?, lower_bound, upper_bound)>
            <!-- Used by: (bag_type general_array_type general_bag_type
general_list_type general_set_type inverse_bag inverse_set list_type
set_type) -->


<!ELEMENT built_in_function (abs | acos | asin | atan | bLength | cos |
exists | exp | format | hiBound | hiIndex | length | loBound | loIndex |
log | log2 | log10 | nvl | odd | rolesOf | sin | sizeOf | sqrt | tan |
typeOf | usedIn | value | value_in | value_unique)>
            <!-- Used by: (aggregate_source expression index_qualified_item
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT case_action (documentation?, case_label, (alias_stmt |
assignment_stmt | case_stmt | compound_stmt | escape_stmt | if_stmt |
null_stmt | procedure_call_stmt | repeat_stmt | return_stmt | skip_stmt))>
            <!-- Used by: (case_stmt) -->
```

```
<!ELEMENT case_label (documentation?, expression+)>
            <!-- Used by: (case_action) -->


<!ELEMENT case_stmt (documentation?, expression, case_action*, otherwise?)>
            <!-- Used by: (case_action otherwise statement_block) -->


<!ELEMENT complex_entity_constructor (documentation?, arg+)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT compound_stmt (documentation?, statement_block)>
            <!-- Used by: (case_action otherwise statement_block) -->


<!ELEMENT constant_block (documentation?, constant_decl*)>
            <!-- Used by: (algorithm_head schema_decl) -->


<!ELEMENT constant_decl (documentation?, constant_id, base_type,
expression)>
            <!-- Used by: (constant_block) -->


<!ELEMENT constant_id (#PCDATA)>
            <!-- Used by: (constant_decl constant_import) -->


<!ELEMENT constant_import (documentation?, constant_id, constant_ref)>
            <!-- Used by: (reference_from) -->


<!ELEMENT constant_ref (#PCDATA)>
            <!-- Used by: (aggregate_source constant_import expression
index_qualified_item interval_high_exclusive interval_high_inclusive
interval_item interval_low_exclusive interval_low_inclusive
numeric_expression) -->


<!ELEMENT const_e EMPTY>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT cos (documentation?, arg)>
            <!-- Used by: (built_in_function) -->
```

```
<!ELEMENT declaration_block (documentation?, (entity_decl | function_decl |
procedure_decl | type_decl)*)>
            <!-- Used by: (algorithm_head) -->


<!ELEMENT derived_attr (documentation?, (attribute_id |
qualified_attribute), base_type, expression)>
            <!-- Used by: (derive_clause) -->


<!ELEMENT derive_clause (documentation?, derived_attr+)>
            <!-- Used by: (entity_decl) -->


<!ELEMENT documentation (#PCDATA)*>
            <!-- Used by: (abs abstract_supertype_of acos add
aggregate_initializer aggregate_source aggregate_type algorithm_head
alias_stmt and applies_to_entities arg array_type asin assignment_stmt atan
bag_type base_type binary bLength bound_spec case_action case_label
case_stmt complex_entity_constructor compound_stmt constant_block
constant_decl constant_import cos declaration_block derived_attr
derive_clause domain_rule element_item element_list entity_constructor
entity_decl entity_import enumeration enumeration_reference equal exists
exp explicit_attr explicit_attr_block expression express_driven_data
formal_parameter formal_parameter_block format function_call function_decl
function_import general_array_type general_bag_type general_list_type
general_set_type generic_type greater_than greater_than_or_equal hiBound
high_index hiIndex if_stmt in increment increment_control
index_qualified_item index_qualifier index_spec insert instance_equal
instance_not_equal integer_divide interface_specification_block interval
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive inverse_attr inverse_bag
inverse_clause inverse_set length less_than less_than_or_equal like
list_type loBound local_variable local_variable_block log log10 log2
logical_expression logical_literal loIndex lower_bound low_index mod
multiply not not_equal numeric_expression nvl odd or otherwise
partial_entity_ref population precision_spec procedure_call_stmt
procedure_decl procedure_formal_parameter_block procedure_import
qualified_attribute qualified_attribute_ref qualifier query raise_to_power
real real_divide reference_from remove repeat_control repeat_stmt
repetition return_stmt rolesOf rule_decl schema_decl select set_type sin
sizeOf sqrt statement_block string subtract subtype_of supertype_and
supertype_and_or supertype_of supertype_one_of tan typeOf type_decl
type_import underlying_type unique_clause unique_rule until upper_bound
usedIn use_from value value_in value_unique var_formal_parameter
where_clause while width_spec xor) -->


<!ELEMENT domain_rule (documentation?, label?, expression)>
            <!-- Used by: (where_clause) -->
```

```
<!ELEMENT element_item (documentation?, expression, repetition?)>
            <!-- Used by: (element_list) -->


<!ELEMENT element_list (documentation?, element_item*)>
            <!-- Used by: (aggregate_initializer) -->


<!ELEMENT entity_constructor (documentation?, entity_ref, arg*)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT entity_decl (documentation?, entity_id, (abstract_supertype_of |
supertype_of)?, subtype_of?, explicit_attr_block?, derive_clause?,
inverse_clause?, unique_clause?, where_clause?)>
            <!-- Used by: (declaration_block schema_decl) -->


<!ELEMENT entity_id (#PCDATA)>
            <!-- Used by: (entity_decl entity_import) -->


<!ELEMENT entity_import (documentation?, entity_id, entity_ref)>
            <!-- Used by: (reference_from use_from) -->


<!ELEMENT entity_ref (#PCDATA)>
            <!-- Used by: (abstract_supertype_of aggregate_type
applies_to_entities base_type entity_constructor entity_import
formal_parameter function_return_type general_array_type general_bag_type
general_list_type general_set_type inverse_attr local_variable
partial_entity_ref population qualified_attribute qualifier select
subtype_of supertype_and supertype_and_or supertype_of supertype_one_of
var_formal_parameter) -->


<!ELEMENT enumeration (documentation?, enumeration_id+)>
            <!-- Used by: (underlying_type) -->


<!ELEMENT enumeration_id (#PCDATA)>
            <!-- Used by: (enumeration) -->


<!ELEMENT enumeration_ref (#PCDATA)>
            <!-- Used by: (enumeration_reference) -->
```

```
<!ELEMENT enumeration_reference (documentation?, type_ref?,
enumeration_ref)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT equal (documentation?, arg+)>
            <!-- Used by: (expression) -->


<!ELEMENT escape_stmt EMPTY>
            <!-- Used by: (case_action otherwise statement_block) -->


<!ELEMENT exists (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT exp (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT explicit_attr (documentation?, (attribute_id |
qualified_attribute)+, optional?, base_type)>
            <!-- Used by: (explicit_attr_block) -->


<!ELEMENT explicit_attr_block (documentation?, explicit_attr+)>
            <!-- Used by: (entity_decl) -->


<!ELEMENT expression (documentation?, (less_than | greater_than |
less_than_or_equal | greater_than_or_equal | not_equal | equal |
instance_not_equal | instance_equal | in | like | aggregate_initializer |
entity_constructor | enumeration_reference | interval | query | not |
binary_literal | integer_literal | logical_literal | real_literal |
string_literal | const_e | pi | self | unknown | constant_ref |
built_in_function | function_call | qualified_attribute_ref | attribute_ref
| partial_entity_ref | population | parameter_ref | variable_ref |
index_qualified_item | raise_to_power | multiply | real_divide |
integer_divide | mod | and | complex_entity_constructor | add | subtract |
or | xor))>
            <!-- Used by: (arg assignment_stmt case_label case_stmt
constant_decl derived_attr domain_rule element_item local_variable
logical_expression return_stmt) -->


<!ELEMENT express_driven_data (documentation?, (schema_decl | data)*)>
            <!-- Used by: NIL -->
```

```
<!ELEMENT false EMPTY>
            <!-- Used by: (logical_literal) -->


<!ELEMENT fixed EMPTY>
            <!-- Used by: (width_spec) -->


<!ELEMENT formal_parameter (documentation?, parameter_id+, (aggregate_type
| general_array_type | general_bag_type | general_list_type |
general_set_type | generic_type | entity_ref | type_ref | binary | boolean
| integer | logical | number | real | string))>
            <!-- Used by: (formal_parameter_block
procedure_formal_parameter_block) -->


<!ELEMENT formal_parameter_block (documentation?, formal_parameter*)>
            <!-- Used by: (function_decl) -->


<!ELEMENT format (documentation?, arg, string)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT function_call (documentation?, function_ref, arg*)>
            <!-- Used by: (aggregate_source expression index_qualified_item
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT function_decl (documentation?, function_id,
formal_parameter_block?, function_return_type, algorithm_head?,
statement_block)>
            <!-- Used by: (declaration_block schema_decl) -->


<!ELEMENT function_id (#PCDATA)>
            <!-- Used by: (function_decl function_import) -->


<!ELEMENT function_import (documentation?, function_id, function_ref)>
            <!-- Used by: (reference_from) -->


<!ELEMENT function_ref (#PCDATA)>
            <!-- Used by: (function_call function_import) -->
```

```
<!ELEMENT function_return_type (aggregate_type | general_array_type |
general_bag_type | general_list_type | general_set_type | generic_type |
entity_ref | type_ref | binary | boolean | integer | logical | number |
real | string)>
            <!-- Used by: (function_decl) -->


<!ELEMENT general_array_type (documentation?, (aggregate_type |
general_array_type | general_bag_type | general_list_type |
general_set_type | generic_type | entity_ref | type_ref | binary | boolean
| integer | logical | number | real | string), bound_spec?, optional?,
unique?)>
            <!-- Used by: (aggregate_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable var_formal_parameter) -->


<!ELEMENT general_bag_type (documentation?, (aggregate_type |
general_array_type | general_bag_type | general_list_type |
general_set_type | generic_type | entity_ref | type_ref | binary | boolean
| integer | logical | number | real | string), bound_spec?)>
            <!-- Used by: (aggregate_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable var_formal_parameter) -->


<!ELEMENT general_list_type (documentation?, (aggregate_type |
general_array_type | general_bag_type | general_list_type |
general_set_type | generic_type | entity_ref | type_ref | binary | boolean
| integer | logical | number | real | string), bound_spec?, unique?)>
            <!-- Used by: (aggregate_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable var_formal_parameter) -->


<!ELEMENT general_set_type (documentation?, (aggregate_type |
general_array_type | general_bag_type | general_list_type |
general_set_type | generic_type | entity_ref | type_ref | binary | boolean
| integer | logical | number | real | string), bound_spec?)>
            <!-- Used by: (aggregate_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable var_formal_parameter) -->


<!ELEMENT generic_type (documentation?, (type_label_id | type_label_ref)?)>
            <!-- Used by: (aggregate_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable var_formal_parameter) -->


<!ELEMENT greater_than (documentation?, arg+)>
            <!-- Used by: (expression) -->
```

```
<!ELEMENT greater_than_or_equal (documentation?, arg+)>
            <!-- Used by: (expression) -->


<!ELEMENT hiBound (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT high_index (documentation?, (integer_literal |
numeric_expression))>
            <!-- Used by: (index_qualifier index_spec) -->


<!ELEMENT hiIndex (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT if_stmt (documentation?, logical_expression, statement_block,
statement_block?)>
            <!-- Used by: (case_action otherwise statement_block) -->


<!ELEMENT import_all EMPTY>
            <!-- Used by: (reference_from use_from) -->


<!ELEMENT in (documentation?, arg, arg)>
            <!-- Used by: (expression) -->


<!ELEMENT increment (documentation?, (integer_literal |
numeric_expression))>
            <!-- Used by: (increment_control) -->


<!ELEMENT increment_control (documentation?, variable_id, lower_bound,
upper_bound, increment?)>
            <!-- Used by: (repeat_control) -->


<!ELEMENT indeterminate EMPTY>
            <!-- Used by: (upper_bound) -->


<!ELEMENT index_qualified_item (documentation?, (self | function_call |
built_in_function | parameter_ref | constant_ref | variable_ref |
population), index_qualifier)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->
```

```
<!ELEMENT index_qualifier (documentation?, low_index, high_index?)>
            <!-- Used by: (index_qualified_item qualified_attribute_ref
qualifier) -->


<!ELEMENT index_spec (documentation?, low_index, high_index?)>
            <!-- Used by: (array_type) -->


<!ELEMENT insert (documentation?, arg, arg, (integer_literal |
numeric_expression))>
            <!-- Used by: (procedure_call_stmt) -->


<!ELEMENT instance_equal (documentation?, arg+)>
            <!-- Used by: (expression) -->


<!ELEMENT instance_not_equal (documentation?, arg+)>
            <!-- Used by: (expression) -->


<!ELEMENT integer EMPTY>
            <!-- Used by: (aggregate_type base_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable underlying_type var_formal_parameter) -->


<!ELEMENT integer_divide (documentation?, arg, arg)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT integer_literal (#PCDATA)>
            <!-- Used by: (aggregate_source arg expression high_index
increment insert interval_high_exclusive interval_high_inclusive
interval_item interval_low_exclusive interval_low_inclusive lower_bound
low_index numeric_expression precision_spec remove repetition upper_bound
width_spec) -->


<!ELEMENT interface_specification_block (documentation?, (reference_from |
use_from)+)>
            <!-- Used by: (schema_decl) -->
```

```
<!ELEMENT interval (documentation?, (interval_low_inclusive |
interval_low_exclusive), interval_item, (interval_high_inclusive |
interval_high_exclusive))>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT interval_high_exclusive (documentation?, (aggregate_initializer |
entity_constructor | enumeration_reference | interval | query | not |
binary_literal | integer_literal | logical_literal | real_literal |
string_literal | const_e | pi | self | unknown | constant_ref |
built_in_function | function_call | qualified_attribute_ref | attribute_ref
| partial_entity_ref | population | parameter_ref | variable_ref |
index_qualified_item | raise_to_power | multiply | real_divide |
integer_divide | mod | and | complex_entity_constructor | add | subtract |
or | xor))>
            <!-- Used by: (interval) -->


