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

**Part 14:
Description methods: The EXPRESS-X
language reference manual**

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

*Partie 14: Méthodes descriptives: Le manuel de référence du langage
EXPRESS-X*



Reference number
ISO 10303-14:2005(E)

Documents
Document Sharing Hub
© ISO 2005

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO 2005

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

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

Published in Switzerland

Contents

1	Scope	1
2	Normative references	2
3	Terms and Definitions	2
3.1	Terms defined in ISO 10303-1	2
3.2	Terms defined in ISO 10303-11	2
3.3	Other definitions.....	3
4	Fundamental principles	4
4.1	Overview	4
4.2	Fundamental principles of the execution model	5
4.2.1	Overview	5
4.2.2	Binding process	5
4.2.3	Instantiation process	6
4.3	Implementation environment	8
5	Conformance requirements	8
5.1	EXPRESS-X conformance classes	8
5.1.1	Overview	8
5.1.2	EXPRESS-X parser conformance classes	9
5.1.3	EXPRESS-X mapping engine conformance classes	9
5.1.4	Consistency checking of EXPRESS-X parsers	9
6	Language specification syntax	10
7	Basic language elements	11
7.1	Overview	11
7.2	Reserved words	11
8	Data types	12
8.1	Overview	12
8.2	View data type.....	12
9	Declarations	12
9.1	Overview	12
9.2	Binding	13
9.2.1	Overview	13
9.2.2	Binding extent	13
9.2.3	Qualification of the binding extent	14
9.2.4	Identification of view and target instances	15
9.2.5	Equivalence classes and the instantiation process	16
9.2.6	Ordering of view and target instances	17
9.3	View declaration	18
9.3.1	Overview	18
9.3.2	View attributes	18
9.3.3	View partitions	19
9.3.4	Constant partitions	20
9.3.5	Dependent views	20
9.3.6	Specifying subtype views	21
9.3.7	Supertype constraints	22
9.4	Map declaration.....	23
9.4.1	Overview	23
9.4.2	Evaluation of the map body	24
9.4.3	Iteration under a single binding instance	24
9.4.4	Map partitions	27
9.4.5	Mapping to a type and its subtypes	28

9.4.6	Explicit declaration of complex entity data types	31
9.4.7	Dependent map	33
9.5	Schema view declaration.....	34
9.6	Schema map declaration	34
9.7	Local declaration	36
9.8	Constant declaration.....	37
9.9	Function declaration.....	37
9.10	Procedure declaration.....	37
9.11	Rule declaration	37
10	Expressions	37
10.1	Overview	37
10.2	View call	39
10.3	Map call.....	41
10.4	Partial binding calls.....	43
10.5	FOR expression	44
10.6	IF expression	47
10.7	CASE expression	47
10.8	Forward path operator	48
10.9	Backward path operator	49
11	Built-in functions	51
11.1	Extent - general function	51
12	Scope and visibility	51
12.1	Overview	51
12.2	Schema view	52
12.3	Schema map	53
12.4	View and dependent view	53
12.5	View partition label.....	53
12.6	View attribute identifier	53
12.7	FOR expression	54
12.8	Map and dependent map	54
12.9	FROM Language Element	54
12.10	Instantiation Loop	54
12.11	Path expression.....	55
13	Interface specification	55
13.1	Overview	55
13.2	The REFERENCE language element.....	55
Annex A	(normative) Information object registration	57
Annex B	(normative) EXPRESS-X language syntax	58
B.1	Tokens	58
B.2	Grammar rules	59
B.3	Cross reference listing	65
Annex C	(normative) EXPRESS-X to EXPRESS transformation algorithm	69
Annex D	(informative) Implementation considerations	71
D.1	Push mapping	71
D.2	Pull mapping	71
D.3	Support of constraint checking	71
D.4	Support for updates	71

Annex E (informative) Path operator unnest function	73
Annex F (informative) Mapping table semantics	74
F.1 Delimiter symbols	74
F.2 Aggregation symbols	76
F.3 Equal sign	77
F.4 Parentheses	77
F.5 Square brackets	78
F.6 Example	78
Bibliography.....	80
Index.....	81

Tables

Table 1-Language Subsets	8
Table 2-Additional EXPRESS-X keywords	11
Table 3-Operator precedence	38
Table 4-Scope and identifier defining items	51

Foreword

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

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

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

Attention is drawn to the possibility that some of the elements of this part of ISO 10303 may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 10303-14 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 structure of this International Standard is described in ISO 10303-1.

Each part of this International Standard is a member of one of the following series: description methods, implementation methods, conformance testing methodology and framework, integrated generic resources, integrated application resources, application protocols, abstract test suites, application interpreted constructs, and application modules. This part is a member of the 10 series.

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

[http://www.tc184-sc4.org/SC4_Open/SC4_Work_Products_Documents/STEP