<!ELEMENT interval_high_inclusive (documentation?, (aggregate_initializer |
entity_constructor | enumeration_reference | interval | query | not |
binary_literal | integer_literal | logical_literal | real_literal |
string_literal | const_e | pi | self | unknown | constant_ref |
built_in_function | function_call | qualified_attribute_ref | attribute_ref
| partial_entity_ref | population | parameter_ref | variable_ref |
index_qualified_item | raise_to_power | multiply | real_divide |
integer_divide | mod | and | complex_entity_constructor | add | subtract |
or | xor))>
            <!-- Used by: (interval) -->


<!ELEMENT interval_item (documentation?, (aggregate_initializer |
entity_constructor | enumeration_reference | interval | query | not |
binary_literal | integer_literal | logical_literal | real_literal |
string_literal | const_e | pi | self | unknown | constant_ref |
built_in_function | function_call | qualified_attribute_ref | attribute_ref
| partial_entity_ref | population | parameter_ref | variable_ref |
index_qualified_item | raise_to_power | multiply | real_divide |
integer_divide | mod | and | complex_entity_constructor | add | subtract |
or | xor))>
            <!-- Used by: (interval) -->
```

```
<!ELEMENT interval_low_exclusive (documentation?, (aggregate_initializer |
entity_constructor | enumeration_reference | interval | query | not |
binary_literal | integer_literal | logical_literal | real_literal |
string_literal | const_e | pi | self | unknown | constant_ref |
built_in_function | function_call | qualified_attribute_ref | attribute_ref
| partial_entity_ref | population | parameter_ref | variable_ref |
index_qualified_item | raise_to_power | multiply | real_divide |
integer_divide | mod | and | complex_entity_constructor | add | subtract |
or | xor))>
            <!-- Used by: (interval) -->


<!ELEMENT interval_low_inclusive (documentation?, (aggregate_initializer |
entity_constructor | enumeration_reference | interval | query | not |
binary_literal | integer_literal | logical_literal | real_literal |
string_literal | const_e | pi | self | unknown | constant_ref |
built_in_function | function_call | qualified_attribute_ref | attribute_ref
| partial_entity_ref | population | parameter_ref | variable_ref |
index_qualified_item | raise_to_power | multiply | real_divide |
integer_divide | mod | and | complex_entity_constructor | add | subtract |
or | xor))>
            <!-- Used by: (interval) -->


<!ELEMENT inverse_attr (documentation?, (attribute_id |
qualified_attribute), entity_ref, attribute_ref, (inverse_set |
inverse_bag)?)>
            <!-- Used by: (inverse_clause) -->


<!ELEMENT inverse_bag (documentation?, bound_spec?)>
            <!-- Used by: (inverse_attr) -->


<!ELEMENT inverse_clause (documentation?, inverse_attr+)>
            <!-- Used by: (entity_decl) -->


<!ELEMENT inverse_set (documentation?, bound_spec?)>
            <!-- Used by: (inverse_attr) -->


<!ELEMENT label (#PCDATA)>
            <!-- Used by: (domain_rule unique_rule) -->


<!ELEMENT length (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT less_than (documentation?, arg+)>
            <!-- Used by: (expression) -->
```

```
<!ELEMENT less_than_or_equal (documentation?, arg+)>
            <!-- Used by: (expression) -->


<!ELEMENT like (documentation?, arg, arg)>
            <!-- Used by: (expression) -->


<!ELEMENT list_type (documentation?, bound_spec?, base_type, unique?)>
            <!-- Used by: (base_type underlying_type) -->


<!ELEMENT loBound (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT local_variable (documentation?, variable_id+, (aggregate_type |
general_array_type | general_bag_type | general_list_type |
general_set_type | generic_type | entity_ref | type_ref | binary | boolean
| integer | logical | number | real | string), expression)>
            <!-- Used by: (local_variable_block) -->


<!ELEMENT local_variable_block (documentation?, local_variable*)>
            <!-- Used by: (algorithm_head) -->


<!ELEMENT log (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT log10 (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT log2 (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT logical EMPTY>
            <!-- Used by: (aggregate_type base_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable underlying_type var_formal_parameter) -->


<!ELEMENT logical_expression (documentation?, expression)>
            <!-- Used by: (if_stmt query until while) -->
```

```
<!ELEMENT logical_literal (documentation?, (false | true | unknown))>
            <!-- Used by: (aggregate_source arg expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT loIndex (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT lower_bound (documentation?, (integer_literal |
numeric_expression))>
            <!-- Used by: (bound_spec increment_control) -->


<!ELEMENT low_index (documentation?, (integer_literal |
numeric_expression))>
            <!-- Used by: (index_qualifier index_spec) -->


<!ELEMENT mod (documentation?, arg, arg)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT multiply (documentation?, arg+)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT not (documentation?, arg)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT not_equal (documentation?, arg+)>
            <!-- Used by: (expression) -->


<!ELEMENT null_stmt EMPTY>
            <!-- Used by: (case_action otherwise statement_block) -->


<!ELEMENT number EMPTY>
            <!-- Used by: (aggregate_type base_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable underlying_type var_formal_parameter) -->
```

```
<!ELEMENT numeric_expression (documentation?, (aggregate_initializer |
entity_constructor | enumeration_reference | interval | query | not |
binary_literal | integer_literal | logical_literal | real_literal |
string_literal | const_e | pi | self | unknown | constant_ref |
built_in_function | function_call | qualified_attribute_ref | attribute_ref
| partial_entity_ref | population | parameter_ref | variable_ref |
index_qualified_item | raise_to_power | multiply | real_divide |
integer_divide | mod | and | complex_entity_constructor | add | subtract |
or | xor))>
            <!-- Used by: (high_index increment insert lower_bound
low_index precision_spec remove repetition upper_bound width_spec) -->


<!ELEMENT nvl (documentation?, arg, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT odd (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT optional EMPTY>
            <!-- Used by: (array_type explicit_attr general_array_type) -->


<!ELEMENT or (documentation?, arg+)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT otherwise (documentation?, (alias_stmt | assignment_stmt |
case_stmt | compound_stmt | escape_stmt | if_stmt | null_stmt |
procedure_call_stmt | repeat_stmt | return_stmt | skip_stmt))>
            <!-- Used by: (case_stmt) -->


<!ELEMENT parameter_id (#PCDATA)>
            <!-- Used by: (formal_parameter var_formal_parameter) -->


<!ELEMENT parameter_ref (#PCDATA)>
            <!-- Used by: (aggregate_source alias_stmt assignment_stmt
expression index_qualified_item interval_high_exclusive
interval_high_inclusive interval_item interval_low_exclusive
interval_low_inclusive numeric_expression qualified_attribute_ref) -->
```

```
<!ELEMENT partial_entity_ref (documentation?, entity_ref)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression
qualified_attribute_ref) -->


<!ELEMENT pi EMPTY>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT population (documentation?, entity_ref)>
            <!-- Used by: (aggregate_source expression index_qualified_item
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT precision_spec (documentation?, (integer_literal |
numeric_expression))>
            <!-- Used by: (real) -->


<!ELEMENT procedure_call_stmt (documentation?, ((insert | remove) |
(procedure_ref, arg*)))>
            <!-- Used by: (case_action otherwise statement_block) -->


<!ELEMENT procedure_decl (documentation?, procedure_id,
procedure_formal_parameter_block?, algorithm_head?, statement_block?)>
            <!-- Used by: (declaration_block schema_decl) -->


<!ELEMENT procedure_formal_parameter_block (documentation?,
(formal_parameter | var_formal_parameter)*)>
            <!-- Used by: (procedure_decl) -->


<!ELEMENT procedure_id (#PCDATA)>
            <!-- Used by: (procedure_decl procedure_import) -->


<!ELEMENT procedure_import (documentation?, procedure_id, procedure_ref)>
            <!-- Used by: (reference_from) -->


<!ELEMENT procedure_ref (#PCDATA)>
            <!-- Used by: (procedure_call_stmt procedure_import) -->
```

```
<!ELEMENT qualified_attribute (documentation?, entity_ref, attribute_ref)>
            <!-- Used by: (derived_attr explicit_attr inverse_attr
unique_rule) -->


<!ELEMENT qualified_attribute_ref (documentation?, attribute_ref,
(index_qualifier | attribute_ref | qualified_attribute_ref | variable_ref |
parameter_ref | partial_entity_ref))>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression
qualified_attribute_ref) -->


<!ELEMENT qualifier (documentation?, (attribute_ref | entity_ref |
index_qualifier), qualifier?)>
            <!-- Used by: (alias_stmt assignment_stmt qualifier) -->


<!ELEMENT query (documentation?, variable_id, (aggregate_source,
logical_expression))>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT raise_to_power (documentation?, arg, arg)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT real (documentation?, precision_spec?)>
            <!-- Used by: (aggregate_type base_type formal_parameter
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable underlying_type var_formal_parameter) -->


<!ELEMENT real_divide (documentation?, arg, arg)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT real_literal (#PCDATA)>
            <!-- Used by: (aggregate_source arg expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->
```

```
<!ELEMENT reference_from (documentation?, schema_ref, (import_all |
(constant_import | entity_import | function_import | procedure_import |
type_import)+))>
            <!-- Used by: (interface_specification_block) -->


<!ELEMENT remove (documentation?, arg, (integer_literal |
numeric_expression))>
            <!-- Used by: (procedure_call_stmt) -->


<!ELEMENT repeat_control (documentation?, increment_control, while?,
until?)>
            <!-- Used by: (repeat_stmt) -->


<!ELEMENT repeat_stmt (documentation?, repeat_control, statement_block)>
            <!-- Used by: (case_action otherwise statement_block) -->


<!ELEMENT repetition (documentation?, (integer_literal |
numeric_expression))>
            <!-- Used by: (element_item) -->


<!ELEMENT return_stmt (documentation?, expression?)>
            <!-- Used by: (case_action otherwise statement_block) -->


<!ELEMENT rolesOf (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT rule_decl (documentation?, rule_id, applies_to_entities,
where_clause)>
            <!-- Used by: (schema_decl) -->


<!ELEMENT rule_id (#PCDATA)>
            <!-- Used by: (rule_decl) -->


<!ELEMENT schema_decl (documentation?, schema_id,
interface_specification_block?, constant_block?, (entity_decl |
function_decl | procedure_decl | type_decl | rule_decl)*)>
            <!-- Used by: (express_driven_data) -->


<!ELEMENT schema_id (#PCDATA)>
            <!-- Used by: (schema_decl) -->
```

```
<!ELEMENT schema_ref (#PCDATA)>
            <!-- Used by: (reference_from use_from) -->


<!ELEMENT select (documentation?, (entity_ref | type_ref)+)>
            <!-- Used by: (underlying_type) -->


<!ELEMENT self EMPTY>
            <!-- Used by: (aggregate_source expression index_qualified_item
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT set_type (documentation?, bound_spec?, base_type)>
            <!-- Used by: (base_type underlying_type) -->


<!ELEMENT sin (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT sizeOf (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT skip_stmt EMPTY>
            <!-- Used by: (case_action otherwise statement_block) -->


<!ELEMENT sqrt (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT statement_block (documentation?, (alias_stmt | assignment_stmt |
case_stmt | compound_stmt | escape_stmt | if_stmt | null_stmt |
procedure_call_stmt | repeat_stmt | return_stmt | skip_stmt)+)>
            <!-- Used by: (alias_stmt compound_stmt function_decl if_stmt
procedure_decl repeat_stmt) -->


<!ELEMENT string (documentation?, width_spec?)>
            <!-- Used by: (aggregate_type base_type formal_parameter format
function_return_type general_array_type general_bag_type general_list_type
general_set_type local_variable underlying_type var_formal_parameter) -->


<!ELEMENT string_literal (#PCDATA)>
            <!-- Used by: (aggregate_source arg expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->
```

```
<!ELEMENT subtract (documentation?, arg, arg)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->


<!ELEMENT subtype_of (documentation?, entity_ref+)>
            <!-- Used by: (entity_decl) -->


<!ELEMENT supertype_and (documentation?, (entity_ref | supertype_one_of |
supertype_and_or | supertype_and)+)>
            <!-- Used by: (abstract_supertype_of supertype_and
supertype_and_or supertype_of supertype_one_of) -->


<!ELEMENT supertype_and_or (documentation?, (entity_ref | supertype_one_of
| supertype_and_or | supertype_and)+)>
            <!-- Used by: (abstract_supertype_of supertype_and
supertype_and_or supertype_of supertype_one_of) -->


<!ELEMENT supertype_of (documentation?, (entity_ref | supertype_one_of |
supertype_and_or | supertype_and))>
            <!-- Used by: (entity_decl) -->


<!ELEMENT supertype_one_of (documentation?, (entity_ref | supertype_one_of
| supertype_and_or | supertype_and)+)>
            <!-- Used by: (abstract_supertype_of supertype_and
supertype_and_or supertype_of supertype_one_of) -->


<!ELEMENT tan (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT TBA EMPTY>
            <!-- Used by: (data) -->


<!ELEMENT true EMPTY>
            <!-- Used by: (logical_literal) -->


<!ELEMENT typeOf (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT type_decl (documentation?, type_id, underlying_type,
where_clause?)>
            <!-- Used by: (declaration_block schema_decl) -->
```

```
<!ELEMENT type_id (#PCDATA)>
            <!-- Used by: (type_decl type_import) -->


<!ELEMENT type_import (documentation?, type_id, type_ref)>
            <!-- Used by: (reference_from use_from) -->


<!ELEMENT type_label_id (#PCDATA)>
            <!-- Used by: (aggregate_type generic_type) -->


<!ELEMENT type_label_ref (#PCDATA)>
            <!-- Used by: (aggregate_type generic_type) -->


<!ELEMENT type_ref (#PCDATA)>
            <!-- Used by: (aggregate_type base_type enumeration_reference
formal_parameter function_return_type general_array_type general_bag_type
general_list_type general_set_type local_variable select type_import
underlying_type var_formal_parameter) -->


<!ELEMENT underlying_type (documentation?, (enumeration | select |
array_type | bag_type | list_type | set_type | binary | boolean | integer |
logical | number | real | string | type_ref))>
            <!-- Used by: (type_decl) -->


<!ELEMENT unique EMPTY>
            <!-- Used by: (array_type general_array_type general_list_type
list_type) -->


<!ELEMENT unique_clause (documentation?, unique_rule+)>
            <!-- Used by: (entity_decl) -->


<!ELEMENT unique_rule (documentation?, label?, (attribute_ref |
qualified_attribute)+)>
            <!-- Used by: (unique_clause) -->


<!ELEMENT unknown EMPTY>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive logical_literal
numeric_expression) -->


<!ELEMENT until (documentation?, logical_expression)>
            <!-- Used by: (repeat_control) -->
```

```
<!ELEMENT upper_bound (documentation?, (indeterminate | integer_literal |
numeric_expression))>
            <!-- Used by: (bound_spec increment_control) -->


<!ELEMENT usedIn (documentation?, arg, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT use_from (documentation?, schema_ref, (import_all |
(entity_import | type_import)+))>
            <!-- Used by: (interface_specification_block) -->


<!ELEMENT value (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT value_in (documentation?, arg, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT value_unique (documentation?, arg)>
            <!-- Used by: (built_in_function) -->


<!ELEMENT variable_id (#PCDATA)>
            <!-- Used by: (alias_stmt increment_control local_variable
query) -->


<!ELEMENT variable_ref (#PCDATA)>
            <!-- Used by: (aggregate_source alias_stmt assignment_stmt
expression index_qualified_item interval_high_exclusive
interval_high_inclusive interval_item interval_low_exclusive
interval_low_inclusive numeric_expression qualified_attribute_ref) -->


<!ELEMENT var_formal_parameter (documentation?, parameter_id,
(aggregate_type | general_array_type | general_bag_type | general_list_type
| general_set_type | generic_type | entity_ref | type_ref | binary |
boolean | integer | logical | number | real | string))>
            <!-- Used by: (procedure_formal_parameter_block) -->


<!ELEMENT where_clause (documentation?, domain_rule+)>
            <!-- Used by: (entity_decl rule_decl type_decl) -->


<!ELEMENT while (documentation?, logical_expression)>
            <!-- Used by: (repeat_control) -->
```

```
<!ELEMENT width_spec (documentation?, (integer_literal |
numeric_expression), fixed?)>
            <!-- Used by: (binary string) -->


<!ELEMENT xor (documentation?, arg, arg)>
            <!-- Used by: (aggregate_source expression
interval_high_exclusive interval_high_inclusive interval_item
interval_low_exclusive interval_low_inclusive numeric_expression) -->
```

Annex B
(normative)

Late Bound DTD elements for EXPRESS-driven data

The following DTD deals with instance data corresponding to an EXPRESS schema. It has been subject to change and may not be entirely in sync with the DTD given in annex A.

The following change information to be removed prior to publication as a CD.

```
<!-- Late binding data section of XML based on Eliot Kimber's
original (defined at the San Francisco meeting).
Suitably improved/corrupted by Robin La Fontaine
Version 1, 7 Sept 1999

Version 2, 11 October 1999: Updated using Eliot Kimber's
architectural DTD for late-bound data which was developed at the
Early-binding workshop at Long Beach, CA, in September 1999.

Version 3: intermediate

Version 4: 19 Oct 1999: Changed in order to allow early to late
mapping with architectures. This meant that some of the sub-element
content had to be brought up into xml-attributes in order to be able
to do the mapping.

Version 5: 21 Oct 1999: Changed entity instance IDs and refs to be
just one type rather than distinguish between
partial/complex/nested/nested_subtype which was unnecessarily
complicated. Also made entity_literal into an xml-entity to get rid
of an unnecessary layer of nesting in the data.
-->

<!--==============================================
    Items brought over from main language schema
    ==============================================
(I know there are other ways to do this, but this is the simplest
for the software I am using!) -->

<!ENTITY % literal ' binary_literal |
  integer_literal |
  logical_literal |
  real_literal |
  string_literal '>

<!ELEMENT enumeration_ref (#PCDATA)>
<!ELEMENT string_literal (#PCDATA)>
<!ELEMENT integer_literal (#PCDATA)>
```

```
<!ELEMENT real_literal (#PCDATA)>
<!ELEMENT binary_literal (#PCDATA)>
<!ELEMENT logical_literal (false | true | unknown)>
<!ELEMENT false EMPTY>
<!ELEMENT true EMPTY>
<!ELEMENT unknown EMPTY>

<!-- data is brought over and modified here -->
<!ELEMENT data
  (schema_instance)
>

<!ATTLIST data
  data_id ID #REQUIRED>

<!--==============================================
    Organizing Parameter Entities
    ==============================================-->

<!ENTITY % entity_instance
  "nested_complex_entity_instance |
   flat_complex_entity_instance"
>

<!ENTITY % entity_literal
  "%entity_instance; |
   entity_instance_ref"
>


<!ENTITY % simple_value
  "%literal;

   bag_literal |
   list_literal |
   set_literal |
   array_literal |

   type_literal |
   %entity_literal;"
>

<!ENTITY % aggregate-contents
  "binary_literal* |
   integer_literal* |
   logical_literal* |
   real_literal* |
   string_literal* |

   bag_literal* |
   list_literal* |
   set_literal* |
   array_literal* |

   type_literal* |
   (%entity_literal;)*"
>


<!ENTITY % schema-ref-att
  "express_schema_name NMTOKEN #REQUIRED">

<!ENTITY % schema-ref-att-opt
  "express_schema_name NMTOKEN #IMPLIED">

<!ENTITY % entity-ref-att
  "express_entity_name NMTOKEN #REQUIRED">

<!ENTITY % type-ref-att
  "express_type_name NMTOKEN #REQUIRED">

<!ENTITY % attribute-ref-att
  "express_attribute_name NMTOKEN #REQUIRED">

<!--============================================
    Main section of DTD
    ============================================-->

<!ELEMENT ISO-10303-data
  (data+)
>

<!ELEMENT schema_instance
  (constant_instances,
   non_constant_instances)
>

<!ATTLIST schema_instance
  %schema-ref-att;>

<!ELEMENT constant_instances
  (%entity_instance;)*
>

<!ELEMENT non_constant_instances
  (%entity_instance;)*
>

<!ELEMENT flat_complex_entity_instance
  (partial_entity_instance+)
>
<!ATTLIST flat_complex_entity_instance
  entity_instance_id ID #REQUIRED>

<!ELEMENT nested_complex_entity_instance
  (attribute_instance*,
   nested_complex_entity_instance_subtype*)
>

<!ATTLIST nested_complex_entity_instance
```

```
  entity_instance_id ID #REQUIRED>

<!ATTLIST nested_complex_entity_instance
  %entity-ref-att;
  %schema-ref-att-opt;>

<!ELEMENT nested_complex_entity_instance_subtype
  (attribute_instance*,
   nested_complex_entity_instance_subtype*)
>

<!ATTLIST nested_complex_entity_instance_subtype
  %entity-ref-att;
  %schema-ref-att-opt;
  entity_instance_id ID #IMPLIED
>

<!ELEMENT partial_entity_instance
  (attribute_instance*)
>

<!ATTLIST partial_entity_instance
  %entity-ref-att;
  %schema-ref-att-opt;
  entity_instance_id ID #IMPLIED
>

<!ELEMENT attribute_instance
  (%simple_value;)
>

<!ATTLIST attribute_instance
  %attribute-ref-att;
>


<!NOTATION uuencode PUBLIC "uuencoded data" >
<!NOTATION mime     PUBLIC "mime encoded data" >
<!NOTATION base64   PUBLIC "base-64 encoded data" >

<!ATTLIST binary_literal
  notation
    NOTATION
    (uuencode |
     mime |
     base64)
    "uuencode"
>

<!ELEMENT set_literal
  (%aggregate-contents;)
>

<!ELEMENT list_literal
  (%aggregate-contents;)
```

```
>

<!ELEMENT bag_literal
  (%aggregate-contents;)
>

<!ELEMENT array_literal
  (%aggregate-contents;)
>

<!ELEMENT type_literal
  (%simple_value; | enumeration_ref)
>

<!ATTLIST type_literal
  %type-ref-att;
  %schema-ref-att-opt;
>

<!ELEMENT entity_instance_ref
  EMPTY
>

<!ATTLIST entity_instance_ref
  entity_instance_idref IDREF #REQUIRED>
```

Annex C(normative)

Late Bound DTD correspondence with EXPRESS syntax

Table C.1 shows the correspondence between the syntax of EXPRESS (as specified in ISO 10303-11) to elements used in the late bound DTD.

Normative because this defines the mapping that shall be used for each EXPRESS element. Columns of tables should be labelled accordingly to make sure not restating EXPRESS syntax.

In table C.1 the EXPRESS syntax is reproduced from ISO10303-11 and the XML DTD element definitions are reproduced from annex A of this part of ISO 10303. In both cases table C.1 is not the primary definition of the respective syntax. Where there are entries in both the EXPRESS syntax column and the XML DTD element definitions, table C.1 defines the correspondence that shall be followed in encoding EXPRESS as XML.

Note: The comment column is presented here as part of development of this part of ISO 10303. It probably should be replaced by footnotes or end notes if still needed.

Table C.1 – Late Bound DTD correspondence with EXPRESS syntax

EXPRESS syntax Ref.	**Express Syntax**	**XML DTD element definitions**	**Comment**
		`<!ELEMENT express_driven_data (%doc;, (schema_decl | data)*) >`	Top level keyword. Will need ‘admin’ data added as needed.
		`<!ELEMENT import_all EMPTY>`	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
		`<!ELEMENT data (TBA)>`	

		`<!ENTITY % doc “documentation?”>`
		`<!ELEMENT documentation`
		`  (#PCDATA)*`
		`>`	

		`<!ELEMENT TBA EMPTY>`	

		`<!ELEMENT less_than`
		`  (%doc;, arg+)`
		`>`
		`<!ELEMENT greater_than`
		`  (%doc;, arg+)`
		`>`
		`<!ELEMENT less_than_or_equal`
		`  (%doc;, arg+)`
		`>`
		`<!ELEMENT greater_than_or_equal`
		`  (%doc;, arg+)`
		`>`
		`<!ELEMENT not_equal`
		`  (%doc;, arg+)`
		`>`
		`<!ELEMENT equal`
		`  (%doc;, arg+)`
		`>`
		`<!ELEMENT instance_equal`
		`  (%doc;, arg+)`
		`>`
		`<!ELEMENT instance_not_equal`
		`  (%doc;, arg+)`
		`>`
		`<!ELEMENT add`
		`  (%doc;, arg+)`
		`>`
			Some of these allow more than two args, which is slightly different from the original though functionally the same. Need to check consistency here both with Express and MathML.

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
		`<!ELEMENT subtract (%doc;, arg, arg) > <!ELEMENT multiply (%doc;, arg+) > <!ELEMENT real_divide (%doc;, arg, arg) > <!ELEMENT integer_divide (%doc;, arg, arg) > <!ELEMENT raise_to_power (%doc;, arg, arg) > <!ELEMENT complex_entity_constructor (%doc;, arg+) >`	
		`<!ELEMENT arg (%doc;, (%literal; | expression)) >`	Literals used here as they do not need to be nested and are the most likely case.
0	`ABS = 'abs' !`	`<!ELEMENT abs (%doc;, arg) >`	
1	`ABSTRACT = 'abstract' !`		

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment

2	`ACOS = 'acos' !`	`<!ELEMENT acos`
		`  (%doc;, arg)`
		`>`	

3	`AGGREGATE = 'aggregate' !`		
4	`ALIAS = 'alias' !`		

5	`AND = 'and' !`	`<!ELEMENT and`
		`  (%doc;, arg+)`
		`>`	

6	`ANDOR = 'andor' !`		
7	`ARRAY = 'array' !`		
8	`AS = 'as' !`		

9	`ASIN = 'asin' !`	`<!ELEMENT asin`
		`  (%doc;, arg)`
		`>`	

10	`ATAN = 'atan' !`	`<!ELEMENT atan`
		`  (%doc;, arg)`
		`>`	

11	`BAG = 'bag' !`		
12	`BEGIN = 'begin' !`		
13	`BINARY = 'binary' !`		

14	`BLENGTH = 'blength' !`	`<!ELEMENT bLength`
		`  (%doc;, arg)`
		`>`	

15	`BOOLEAN = 'boolean' !`		
16	`BY = 'by' !`		
17	`CASE = 'case' !`		
18	`CONSTANT = 'constant' !`		
19	`CONST_E = 'const_e' !`	`<!ELEMENT const_e EMPTY>`	
20	`CONTEXT = 'context' !`		

21	`COS = 'cos' !`	`<!ELEMENT cos`
		`  (%doc;, arg)`
		`>`	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
22	DERIVE = 'derive' !		
23	DIV = 'div' !		
24	ELSE = 'else' !		Not needed – see if_stmt
25	END = 'end' !		
26	END_ALIAS = 'end_alias' !		
27	END_CASE = 'end_case' !		
28	END_CONSTANT = 'end_constant' !		
29	END_CONTEXT = 'end_context' !		
30	END_ENTITY = 'end_entity' !		
31	END_FUNCTION = 'end_function' !		
32	END_IF = 'end_if' !		
33	END_LOCAL = 'end_local' !		
34	END_MODEL = 'end_model' !		
35	END_PROCEDURE = 'end_procedure' !		
36	END_REPEAT = 'end_repeat' !		
37	END_RULE = 'end_rule' !		
38	END_SCHEMA = 'end_schema' !		
39	END_TYPE = 'end_type' !		
40	ENTITY = 'entity' !		
41	ENUMERATION = 'enumeration' !		
42	ESCAPE = 'escape' !		
43	EXISTS = 'exists' !	<!ELEMENT exists (%doc;, arg) >	
44	EXP = 'exp' !	<!ELEMENT exp (%doc;, arg) >	
45	FALSE = 'false' !	<!ELEMENT false EMPTY>	
46	FIXED = 'fixed' !	<!ELEMENT fixed EMPTY>	
47	FOR = 'for' !		
48	FORMAT = 'format' !	<!ELEMENT format	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
		(%doc;, arg, string) >	
49	FROM = 'from' !		
50	FUNCTION = 'function' !		
51	GENERIC = 'generic' !		
52	HIBOUND = 'hibound' !	<!ELEMENT hiBound (%doc;, arg) >	
53	HIINDEX = 'hiindex' !	<!ELEMENT hiIndex (%doc;, arg) >	
54	IF = 'if' !		
55	IN = 'in' !	<!ELEMENT in (%doc;, arg, arg) >	
56	INSERT = 'insert' !	<!ELEMENT insert (%doc;, arg, arg, (%numeric_expression_top;)) >	Could have clearer keywords for list and item (first two arg's).
57	INTEGER = 'integer' !		
58	INVERSE = 'inverse' !		
59	LENGTH = 'length' !	<!ELEMENT length (%doc;, arg) >	
60	LIKE = 'like' !	<!ELEMENT like (%doc;, arg, arg) >	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
61	`LIST = 'list' !`		
62	`LOBOUND = 'lobound' !`	`<!ELEMENT loBound (%doc;, arg) >`	
63	`LOCAL = 'local' !`		
64	`LOG = 'log' !`	`<!ELEMENT log (%doc;, arg) >`	
65	`LOG10 = 'log10' !`	`<!ELEMENT log10 (%doc;, arg) >`	
66	`LOG2 = 'log2' !`	`<!ELEMENT log2 (%doc;, arg) >`	
67	`LOGICAL = 'logical' !`		
68	`LOINDEX = 'loindex' !`	`<!ELEMENT loIndex (%doc;, arg) >`	
69	`MOD = 'mod' !`	`<!ELEMENT mod (%doc;, arg, arg) >`	
70	`MODEL = 'model' !`		
71	`NOT = 'not' !`	`<!ELEMENT not (%doc;, arg)>`	
72	`NUMBER = 'number' !`		
73	`NVL = 'nvl' !`	`<!ELEMENT nvl (%doc;, arg, arg) >`	Nvl is about the only acronym which is difficult! Change to null_value?
74	`ODD = 'odd' !`	`<!ELEMENT odd (%doc;, arg)>`	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
75	`OF = 'of' !`		
76	`ONEOF = 'oneof' !`		
77	`OPTIONAL = 'optional' !`	`<!ELEMENT optional EMPTY>`	

78	`OR = 'or' !`	`<!ELEMENT or`
		`  (%doc;, arg+)`
		`>`	

79	`OTHERWISE = 'otherwise' !`	`<!ELEMENT otherwise`
		`  (%doc;, (%stmt;))`
		`>`	

80	`PI = 'pi' !`	`<!ELEMENT pi EMPTY>`	
81	`PROCEDURE = 'procedure' !`		
82	`QUERY = 'query' !`		
83	`REAL = 'real' !`		
84	`REFERENCE = 'reference' !`		

85	`REMOVE = 'remove' !`	`<!ELEMENT remove`
		` (%doc;, arg,`
		`  (%numeric_expression_top;))`
		`>`	

86	`REPEAT = 'repeat' !`		
87	`RETURN = 'return' !`		Not needed – see return_stmt
88	`ROLESOF = 'rolesof' !`	`<!ELEMENT rolesOf (%doc;, arg)>`	
89	`RULE = 'rule' !`		
90	`SCHEMA = 'schema' !`		
91	`SELECT = 'select' !`		
92	`SELF = 'self' !`	`<!ELEMENT self EMPTY>`	
93	`SET = 'set' !`		
94	`SIN = 'sin' !`	`<!ELEMENT sin (%doc;, arg)>`	
95	`SIZEOF = 'sizeof' !`	`<!ELEMENT sizeOf (%doc;, arg)>`	
96	`SKIP = 'skip' !`		
97	`SQRT = 'sqrt' !`	`<!ELEMENT sqrt (%doc;, arg)>`	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
98	STRING = 'string' !		
99	SUBTYPE = 'subtype' !		
100	SUPERTYPE = 'supertype' !		
101	TAN = 'tan' !	<!ELEMENT tan (%doc;, arg)>	
102	THEN = 'then' !		Not needed – see if_stmt
103	TO = 'to' !		
104	TRUE = 'true' !	<!ELEMENT true EMPTY>	
105	TYPE = 'type' !		
106	TYPEOF = 'typeof' !	<!ELEMENT typeOf (%doc;, arg)>	
107	UNIQUE = 'unique' !	<!ELEMENT unique EMPTY>	
108	UNKNOWN = 'unknown' !	<!ELEMENT unknown EMPTY>	
109	UNTIL = 'until' !		
110	USE = 'use' !		
111	USEDIN = 'usedin' !	<!ELEMENT usedIn (%doc;, arg, arg)>	
112	VALUE = 'value' !	<!ELEMENT value (%doc;, arg)>	
113	VALUE_IN = 'value_in' !	<!ELEMENT value_in (%doc;, arg, arg)>	
114	VALUE_UNIQUE = 'value_unique' !	<!ELEMENT value_unique (%doc;, arg)>	
115	VAR = 'var' !		Not needed - see formal_parameter
116	WHERE = 'where' !		
117	WHILE = 'while' !		

118	XOR = 'xor' !	<!ELEMENT xor
		  (%doc;, arg,
		   arg)
		>	

119	Bit = '0' | '1' .		
120	Digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .		

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
121	digits = digit { digit } .		
122	encoded_character = octet octet octet octet .		
123	hex_digit = digit | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' .		
124	letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' .		
125	lparen_not_star = '(' not_star .		
126	not_lparen_star = not_paren_star | ')' .		
127	not_paren_star = letter | digit | not_paren_star_special .		
128	not_paren_star_quote_special = '!' | '"' | '#' | '$' | '%' | '&' | '+' | ',' | '-' | '.' | '/' | ':' | ';' | '<' | '=' | '>' | '?' | '@' | '[' | '\' | ']' | '^' | '_' | '`' | '{' | '|' | '}' | '~' .		
129	not_paren_star_special = not_paren_star_quote_special | '''' .		
130	not_quote = not_paren_star_quote_special | letter | digit | '(' | ')' | '*' .		
131	not_rparen = not_paren_star | '*' | '(' .		
132	not_star = not_paren_star | '(' | ')' .		
133	octet = hex_digit hex_digit .		

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
134	special = not_paren_star_quote_special | '(' | ')' | '*' | '''' !		
135	star_not_rparen = '*' not_rparen .		
136	binary_literal = '%' bit { bit } !	<!ELEMENT binary_literal (#PCDATA) >	
137	encoded_string_literal = '"' encoded_character { encoded_character } '"' .		Not needed – see string_literal
138	integer_literal = digits !	<!ELEMENT integer_literal (#PCDATA) >	
139	real_literal = digits '.' [digits] ['e' [sign] digits] !	<!ELEMENT real_literal (#PCDATA) >	
140	simple_id = letter { letter | digit | '_' } !		
141	simple_string_literal = \q { (\q \q) | not_quote | \s | \o } \q .		Not needed – see string_literal
142	embedded_remark = '(*' { not_lparen_star | lparen_not_star | star_not_rparen | embedded_remark } '*)' .		
143	remark = embedded_remark | tail_remark .		
144	tail_remark = '--' { \a | \s | \o } \n .		
145	attribute_ref = attribute_id !	<!ELEMENT attribute_ref (#PCDATA)	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
		>	
146	constant_ref = constant_id !	<!ELEMENT constant_ref (#PCDATA) >	
147	entity_ref = entity_id !	<!ELEMENT entity_ref (#PCDATA) >	
148	enumeration_ref = enumeration_id !	<!ELEMENT enumeration_ref (#PCDATA) >	
149	function_ref = function_id !	<!ELEMENT function_ref (#PCDATA) >	
150	parameter_ref = parameter_id !	<!ELEMENT parameter_ref (#PCDATA) >	
151	procedure_ref = procedure_id !	<!ELEMENT procedure_ref (#PCDATA) >	
152	schema_ref = schema_id !	<!ELEMENT schema_ref (#PCDATA) >	
153	type_label_ref = type_label_id !	<!ELEMENT type_label_ref (#PCDATA) >	
154	type_ref = type_id !	<!ELEMENT type_ref (#PCDATA) >	
155	variable_ref = variable_id !	<!ELEMENT variable_ref (#PCDATA) >	
156	abstract_supertype_declaration = ABSTRACT SUPERTYPE [	<!ELEMENT abstract_supertype_of (%doc;,	

EXPRESS syntax Ref.	**Express Syntax**	**XML DTD element definitions**	**Comment**
	subtype_constraint] .	(%supertype_expression;)) >	
157	actual_parameter_list = '(' parameter { ',' parameter } ')' .	<!ENTITY % actual_parameter_list “arg*” >	The ‘*’ rather than ‘+’ is due to optional use of actual_parameter_list.
158	add_like_op = '+' | '-' | OR | XOR .	<!ENTITY % add_like_op “add | subtract | or | xor “ >	
159	aggregate_initializer = '[' [element { ',' element }] ']' !	<!ELEMENT aggregate_initializer (%doc;, element_list) >	
160	aggregate_source = simple_expression .	<!ELEMENT aggregate_source (%doc;, (%simple_expression;)) >	
161	aggregate_type = AGGREGATE [':' type_label] OF parameter_type !	<!ELEMENT aggregate_type (%doc;, (%parameter_type;), (%type_label;)?) >	
162	aggregation_types = array_type | bag_type | list_type | set_type !	<!ENTITY % aggregation_types “ array_type | bag_type | list_type | set_type“>	
163	algorithm_head = { declaration } [constant_decl] [local_decl] .	<!ELEMENT algorithm_head (%doc;, declaration_block?,	

EXPRESS syntax Ref.	**Express Syntax**	**XML DTD element definitions**	**Comment**
		constant_block?, local_variable_block?) >	
164	alias_stmt = ALIAS variable_id FOR general_ref { qualifier } ';' stmt { stmt } END_ALIAS ';' !	<!ELEMENT alias_stmt (%doc;, variable_id, (%general_ref;), qualifier?, statement_block) >	
165	array_type = ARRAY bound_spec OF [OPTIONAL] [UNIQUE] base_type !	<!ELEMENT array_type (%doc;, index_spec, base_type, optional?, unique?) > <!ELEMENT index_spec (%doc;, low_index, high_index?) >	We have left in *type for now but it should possibly go. Index used in place of bound*spec as this is different from other aggregates (ref. Request from Phil Spiby).
166	assignment_stmt = general_ref { qualifier } ':=' expression ';' !	<!ELEMENT assignment_stmt (%doc;, (%general_ref;), qualifier?, expression) >	
167	attribute_decl = attribute_id | qualified_attribute .	<!ENTITY % attribute_decl “attribute_id | qualified_attribute” >	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
168	attribute_id = simple_id !	<!ELEMENT attribute_id (#PCDATA) >	
169	attribute_qualifier = '.' attribute_ref .	<!ENTITY % attribute_qualifier “attribute_ref” >	
170	bag_type = BAG [bound_spec] OF base_type !	<!ELEMENT bag_type (%doc;, bound_spec?, base_type) >	We have left in _type for now but it should possibly go.
171	base_type = aggregation_types | simple_types | named_types .	<!ELEMENT base_type (%doc;, (%aggregation_types; | %simple_types; | %named_types;)) >	
172	binary_type = BINARY [width_spec] !	<!ELEMENT binary (%doc;, width_spec?) >	
173	boolean_type = BOOLEAN !	<!ELEMENT boolean EMPTY >	
174	bound_1 = numeric_expression .	<!ELEMENT lower_bound (%doc;, (%numeric_expression_top;)) >	
175	bound_2 = numeric_expression .	<!ELEMENT upper_bound (%doc;, (indeterminate | %numeric_expression_top;)) > <!ELEMENT indeterminate	Added indeterminate here as it is common and otherwise deeply

EXPRESS syntax Ref.	**Express Syntax**	**XML DTD element definitions**	**Comment**
		EMPTY >	nested!
176	bound_spec = '[' bound_1 ':' bound_2 ']' .	<!ELEMENT bound_spec (%doc;, lower_bound, upper_bound) >	Cardinality might be a better element name here. Lower/upper bound used as improved names instead of bound_1/2
177	built_in_constant = CONST_E | PI | SELF | '?' !	<!ENTITY % built_in_constant “const_e | pi | self | unknown” >	
178	built_in_function = ABS | ACOS | ASIN | ATAN | BLENGTH | COS | EXISTS | EXP | FORMAT | HIBOUND | HIINDEX | LENGTH | LOBOUND | LOINDEX | LOG | LOG2 | LOG10 | NVL | ODD | ROLESOF | SIN | SIZEOF | SQRT | TAN | TYPEOF | USEDIN | VALUE | VALUE_IN | VALUE_UNIQUE !	<!ELEMENT built_in_function (abs | acos | asin | atan | bLength | cos | exists | exp | format | hiBound | hiIndex | length | loBound | loIndex | log | log2 | log10 | nvl | odd | rolesOf | sin | sizeOf | sqrt | tan | typeOf | usedIn | value | value_in | value_unique) >	Should we use capitalisation here? More like XML if we do, but we have inconsistency with use of ‘_’ which needs to be resolved.
179	built_in_procedure = INSERT | REMOVE !	<!ENTITY % built_in_procedure “insert | remove” >	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
180	case_action = case_label { ',' case_label } ':' stmt .	<!ELEMENT case_action (%doc;, case_label, (%stmt;)) >	Multiple expressions are contained in case_label, so only one case_label needed here.
181	case_label = expression .	<!ELEMENT case_label (%doc;, expression+) >	
182	case_stmt = CASE selector OF { case_action } [OTHERWISE ':' stmt] END_CASE ';' !	<!ELEMENT case_stmt (%doc;, %selector;, case_action*, otherwise?) >	
183	compound_stmt = BEGIN stmt { stmt } END ';' !	<!ELEMENT compound_stmt (%doc;, statement_block) >	Could dispense with this keyword – kept for clarity.
184	constant_body = constant_id ':' base_type ':=' expression ';' .	<!ELEMENT constant_decl (%doc;, constant_id, base_type, expression) >	Slight name change here as all other _decl are single item declarations.
185	constant_decl = CONSTANT constant_body { constant_body } END_CONSTANT ';' !	<!ELEMENT constant_block (%doc;, constant_decl*) >	
186	constant_factor = built_in_constant | constant_ref .	<!ENTITY % constant_factor “%built_in_constant; | constant_ref”	

EXPRESS syntax Ref.	**Express Syntax**	**XML DTD element definitions**	**Comment**
		`>`	
187	`constant_id = simple_id !`	`<!ELEMENT constant_id (#PCDATA) >`	
188	`constructed_types = enumeration_type | select_type !`	`<!ENTITY % constructed_types “enumeration | select “>`	
189	`declaration = entity_decl | function_decl | procedure_decl | type_decl .`	`<!ENTITY % declaration “entity_decl | function_decl | procedure_decl | type_decl” > <!ELEMENT declaration_block (%doc;, (%declaration;)*) >`	
190	`derived_attr = attribute_decl ':' base_type ':=' expression ';' .`	`<!ELEMENT derived_attr (%doc;, (%attribute_decl;), base_type, expression) >`	
191	`derive_clause = DERIVE derived_attr { derived_attr } !`	`<!ELEMENT derive_clause (%doc;, derived_attr+) >`	
192	`domain_rule = [ label ':' ] expression !`	`<!ELEMENT domain_rule (%doc;, label?, expression) >`	Should this be logical_expression?
193	`element = expression [ ':' repetition ] .`	`<!ELEMENT element_list (%doc;, element_item*) > <!ELEMENT element_item (%doc;, expression, repetition?)`	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
		>	
194	entity_body = { explicit_attr } [derive_clause] [inverse_clause] [unique_clause] [where_clause] .	<!ENTITY % entity_body “explicit_attr_block?, derive_clause?, inverse_clause?, unique_clause?, where_clause?” >	
195	entity_constructor = entity_ref '(' [expression { ',' expression }] ')' !	<!ELEMENT entity_constructor (%doc;, entity_ref, arg*) >	Used arg* here to be consistent with other functions. Really this should specify attribute names etc. to be consistent with late-bound format.
196	entity_decl = entity_head entity_body END_ENTITY ';' !	<!ELEMENT entity_decl (%doc;, %entity_head;, %entity_body;) >	
197	entity_head = ENTITY entity_id [subsuper] ';' .	<!ENTITY % entity_head “entity_id, %subsuper;” >	Subsuper is required as items within it are optional.
198	entity_id = simple_id !	<!ELEMENT entity_id (#PCDATA) >	

EXPRESS syntax Ref.	**Express Syntax**	**XML DTD element definitions**	**Comment**
199	`enumeration_id = simple_id !`	`<!ELEMENT enumeration_id (#PCDATA) >`	
200	`enumeration_reference = [ type_ref '.' ] enumeration_ref !`	`<!ELEMENT enumeration_reference (%doc;, type_ref?, enumeration_ref) >`	
201	`enumeration_type = ENUMERATION OF '(' enumeration_id { ',' enumeration_id } ')' !`	`<!ELEMENT enumeration (%doc;, enumeration_id+) >`	
202	`escape_stmt = ESCAPE ';' !`	`<!ELEMENT escape_stmt EMPTY >`	
203	`explicit_attr = attribute_decl { ',' attribute_decl } ':' [ OPTIONAL ] base_type ';' !`	`<!ELEMENT explicit_attr_block (%doc;, explicit_attr+) >` `<!ELEMENT explicit_attr (%doc;, (%attribute_decl;)+, optional?, base_type) >`	It would be better for the reader to take away the ‘+’ on attribute_decl so only one can be defined here. The shorthand is confusing and not often used, and easy to remove on conversion.
204	`expression = simple_expression [ rel_op_extended simple_expression ] !`	`<!ELEMENT expression (%doc;, (%rel_op_extended; | %simple_expression;))`	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
		>	
205	factor = simple_factor ['**' simple_factor] .	<!ENTITY % factor "%simple_factor; | raise_to_power" >	** is called raise_to_power to avoid any confusion with exp.
206	formal_parameter = parameter_id { ',' parameter_id } ':' parameter_type .	<!ELEMENT formal_parameter_block (%doc;, formal_parameter*) > <!ELEMENT formal_parameter (%doc;, parameter_id+, (%parameter_type;)) > <!ELEMENT procedure_formal_parameter_block (%doc;, (formal_parameter | var_formal_parameter)*) > <!ELEMENT var_formal_parameter (%doc;, parameter_id, (%parameter_type;)) >	Need the extra procedure types because of the possibility of VAR parameters there.
207	function_call = (built_in_function | function_ref) [actual_parameter_list] !	<!ELEMENT function_call (%doc;, function_ref, %actual_parameter_list;) >	Built-in functions are made explicit in the qualifiable_factor which is the only place this is used.
208	function_decl = function_head [algorithm_head] stmt { stmt }	<!ELEMENT function_decl (%doc;, %function_head;,	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
	END_FUNCTION ';' !	algorithm_head?, statement_block) >	
209	function_head = FUNCTION function_id ['(' formal_parameter { ';' formal_parameter } ')'] ':' parameter_type ';' .	<!ENTITY % function_head "function_id, formal_parameter_block?, function_return_type" >	
210	function_id = simple_id !	<!ELEMENT function_id (#PCDATA) >	
211	generalized_types = aggregate_type | general_aggregation_types | generic_type !	<!ENTITY % generalized_types "aggregate_type | %general_aggregation_types; | generic_type" >	
212	general_aggregation_types = general_array_type | general_bag_type | general_list_type | general_set_type !	<!ENTITY % general_aggregation_types "general_array_type | general_bag_type | general_list_type | general_set_type" >	
213	general_array_type = ARRAY [bound_spec] OF [OPTIONAL] [UNIQUE] parameter_type .	<!ELEMENT general_array_type (%doc;, (%parameter_type;), bound_spec?, optional?, unique?) >	
214	general_bag_type = BAG [bound_spec] OF parameter_type .	<!ELEMENT general_bag_type (%doc;, (%parameter_type;), bound_spec?) >	
215	general_list_type = LIST [	<!ELEMENT general_list_type	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
	bound_spec] OF [UNIQUE] parameter_type .	(%doc;, (%parameter_type;), bound_spec?, unique?) >	
216	general_ref = parameter_ref | variable_ref .	<!ENTITY % general_ref “parameter_ref | variable_ref” >	
217	general_set_type = SET [bound_spec] OF parameter_type .	<!ELEMENT general_set_type (%doc;, (%parameter_type;), bound_spec?) >	
218	generic_type = GENERIC [':' type_label] !	<!ELEMENT generic_type (%doc;, (%type_label;)?) >	
219	group_qualifier = '\' entity_ref .	<!ENTITY % group_qualifier “entity_ref” >	
220	if_stmt = IF logical_expression THEN stmt { stmt } [ELSE stmt { stmt }] END_IF ';' !	<!ELEMENT if_stmt (%doc;, logical_expression, statement_block, statement_block?) >	
221	increment = numeric_expression .	<!ELEMENT increment (%doc;, (%numeric_expression_top;)) >	
222	increment_control = variable_id ':=' bound_1 TO bound_2 [BY increment] .	<!ELEMENT increment_control (%doc;, variable_id, lower_bound, upper_bound, increment?) >	

EXPRESS syntax Ref.	**Express Syntax**	**XML DTD element definitions**	**Comment**

223	`index = numeric_expression .`	`<!ENTITY % index`
		`  “%numeric_expression_top;”`
		`>`	

224	`index_1 = index .`	`<!ELEMENT low_index`
		` (%doc;,`
		`  (%index;))`
		`>`
			Name change here as it becomes a keyword so meaning is by keyword rather than position.

225	`index_2 = index .`	`<!ELEMENT high_index`
		` (%doc;,`
		`  (%index;))`
		`>`	

226	`index_qualifier = '[' index_1 [ ':' index_2 ] ']' .`	`<!ELEMENT index_qualifier`
		` (%doc;, low_index,`
		`  high_index?)`
		`>`	

227	`integer_type = INTEGER !`	`<!ELEMENT integer`
		` EMPTY`
		`>`	

228	`interface_specification = reference_clause | use_clause .`	`<!ELEMENT interface_specification_block`
		`  (%doc;, (reference_from |`
		`   use_from)+)`
		`>`	

229	`interval = '{' interval_low interval_op interval_item interval_op interval_high '}' !`	`<!ELEMENT interval`
		`  (%doc;, (interval_low_inclusive |`
		`    interval_low_exclusive),`
		`   interval_item,`
		`   (interval_high_inclusive |`
		`    interval_high_exclusive))`
		`>`	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
230	interval_high = simple_expression .	`<!ELEMENT interval_high_inclusive (%doc;, (%simple_expression;)) >` `<!ELEMENT interval_high_exclusive (%doc;, (%simple_expression;)) >`	These should probably be numeric_expressions?
231	interval_item = simple_expression .	`<!ELEMENT interval_item (%doc;, (%simple_expression;)) >`	
232	interval_low = simple_expression .	`<!ELEMENT interval_low_inclusive (%doc;, (%simple_expression;)) >` `<!ELEMENT interval_low_exclusive (%doc;, (%simple_expression;)) >`	
233	interval_op = '<' | '<=' .		Not needed: adopted the in/exclusive approach as per XML schema.
234	inverse_attr = attribute_decl ':' [(SET | BAG) [bound_spec] OF] entity_ref FOR attribute_ref ';' .	`<!ELEMENT inverse_attr (%doc;, (%attribute_decl;), entity_ref, attribute_ref, (inverse_set | inverse_bag)?) >`	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment

		`<!ELEMENT inverse_set`
		` (%doc;, bound_spec?)`
		`>`
		`<!ELEMENT inverse_bag`
		` (%doc;, bound_spec?)`
		`>`	

235	`inverse_clause = INVERSE inverse_attr { inverse_attr } !`	`<!ELEMENT inverse_clause`
		`(%doc;, inverse_attr+)`
		`>`	

236	`label = simple_id !`	`<!ELEMENT label`
		` (#PCDATA)`
		`>`	

237	`list_type = LIST [ bound_spec ] OF [ UNIQUE ] base_type !`	`<!ELEMENT list_type`
		`  (%doc;, bound_spec?,`
		`   base_type,`
		`   unique?)`
		`>`
			Should base_type be first always?

238	`literal = binary_literal | integer_literal | logical_literal | real_literal | string_literal !`	`<!ENTITY % literal`
		`  “binary_literal |`
		`  integer_literal |`
		`  logical_literal |`
		`  real_literal |`
		`  string_literal”`
		`>`	

239	`local_decl = LOCAL local_variable { local_variable } END_LOCAL ';' !`	`<!ELEMENT local_variable_block`
		` (%doc;, local_variable*)`
		`>`
			More appropriate name used to be consistent with other uses of decl and block.

240	`local_variable = variable_id { ',' variable_id } ':' parameter_type [`	`<!ELEMENT local_variable`
		` (%doc;, variable_id+,`
			Should this be

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
	':=' expression] ';' .	(%parameter_type;), expression) >	local_varibal e_decl?
241	logical_expression = expression .	<!ELEMENT logical_expression (%doc;, expression) >	
242	logical_literal = FALSE | TRUE | UNKNOWN !	<!ELEMENT logical_literal (%doc;, (false | true | unknown)) >	
243	logical_type = LOGICAL !	<!ELEMENT logical EMPTY >	
244	multiplication_like_op = '*' | '/' | DIV | MOD | AND | '||' .	<!ENTITY % multiplication_like_op “multiply | real_divide | integer_divide | mod | and | complex_entity_constructor“ >	This allows some non-numeric operators within numeric_expre ssion, but that is the way the current Express syntax is structured! We could improve this, but it may not be too easy.

EXPRESS syntax Ref.	**Express Syntax**	**XML DTD element definitions**	**Comment**
245	named_types = entity_ref | type_ref !	<!ENTITY % named_types "entity_ref | type_ref " >	
246	named_type_or_rename = named_types [AS (entity_id | type_id)] .		Not needed - see use_from
247	null_stmt = ';' !	<!ELEMENT null_stmt EMPTY >	
248	number_type = NUMBER !	<!ELEMENT number EMPTY >	
249	numeric_expression = simple_expression !	<!ENTITY % numeric_expression_top "integer_literal | numeric_expression" > <!ELEMENT numeric_expression (%doc;, (%simple_expression;)) >	We have brought out integer_liter al here as it is the most common and is otherwise nested several levels down!
250	one_of = ONEOF '(' supertype_expression { ',' supertype_expression } ')' .	<!ELEMENT supertype_one_of (%doc;, (%supertype_expression;)+) >	
251	parameter = expression .		Used arg instead.
252	parameter_id = simple_id !	<!ELEMENT parameter_id (#PCDATA) >	
253	parameter_type = generalized_types | named_types | simple_types .	<!ENTITY % parameter_type "%generalized_types; | %named_types; |	Function return type is a specific

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
		%simple_types;" > <!ELEMENT function_return_type (%parameter_type;) >	use here which is wrapped for clarity.
254	population = entity_ref .	<!ELEMENT population (%doc;, entity_ref) >	
255	precision_spec = numeric_expression .	<!ELEMENT precision_spec (%doc;, (%numeric_expression_top;)) >	
256	primary = literal | (qualifiable_factor { qualifier }) .	<!ENTITY % primary "%literal; | %qualifiable_factor;" >	
257	procedure_call_stmt = (built_in_procedure | procedure_ref) [actual_parameter_list] ';' !	<!ELEMENT procedure_call_stmt (%doc;, ((%built_in_procedure;) | (procedure_ref, %actual_parameter_list;))) >	Parameter_list only needed when there is a procedure_ref
258	procedure_decl = procedure_head [algorithm_head] { stmt } END_PROCEDURE ';' !	<!ELEMENT procedure_decl (%doc;, %procedure_head;, algorithm_head?, statement_block?) >	
259	procedure_head = PROCEDURE procedure_id ['(' [VAR] formal_parameter { ';' [VAR] formal_parameter } ')'] ';' .	<!ENTITY % procedure_head "procedure_id, procedure_formal_parameter_block?" >	
260	procedure_id = simple_id !	<!ELEMENT procedure_id (#PCDATA)	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
		>	
261	qualifiable_factor = attribute_ref | constant_factor | function_call | general_ref | population !	<!ENTITY % qualifiable_factor “%constant_factor; | built_in_function | function_call | qualified_attribute_ref | attribute_ref | partial_entity_ref | population | parameter_ref | variable_ref | index_qualified_item” > <!ELEMENT qualified_attribute_ref (%doc;, attribute_ref, (index_qualifier | attribute_ref | qualified_attribute_ref | variable_ref | parameter_ref | partial_entity_ref))> <!ELEMENT partial_entity_ref (%doc;, entity_ref)> <!ELEMENT index_qualified_item (%doc;, (self | function_call | built_in_function | parameter_ref | constant_ref | variable_ref population),	This has been changed to make it more explicit to avoid confusion about what is allowed where: it does not translate as is to XML at all well, because qualifiable_factor has to be an element and it is then very unintuitive to look inside this for the items within it. Qualification works backwards relative to the original Express, e.g. a.b.c becomes qualified ref

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
		index_qualifier)>	to c, qualified by b which is qualified by a. Partial_entity_ref is used here because entity_ref in this context is pointing to another entity which is related to the current one by sub/super type, directly or indirectly. Not clear if this caters correctly for referencing multi-dimensioned arrays etc. Needs to be checked!
262	qualified_attribute = SELF group_qualifier attribute_qualifier !	<!ELEMENT qualified_attribute (%doc;, %group_qualifier;, %attribute_qualifier;)	Would this be clearer called a

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
		`>`	‘supertype_at tribute’?

263	`qualifier = attribute_qualifier | group_qualifier | index_qualifier !`	`<!ENTITY % qualifier_content`
		`  “%attribute_qualifier; |`
		`   %group_qualifier; |`
		`   index_qualifier”`
		`>`
		`<!ELEMENT qualifier`
		` (%doc;, (%qualifier_content;),`
		`  qualifier?)`
		`>`
			All uses of qualifier are in {}, so making it recursive is natural. However, this should be cleaned up to be more semantically rich/correct.

264	`query_expression = QUERY '(' variable_id '<*' aggregate_source '|' logical_expression ')' !`	`<!ELEMENT query`
		`  (%doc;, variable_id,`
		`   (aggregate_source,`
		`    logical_expression))`
		`>`	

265	`real_type = REAL [ '(' precision_spec ')' ] !`	`<!ELEMENT real`
		` (%doc;, precision_spec?)`
		`>`	

266	`referenced_attribute = attribute_ref | qualified_attribute .`	`<!ENTITY % referenced_attribute`
		` “attribute_ref |`
		`  qualified_attribute”`
		`>`	

267	`reference_clause = REFERENCE FROM schema_ref [ '(' resource_or_rename { ',' resource_or_rename } ')' ] ';' !`	`<!ELEMENT reference_from`
		`  (%doc;, schema_ref,`
		`   (import_all |`
		`    (constant_import |`
		`     entity_import |`
		`     function_import |`
		`     procedure_import |`
			The defaulted values (definition ids) have been made explicit here which will be

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment

		`type_import)+))`
		`>`
		`<!ELEMENT constant_import`
		`  (%doc;, constant_id,`
		`   constant_ref)`
		`>`
		`<!ELEMENT entity_import`
		`  (%doc;, entity_id,`
		`  entity_ref)`
		`>`
		`<!ELEMENT function_import`
		`  (%doc;, function_id,`
		`  function_ref)`
		`>`
		`<!ELEMENT procedure_import`
		`  (%doc;, procedure_id,`
		` procedure_ref)`
		`>`
		`<!ELEMENT type_import`
		`  (%doc;, type_id,`
		`  type_ref)`
		`>`
			easier for readers and is particularly important for easy set-up of hyperlinks to definitions.

268	`rel_op = '<' | '>' | '<=' | '>=' | '<>' | '=' | ':<>:' | ':=:' .`	`<!ENTITY % rel_op`
		`  “less_than |`
		`  greater_than |`
		`  less_than_or_equal |`
		`  greater_than_or_equal |`
		`  not_equal |`
		`  equal |`
		`  instance_not_equal |`
		`  instance_equal”`
		`>`	

269	`rel_op_extended = rel_op | IN |`	`<!ENTITY % rel_op_extended`	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
	LIKE .	“%rel_op; | in | like” >	
270	rename_id = constant_id | entity_id | function_id | procedure_id | type_id .		Not needed, xx_import allows rename which is more specific.
271	repeat_control = [increment_control] [while_control] [until_control] .	<!ELEMENT repeat_control (%doc;, increment_control, while?, until?) >	
272	repeat_stmt = REPEAT repeat_control ';' stmt { stmt } END_REPEAT ';' !	<!ELEMENT repeat_stmt (%doc;, repeat_control, statement_block) >	
273	repetition = numeric_expression .	<!ELEMENT repetition (%doc;, (%numeric_expression_top;)) >	Clearer to have this as an element rather than entity.
274	resource_or_rename = resource_ref [AS rename_id] .		Not needed – see reference_clause.
275	resource_ref = constant_ref | entity_ref | function_ref | procedure_ref | type_ref .		Not needed – see reference_clause.
276	return_stmt = RETURN ['(' expression ')'] ';' !	<!ELEMENT return_stmt (%doc;, expression?)	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
		>	
277	rule_decl = rule_head [algorithm_head] { stmt } where_clause END_RULE ';' !	<!ELEMENT rule_decl (%doc;, %rule_head;, where_clause) >	
278	rule_head = RULE rule_id FOR '(' entity_ref { ',' entity_ref } ')' ';' .	<!ENTITY % rule_head “rule_id, applies_to_entities” > <!ELEMENT applies_to_entities (%doc;, entity_ref+) >	Entity_ref needs to be nested here as it is repeated, so a meaningful element name is used.
279	rule_id = simple_id !	<!ELEMENT rule_id (#PCDATA) >	

280	schema_body = { interface_specification } [constant_decl] { declaration | rule_decl } .	<!ENTITY % schema_body
		  “interface_specification_block?,
		   constant_block?,
		   (%declaration; |
		    rule_decl)*”
		>	

281	schema_decl = SCHEMA schema_id ';' schema_body END_SCHEMA ';' !	<!ELEMENT schema_decl
		  (%doc;, schema_id,
		   %schema_body;)
		>	

282	schema_id = simple_id !	<!ELEMENT schema_id (#PCDATA) >	
283	selector = expression .	<!ENTITY % selector “expression” >	
284	select_type = SELECT '('	<!ELEMENT select	

EXPRESS syntax Ref.	**Express Syntax**	**XML DTD element definitions**	**Comment**
	named_types { ',' named_types } ')' !	(%doc;, (%named_types;)+) >	
285	set_type = SET [bound_spec] OF base_type !	<!ELEMENT set_type (%doc;, bound_spec?, base_type) >	
286	sign = '+' | '-' .		Part of PCDATA in numeric values.
287	simple_expression = term { add_like_op term } !	<!ENTITY % simple_expression "%term; | %add_like_op;" >	Pre-fix operators.
288	simple_factor = aggregate_initializer | entity_constructor | enumeration_reference | interval | query_expression | ([unary_op] ('(' expression')' | primary)) .	<!ENTITY % simple_factor "aggregate_initializer | entity_constructor | enumeration_reference | interval | query | not | %primary;" >	We cannot negate expressions at the moment. May need to adjust although subtract could be used.
289	simple_types = binary_type | boolean_type | integer_type | logical_type | number_type | real_type | string_type !	<!ENTITY % simple_types "binary | boolean | integer | logical | number | real | string " >	
290	skip_stmt = SKIP ';' !	<!ELEMENT skip_stmt EMPTY >	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
291	stmt = alias_stmt | assignment_stmt | case_stmt | compound_stmt | escape_stmt | if_stmt | null_stmt | procedure_call_stmt | repeat_stmt | return_stmt | skip_stmt !	<!ENTITY % stmt “alias_stmt | assignment_stmt | case_stmt | compound_stmt | escape_stmt | if_stmt | null_stmt | procedure_call_stmt | repeat_stmt | return_stmt | skip_stmt” > <!ELEMENT statement_block (%doc;, (%stmt;)+) >	
292	string_literal = simple_string_literal | encoded_string_literal !	<!ELEMENT string_literal (#PCDATA) >	
293	string_type = STRING [width_spec] !	<!ELEMENT string (%doc;, width_spec?) >	
294	subsuper = [supertype_constraint] [subtype_declaration] !	<!ENTITY % subsuper “(%supertype_constraint;)?, subtype_of?” >	
295	subtype_constraint = OF '(' supertype_expression ')' .		See supertype_expression
296	subtype_declaration = SUBTYPE OF '(' entity_ref { ',' entity_ref } ')' !	<!ELEMENT subtype_of (%doc;, entity_ref+) >	
297	supertype_constraint =	<!ENTITY % supertype_constraint	

EXPRESS syntax Ref.	**Express Syntax**	**XML DTD element definitions**	**Comment**
	abstract_supertype_declaration | supertype_rule .	“abstract_supertype_of | supertype_of “ >	
298	supertype_expression = supertype_factor { ANDOR supertype_factor } .	<!ENTITY % supertype_expression “entity_ref | supertype_one_of | supertype_and_or | supertype_and “ >	
299	supertype_factor = supertype_term { AND supertype_term } .		See supertype_expression
300	supertype_rule = SUPERTYPE subtype_constraint .	<!ELEMENT supertype_of (%doc;, (%supertype_expression;)) > <!ELEMENT supertype_and_or (%doc;, (%supertype_expression;)+) > <!ELEMENT supertype_and (%doc;, (%supertype_expression;)+) >	
301	supertype_term = entity_ref | one_of | '(' supertype_expression ')' .		See supertype_expression
302	syntax = schema_decl { schema_decl } .		In express_driven_data.
303	term = factor { multiplication_like_op factor } .	<!ENTITY % term “%factor; | %multiplication_like_op;” >	
304	type_decl = TYPE type_id '='	<!ELEMENT type_decl	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment
	underlying_type ';' [where_clause] END_TYPE ';' !	(%doc;, type_id, underlying_type, where_clause?) >	
305	type_id = simple_id !	<!ELEMENT type_id (#PCDATA) >	
306	type_label = type_label_id | type_label_ref .	<!ENTITY % type_label “type_label_id | type_label_ref” >	
307	type_label_id = simple_id !	<!ELEMENT type_label_id (#PCDATA) >	
308	unary_op = '+' | '-' | NOT .		The + and – are part of a number within #PCDATA, so the NOT is all that is left and has been moved up to simple_factor .
309	underlying_type = constructed_types | aggregation_types | simple_types | type_ref .	<!ELEMENT underlying_type (%doc;, (%constructed_types; | %aggregation_types; | %simple_types; | type_ref)) >	
310	unique_clause = UNIQUE unique_rule	<!ELEMENT unique_clause	

EXPRESS syntax Ref.	Express Syntax	XML DTD element definitions	Comment

	`';' { unique_rule ';' } !`	`(%doc;, unique_rule+)`
		`>`	

311	`unique_rule = [ label ':' ] referenced_attribute { ',' referenced_attribute } .`	`<!ELEMENT unique_rule`
		`(%doc;, label?,`
		`(%referenced_attribute;)+)`
		`>`	

312	`until_control = UNTIL logical_expression !`	`<!ELEMENT until`
		`(%doc;, logical_expression)`
		`>`	

313	`use_clause = USE FROM schema_ref [ '(' named_type_or_rename { ',' named_type_or_rename } ')' ] ';' !`	`<!ELEMENT use_from`
		`(%doc;, schema_ref,`
		`(import_all |`
		`(entity_import |`
		`type_import)+))`
		`>`
			The default that all items are imported has been made explicit here. This is much clearer.

314	`variable_id = simple_id !`	`<!ELEMENT variable_id`
		`(#PCDATA)`
		`>`	

315	`where_clause = WHERE domain_rule ';' { domain_rule ';' } !`	`<!ELEMENT where_clause`
		`(%doc;, domain_rule+)`
		`>`	

316	`while_control = WHILE logical_expression !`	`<!ELEMENT while`
		`(%doc;, logical_expression)`
		`>`	

317	`width = numeric_expression .`	`<!ENTITY % width`
		`“%numeric_expression_top;”`
		`>`	

318	`width_spec = '(' width ')' [ FIXED ] .`	`<!ELEMENT width_spec`
		`(%doc;, (%width;),`
		`fixed?)`
		`>`	

Annex D
(normative)

Information object registration

D.1 Document identification

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

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

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

D.2 Information identification

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

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

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

Assign other information identifiers as appropriate

Annex E
(informative)

Computer interpretable listings

This annex references a listing of the DTD's specified in this part of ISO 10303. These listings are available in computer-interpretable form and can be found at the following URL:

http://www.mel.nist.gov/step/parts/part28e1/cd/

If there is difficulty accessing these sites contact ISO Central Secretariat or contact the ISO TC 184/SC4 Secretariat directly at: sc4sec@cme.nist.gov.

NOTE – The information provided in computer-interpretable form at the above URLs is informative. The information that is contained in the body of this part of ISO 10303 is normative.

Annex F
(informative)

Examples

F.1 Example schema

The following example schema

```
SCHEMA mr_jones_garden;

(*
---------------------------------------------------------
*)

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

(*
---------------------------------------------------------
*)

TYPE
  temperature  = REAL;
END_TYPE;

(*
---------------------------------------------------------
*)

ENTITY real_value_range
    ABSTRACT SUPERTYPE OF (ONEOF (temperature_range,
                                  ph_range));
  minimum_value               : REAL;
  maximum_value               : REAL;
WHERE
  the_values_must_be_sensible :
    maximum_value >= minimum_value;
END_ENTITY;

(*
---------------------------------------------------------
*)

ENTITY temperature_range
    SUBTYPE OF (real_value_range);
  SELF\real_value_range.minimum_value : temperature;
  SELF\real_value_range.maximum_value : temperature;
END_ENTITY;

(*
---------------------------------------------------------
*)

TYPE
  plant_name  = STRING;
END_TYPE;

(*
---------------------------------------------------------
*)
```

```
TYPE
  ph  = REAL;
WHERE
  the_ph_is_between_0_and_14 :
    {0 <= SELF <= 14};
END_TYPE;

(*
---------------------------------------------------------
*)

ENTITY ph_range
     SUBTYPE OF (real_value_range);
  SELF\real_value_range.minimum_value : ph;
  SELF\real_value_range.maximum_value : ph;
END_ENTITY;

(*
---------------------------------------------------------
*)

ENTITY garden;
  has_greenhouse               : greenhouse;
  climatic_temperature_range : temperature_range;
  has_beds                    : SET [5 : 5] OF bed;
END_ENTITY;

(*
---------------------------------------------------------
*)

RULE only_one_garden FOR (garden);
WHERE
  mr_jones_has_one_garden :
    SIZEOF (garden) = 1;
END_RULE;

(*
---------------------------------------------------------
*)

ENTITY greenhouse;
  enforced_temperature_range  : temperature_range;
  holds_plants                : SET [1 : ?] OF greenhouse_plant;
INVERSE
  the_garden                  : garden FOR has_greenhouse;
END_ENTITY;

(*
---------------------------------------------------------
*)

ENTITY bed;
  acidity                     : ph;
  holds_plants                : SET [1 : ?] OF outdoors_plant;
INVERSE
  the_garden                 : garden FOR has_beds;
UNIQUE
  every_bed_has_a_different_acidity :
    acidity;
END_ENTITY;

(*
--------------------------------------------------------
*)

ENTITY plant
    ABSTRACT SUPERTYPE OF (ONEOF (greenhouse_plant,
                                  outdoors_plant));
  colour                      : flower_colour;
  latin_name                  : plant_name;
  english_names               : OPTIONAL SET [1 : ?] OF plant_name;
  survival_temperature_range  : temperature_range;
UNIQUE
  the_latin_name_of_a_plant_species_is_unique :
    latin_name;
END_ENTITY;

(*
--------------------------------------------------------
*)

ENTITY greenhouse_plant
    SUBTYPE OF (plant);
INVERSE
  the_greenhouse              : greenhouse FOR holds_plants;
WHERE
(*
A greenhouse plant can survive in the greenhouse temperature
*)

r1:
 is_sub_range (the_greenhouse.enforced_temperature_range,
              SELF\plant.survival_temperature_range);
(*
A greenhouse plan cannot survive in the garden temperature
*)

r2:
NOT is_sub_range(the_greenhouse.the_garden.climatic_temperature_range,
                  SELF\plant.survival_temperature_range);
END_ENTITY;

(*
--------------------------------------------------------
*)

ENTITY outdoors_plant
    SUBTYPE OF (plant);
  survival_ph_range           : ph_range;
INVERSE
  the_beds     : SET [1 : ?] OF bed FOR holds_plants;
WHERE
(*
The ph_range of the outdoors plant must include the ph value of the
bed
*)

r1:
    QUERY (b <* the_beds |
    value_is_within_range (b.acidity, survival_ph_range))
    = the_beds;
(*
An outdoors plant can survive in the garden temperature
*)

 r2:
 is_sub_range (the_beds [1].the_garden.climatic_temperature_range,
                  SELF\plant.survival_temperature_range);
END_ENTITY;

(*
---------------------------------------------------------
*)

FUNCTION value_is_within_range (v : REAL;
   r : real_value_range) : BOOLEAN;
  RETURN ( (v >= r.minimum_value) AND (v <= r.maximum_value));
END_FUNCTION;

(*
---------------------------------------------------------
*)

FUNCTION is_sub_range (r1,
    r2 : real_value_range) : BOOLEAN;
  RETURN (value_is_within_range (r1.minimum_value, r2) AND
          value_is_within_range (r1.maximum_value, r2));
END_FUNCTION;

END_SCHEMA
```

This schema is encoded in XML as follows:

```
<?xml version="1.0"?>
<!DOCTYPE schema_decl SYSTEM
"../master%20DTD%20tables/version%204/temp/express-v4-expand.dtd">
<schema_decl>
  <documentation>This is an example model created by Alan Williams
(The University of
    Manchester) to illustrate different aspects of EXPRESS.
    This XML version was created by Robin La Fontaine on 16 October 1999.
    </documentation>
  <schema_id>mr_jones_garden</schema_id>
  <type_decl>
    <type_id>flower_colour</type_id>
    <underlying_type>
      <enumeration>
        <enumeration_id>red</enumeration_id>
        <enumeration_id>yellow</enumeration_id>
        <enumeration_id>white</enumeration_id>
      </enumeration>
    </underlying_type>
  </type_decl>
  <type_decl>
    <type_id>temperature</type_id>
    <underlying_type>
      <real>
      </real>
    </underlying_type>
  </type_decl>
  <entity_decl>
    <entity_id>real_value_range</entity_id>
    <abstract_supertype_of>
      <supertype_one_of>
        <entity_ref>temperature_range</entity_ref>
        <entity_ref>ph_range</entity_ref>
      </supertype_one_of>
    </abstract_supertype_of>
    <explicit_attr_block>
      <explicit_attr>
        <attribute_id>minimum_value</attribute_id>
        <base_type>
          <real>
          </real>
        </base_type>
      </explicit_attr>
      <explicit_attr>
        <attribute_id>maximum_value</attribute_id>
        <base_type>
          <real>
          </real>
        </base_type>
      </explicit_attr>
    </explicit_attr_block>
    <where_clause>
      <domain_rule>
        <label>the_values_must_be_sensible</label>
        <expression>
```

```
          <greater_than_or_equal>
            <arg>
              <expression>
                <attribute_ref>maximum_value</attribute_ref>
              </expression>
            </arg>
            <arg>
              <expression>
                <attribute_ref>minimum_value</attribute_ref>
              </expression>
            </arg>
          </greater_than_or_equal>
        </expression>
      </domain_rule>
    </where_clause>
  </entity_decl>
  <entity_decl>
    <entity_id>temperature_range</entity_id>
    <subtype_of>
      <entity_ref>real_value_range</entity_ref>
    </subtype_of>
    <explicit_attr_block>
      <explicit_attr>
        <qualified_attribute>
                <entity_ref>real_value_range</entity_ref>
                <attribute_ref>minimum_value</attribute_ref>
        </qualified_attribute>
        <base_type>
          <type_ref>temperature</type_ref>
        </base_type>
      </explicit_attr>
      <explicit_attr>
        <qualified_attribute>
                <entity_ref>real_value_range</entity_ref>
                <attribute_ref>maximum_value</attribute_ref>
        </qualified_attribute>
        <base_type>
          <type_ref>temperature</type_ref>
        </base_type>
      </explicit_attr>
    </explicit_attr_block>
  </entity_decl>
  <type_decl>
    <type_id>ph</type_id>
    <underlying_type>
      <real>
      </real>
    </underlying_type>
    <where_clause>
      <domain_rule>
        <label>the_ph_is_between_0_and_14</label>
        <expression>
          <less_than_or_equal>
            <arg>
                <integer_literal>0</integer_literal>
            </arg>
            <arg>
              <expression><self/>
              </expression>
            </arg>
            <arg>
              <integer_literal>14</integer_literal>
            </arg>
          </less_than_or_equal>
        </expression>
      </domain_rule>
    </where_clause>
  </type_decl>
  <entity_decl>
    <entity_id>ph_range</entity_id>
    <subtype_of>
      <entity_ref>real_value_range</entity_ref>
    </subtype_of>
    <explicit_attr_block>
      <explicit_attr>
        <qualified_attribute>
                <entity_ref>real_value_range</entity_ref>
                <attribute_ref>minimum_value</attribute_ref>
        </qualified_attribute>
        <base_type>
          <type_ref>ph</type_ref>
        </base_type>
      </explicit_attr>
      <explicit_attr>
        <qualified_attribute>
                <entity_ref>real_value_range</entity_ref>
                <attribute_ref>maximum_value</attribute_ref>
        </qualified_attribute>
        <base_type>
          <type_ref>ph</type_ref>
        </base_type>
      </explicit_attr>
    </explicit_attr_block>
  </entity_decl>
  <entity_decl>
    <entity_id>garden</entity_id>
    <explicit_attr_block>
      <explicit_attr>
        <attribute_id>has_greenhouse</attribute_id>
        <base_type>
          <entity_ref>greenhouse</entity_ref>
        </base_type>
      </explicit_attr>
      <explicit_attr>
                <attribute_id>climatic_temperature_range</attribute_id>
        <base_type>
                <entity_ref>temperature_range</entity_ref>
        </base_type>
      </explicit_attr>
      <explicit_attr>
```

```
        <attribute_id>has_beds</attribute_id>
        <base_type>
          <set_type>
            <bound_spec>
              <lower_bound>
                <integer_literal>5</integer_literal>
              </lower_bound>
              <upper_bound>
                <integer_literal>5</integer_literal>
              </upper_bound>
            </bound_spec>
            <base_type>
                <entity_ref>bed</entity_ref>
            </base_type>
          </set_type>
        </base_type>
      </explicit_attr>
    </explicit_attr_block>
  </entity_decl>
  <rule_decl>
    <rule_id>only_one_garden</rule_id>
    <applies_to_entities>
      <entity_ref>garden</entity_ref>
    </applies_to_entities>
    <where_clause>
      <domain_rule>
        <label>mr_jones_has_one_garden</label>
        <expression>
          <equal>
            <arg>
              <expression>
                <built_in_function>
                <sizeOf>
                <arg>
                  <expression>
                    <population>
                      <entity_ref>garden</entity_ref>
                    </population>
                  </expression>
                </arg>
                </sizeOf>
                </built_in_function>
              </expression>
            </arg>
            <arg>
                <integer_literal>1</integer_literal>
            </arg>
          </equal>
        </expression>
      </domain_rule>
    </where_clause>
  </rule_decl>
  <entity_decl>
    <entity_id>greenhouse</entity_id>
    <explicit_attr_block>
      <explicit_attr>
                <attribute_id>enforced_temperature_range</attribute_id>
        <base_type>
                <entity_ref>temperature_range</entity_ref>
        </base_type>
      </explicit_attr>
      <explicit_attr>
        <attribute_id>holds_plants</attribute_id>
        <base_type>
          <set_type>
            <bound_spec>
              <lower_bound>
                <integer_literal>1</integer_literal>
              </lower_bound>
                <upper_bound><indeterminate/>
              </upper_bound>
            </bound_spec>
            <base_type>
                <entity_ref>greenhouse_plant</entity_ref>
            </base_type>
          </set_type>
        </base_type>
      </explicit_attr>
    </explicit_attr_block>
    <inverse_clause>
      <inverse_attr>
        <attribute_id>the_garden</attribute_id>
        <entity_ref>garden</entity_ref>
        <attribute_ref>has_greenhouse</attribute_ref>
      </inverse_attr>
    </inverse_clause>
  </entity_decl>
  <entity_decl>
    <entity_id>bed</entity_id>
    <explicit_attr_block>
      <explicit_attr>
        <attribute_id>acidity</attribute_id>
        <base_type>
          <type_ref>ph</type_ref>
        </base_type>
      </explicit_attr>
      <explicit_attr>
        <attribute_id>holds_plants</attribute_id>
        <base_type>
          <set_type>
            <bound_spec>
              <lower_bound>
                <integer_literal>1</integer_literal>
              </lower_bound>
                <upper_bound><indeterminate/>
              </upper_bound>
            </bound_spec>
            <base_type>
                <entity_ref>outdoors_plant</entity_ref>
            </base_type>
          </set_type>
        </base_type>
      </explicit_attr>
    </explicit_attr_block>
    <inverse_clause>
      <inverse_attr>
        <attribute_id>the_garden</attribute_id>
        <entity_ref>garden</entity_ref>
        <attribute_ref>has_beds</attribute_ref>
      </inverse_attr>
    </inverse_clause>
    <unique_clause>
      <unique_rule>
                <label>every_bed_has_a_different_acidity</label>
        <attribute_ref>acidity</attribute_ref>
      </unique_rule>
    </unique_clause>
  </entity_decl>
  <entity_decl>
    <entity_id>plant</entity_id>
    <abstract_supertype_of>
      <supertype_one_of>
        <entity_ref>greenhouse_palnt</entity_ref>
        <entity_ref>outdoors_plant</entity_ref>
      </supertype_one_of>
    </abstract_supertype_of>
    <explicit_attr_block>
      <explicit_attr>
        <attribute_id>colour</attribute_id>
        <base_type>
          <type_ref>flower_colour</type_ref>
        </base_type>
      </explicit_attr>
      <explicit_attr>
        <attribute_id>latin_name</attribute_id>
        <base_type>
          <type_ref>plant_name</type_ref>
        </base_type>
      </explicit_attr>
      <explicit_attr>
        <attribute_id>english_names</attribute_id>
        <base_type>
          <set_type>
            <bound_spec>
              <lower_bound>
                <integer_literal>1</integer_literal>
              </lower_bound>
                <upper_bound><indeterminate/>
              </upper_bound>
            </bound_spec>
            <base_type>
                <type_ref>plant_name</type_ref>
            </base_type>
          </set_type>
        </base_type>
      </explicit_attr>
    </explicit_attr_block>
    <unique_clause>
      <unique_rule>
                <label>the_latin_name_of_a_plant_species_is_unique</label>
        <attribute_ref>latin_name</attribute_ref>
      </unique_rule>
    </unique_clause>
  </entity_decl>
  <entity_decl>
    <entity_id>greenhouse_plant</entity_id>
    <subtype_of>
      <entity_ref>plant</entity_ref>
    </subtype_of>
    <inverse_clause>
      <inverse_attr>
        <attribute_id>the_greenhouse</attribute_id>
        <entity_ref>greenhouse</entity_ref>
        <attribute_ref>holds_plants</attribute_ref>
      </inverse_attr>
    </inverse_clause>
    <where_clause>
      <domain_rule>
        <documentation>A greenhouse plant can survive in the greenhouse
          temperature</documentation>
        <label>r1</label>
        <expression>
          <function_call>
                <function_ref>is_sub_range</function_ref>
            <arg>
              <expression>
                <qualified_attribute_ref>
                <documentation>RLF note: these are necessarily in reverse
to

                the original!</documentation>
                <attribute_ref>enforced_temperature_range</attribute_ref>
                <attribute_ref>the_greenhouse</attribute_ref>
                </qualified_attribute_ref>
              </expression>
            </arg>
            <arg>
              <expression>
                <qualified_attribute_ref>
                <attribute_ref>survival_temperature_range</attribute_ref>
                <partial_entity_ref>
                <entity_ref>plant</entity_ref>
                </partial_entity_ref>
                </qualified_attribute_ref>
              </expression>
            </arg>
          </function_call>
        </expression>
      </domain_rule>
      <domain_rule>
        <documentation>A greenouse plant cannot survive in the garden
```

```
          temperature</documentation>
        <label>r2</label>
        <expression>
          <not>
            <arg>
              <expression>
                <function_call>
                <function_ref>is_sub_range</function_ref>
                  <arg>
                <expression>
                  <qualified_attribute_ref>
                    <documentation>RLF note: these are necessarily in
                      reverse to the original!</documentation>

<attribute_ref>climatic_temperature_range</attribute_ref>
                    <qualified_attribute_ref>
                      <attribute_ref>the_garden</attribute_ref>
                      <attribute_ref>the_greenhouse</attribute_ref>
                    </qualified_attribute_ref>
                  </qualified_attribute_ref>
                </expression>
                  </arg>
                  <arg>
                <expression>
                  <qualified_attribute_ref>

<attribute_ref>survival_temperature_range</attribute_ref>
                    <partial_entity_ref>
                      <entity_ref>plant</entity_ref>
                    </partial_entity_ref>
                  </qualified_attribute_ref>
                </expression>
                  </arg>
                </function_call>
              </expression>
            </arg>
          </not>
        </expression>
      </domain_rule>
    </where_clause>
  </entity_decl>
  <entity_decl>
    <entity_id>outdoors_plant</entity_id>
    <subtype_of>
      <entity_ref>plant</entity_ref>
    </subtype_of>
    <explicit_attr_block>
      <explicit_attr>
        <attribute_id>survival_ph_range</attribute_id>
        <base_type>
          <entity_ref>ph_range</entity_ref>
        </base_type>
      </explicit_attr>
    </explicit_attr_block>
    <inverse_clause>
```

```
      <inverse_attr>
        <attribute_id>the_beds</attribute_id>
        <entity_ref>bed</entity_ref>
        <attribute_ref>holds_plants</attribute_ref>
        <inverse_set>
          <bound_spec>
            <lower_bound>
                <integer_literal>1</integer_literal>
            </lower_bound>
            <upper_bound><indeterminate/>
            </upper_bound>
          </bound_spec>
        </inverse_set>
      </inverse_attr>
    </inverse_clause>
    <where_clause>
      <domain_rule>
        <documentation>The ph_range of the outdoors plant must include the
ph
          value of the bed</documentation>
        <label>r1</label>
        <expression>
          <query>
            <variable_id>b</variable_id>
            <aggregate_source>
                <attribute_ref>the_beds</attribute_ref>
            </aggregate_source>
            <logical_expression>
              <expression>
                <equal>
                  <arg>
                <expression>
                  <function_call>
                    <function_ref>value_is_within_range</function_ref>
                    <arg>
                      <expression>
                        <qualified_attribute_ref>
                          <attribute_ref>acidity</attribute_ref>
                          <variable_ref>b</variable_ref>
                        </qualified_attribute_ref>
                      </expression>
                    </arg>
                    <arg>
                      <expression>
                        <attribute_ref>survival_ph_range</attribute_ref>
                      </expression>
                    </arg>
                  </function_call>
                </expression>
                  </arg>
                  <arg>
                <expression>
                  <attribute_ref>the_beds</attribute_ref>
                </expression>
                  </arg>
                </equal>
              </expression>
            </logical_expression>
          </query>
        </expression>
      </domain_rule>
      <domain_rule>
        <documentation>An outdoors plant can survive in the garden
          temperature</documentation>
        <label>r2</label>
        <expression>
          <function_call>
            <function_ref>is_sub_range</function_ref>
            <arg>
              <expression>
                <qualified_attribute_ref>
                <attribute_ref>climatic_temperature_range</attribute_ref>
                <qualified_attribute_ref>
                <attribute_ref>the_garden</attribute_ref>
                <qualified_attribute_ref>
                  <attribute_ref>the_beds</attribute_ref>
                  <index_qualifier>
                    <low_index>
                      <integer_literal>1</integer_literal>
                    </low_index>
                      </index_qualifier>
                </qualified_attribute_ref>
                </qualified_attribute_ref>
                </qualified_attribute_ref>
              </expression>
            </arg>
            <arg>
              <expression>
                <qualified_attribute_ref>
                <attribute_ref>survival_temperature_range</attribute_ref>
                <partial_entity_ref>
                <entity_ref>plant</entity_ref>
                </partial_entity_ref>
                </qualified_attribute_ref>
              </expression>
            </arg>
          </function_call>
        </expression>
      </domain_rule>
    </where_clause>
  </entity_decl>
  <function_decl>
    <function_id>value_is_within_range</function_id>
    <formal_parameter_block>
      <formal_parameter>
        <parameter_id>v</parameter_id>
        <real>
        </real>
      </formal_parameter>
      <formal_parameter>
        <parameter_id>r</parameter_id>
        <entity_ref>real_value_range</entity_ref>
      </formal_parameter>
    </formal_parameter_block>
    <function_return_type><boolean/>
    </function_return_type>
    <statement_block>
      <return_stmt>
        <expression>
          <and>
            <arg>
              <expression>
                <greater_than_or_equal>
                  <arg>
                <expression>
                  <parameter_ref>v</parameter_ref>
                </expression>
                  </arg>
                  <arg>
                <expression>
                  <qualified_attribute_ref>
                    <attribute_ref>minimum_value</attribute_ref>
                    <parameter_ref>r</parameter_ref>
                  </qualified_attribute_ref>
                </expression>
                  </arg>
                </greater_than_or_equal>
              </expression>
            </arg>
            <arg>
              <expression>
                <less_than_or_equal>
                  <arg>
                <expression>
                  <parameter_ref>v</parameter_ref>
                </expression>
                  </arg>
                  <arg>
                <expression>
                  <qualified_attribute_ref>
                    <attribute_ref>maximum_value</attribute_ref>
                    <parameter_ref>r1</parameter_ref>
                  </qualified_attribute_ref>
                </expression>
                  </arg>
                </less_than_or_equal>
              </expression>
            </arg>
          </and>
        </expression>
      </return_stmt>
    </statement_block>
  </function_decl>
  <function_decl>
    <function_id>is_sub_range</function_id>
```

```
<formal_parameter_block>
  <formal_parameter>
    <parameter_id>r1</parameter_id>
    <parameter_id>r2</parameter_id>
    <entity_ref>real_value_range</entity_ref>
  </formal_parameter>
</formal_parameter_block>
<function_return_type><boolean/>
</function_return_type>
<statement_block>
  <return_stmt>
    <expression>
      <and>
        <arg>
          <expression>
            <function_call>
            <function_ref>value_is_within_range</function_ref>
              <arg>
            <expression>
              <qualified_attribute_ref>
                <attribute_ref>minimum_value</attribute_ref>
                <parameter_ref>r1</parameter_ref>
              </qualified_attribute_ref>
            </expression>
              </arg>
              <arg>
            <expression>
                  <parameter_ref>r2</parameter_ref>
            </expression>
              </arg>
            </function_call>
          </expression>
        </arg>
        <arg>
          <expression>
            <function_call>
            <function_ref>value_is_within_range</function_ref>
              <arg>
            <expression>
              <qualified_attribute_ref>
                <attribute_ref>maximum_value</attribute_ref>
                <parameter_ref>r1</parameter_ref>
              </qualified_attribute_ref>
            </expression>
              </arg>
              <arg>
            <expression>
              <parameter_ref>r2</parameter_ref>
            </expression>
              </arg>
            </function_call>
          </expression>
        </arg>
      </and>
    </expression>
```

```
      </return_stmt>
    </statement_block>
  </function_decl>
</schema_decl>
```

F.2 Example schema and data

Specify simple example schema and data set here. What follows is probably too simple but has been used in developing the ideas. It is also too specific to the project leader's household.

```
SCHEMA pets;

ENTITY pet;
SUPERTYPE OF (ONEOF(dog, cat, chinchilla));
  name : STRING;
  owner : person;
END_ENTITY;

ENTITY person ;
  name : STRING;
END_ENTITY;

ENTITY dog SUBTYPE  OF (pet);
END_ENTITY;
ENTITY cat  SUBTYPE  OF (pet);
END_ENTITY;
ENTITY  chinchilla  SUBTYPE  OF (pet);
END_ENTITY;

END_SCHEMA;
```

A simple data set for this schema is (using the syntax of ISO 10303-21):

```
#1=CAT('Whiskey', #10);
#2=DOG('Maddy', #11);
#3=CHINCHILLA('Rita', #12);

#10=PERSON('Nigel Shaw');
#11=PERSON('Andrew Shaw');
#12=PERSON('Iain Shaw');
```

The data population for the example is based on Nigel Shaw’s household. There are two sons, Iain and Andrew, who each own a pet, respectively Rita the chinchilla and Maddy the dog. There is also a cat (Whiskey) which they consider to belong to Nigel.

F.3 Late bound example

The late bound form for both the data specified in clause F.1 is as follows:

This uses the nested form and is therefore an example of data encoded according to the meta-DTD.

```
<?xml version="1.0"?>
<!DOCTYPE ISO-10303-data SYSTEM "late-bound-dtd-v5-exp.dtd">
<ISO-10303-data>
 <data data_id="data_set_1">
  <schema_instance express_schema_name="pets">
   <constant_instances>
   </constant_instances>
   <non_constant_instances>
    <nested_complex_entity_instance express_entity_name="pet"
entity_instance_id="E1">
      <attribute_instance express_attribute_name="name">
       <string_literal>Whiskey</string_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="owner">
        <entity_instance_ref entity_instance_idref="E10"/>
      </attribute_instance>
     <nested_complex_entity_instance_subtype
express_entity_name="cat">
     </nested_complex_entity_instance_subtype>
    </nested_complex_entity_instance>

    <nested_complex_entity_instance express_entity_name="pet"
entity_instance_id="E2" >
      <attribute_instance express_attribute_name="name">
       <string_literal>Maddy</string_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="owner">
        <entity_instance_ref entity_instance_idref="E11"/>
      </attribute_instance>
     <nested_complex_entity_instance_subtype
express_entity_name="dog">
     </nested_complex_entity_instance_subtype>
    </nested_complex_entity_instance>

    <nested_complex_entity_instance express_entity_name="pet"
entity_instance_id="E3">
      <attribute_instance express_attribute_name="name">
       <string_literal>Rita</string_literal>
      </attribute_instance>
      <attribute_instance express_attribute_name="owner">
        <entity_instance_ref entity_instance_idref="E12"/>
      </attribute_instance>
     <nested_complex_entity_instance_subtype
express_entity_name="chinchilla">
     </nested_complex_entity_instance_subtype>
    </nested_complex_entity_instance>
    <nested_complex_entity_instance express_entity_name="person"
entity_instance_id="E10">
```

```
     <attribute_instance express_attribute_name="name">
      <string_literal>Nigel Shaw</string_literal>
     </attribute_instance>
    </nested_complex_entity_instance>
    <nested_complex_entity_instance express_entity_name="person"
entity_instance_id="E11">
     <attribute_instance express_attribute_name="name">
      <string_literal>Andrew Shaw</string_literal>
     </attribute_instance>
    </nested_complex_entity_instance>
    <nested_complex_entity_instance express_entity_name="person"
entity_instance_id="E12">
     <attribute_instance express_attribute_name="name">
      <string_literal>Iain Shaw</string_literal>
     </attribute_instance>
    </nested_complex_entity_instance>
   </non_constant_instances>
  </schema_instance>
 </data>
</ISO-10303-data>
```

F.4 PDML derived early binding

Provide here an example of an early binding based on the meta-DTD. This will be (based on) PDML.

F.4.1 The resulting DTD

Give here

F.4.2 Mapping from EXPRESS

Describe the mapping from EXPRESS used to define the DTD for a specific schema.

F.4.3 Instance Example.

The same data as before encoded according to the DTD given in F.3.1.

It will look something like this:

```
<?xml version="1.0"?>
<!DOCTYPE pets SYSTEM "F:\design docs\DTD development\master DTD
tables\late bound data\pets-eb-v1.dtd">
<pets id="Example_schema_1">
 <pet id="E1">
  <pet.name>
   <string>Whiskey</string>
  </pet.name>
  <pet.owner><person-ref refid="E10"/>
  </pet.owner> <cat id="E1.1"/>
 </pet>
```

```
  <pet id="E2">
   <pet.name>
    <string>Maddy</string>
   </pet.name>
   <pet.owner><person-ref refid="E11"/>
   </pet.owner><dog id="E2.1"/>
  </pet>
  <pet id="E3">
   <pet.name>
    <string>Rita</string>
   </pet.name>
   <pet.owner><person-ref refid="E12"/>
   </pet.owner><chinchilla id="E3.1"/>
  </pet>
  <person id="E10">
   <person.name>
    <string>Nigel Shaw</string>
   </person.name>
  </person>
  <person id="E11">
   <person.name>
    <string>Andrew Shaw</string>
   </person.name>
  </person>
  <person id="E12">
   <person.name>
    <string>Iain Shaw</string>
   </person.name>
  </person>
 </pets>
```

F.5 Another early bound example

It is proposed here to present a different early-bound DTD, probably hand-crafted that is also compliant with the meta-DTD. This will use a different set of design decisions/optimisation from PDML.

F.5.1 The resulting DTD

F.5.2 Mapping from EXPRESS

F.5.3 Instance Example.

F.6 Comparison

It may be appropriate to provide a commentary on the differences between F.3 and F.4.

Annex G
(informative)

Related initiatives

G.1 XMI

Comment on the relationship or show how XMI can be applied to EXPRESS-driven data.

G.2 ISO TC211

Comment on the relationship.

G.3 ISO TC215

Comment on the relationship.

Bibliography.

[1] ISO 10303-21:1994, *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure.*

[2] ISO 10303-22: 1999, *Industrial automation systems and integration – Product data representation and exchange – Part 22: Implementation method: Standard data access interface specification*.

[3] PDML documentation.

Index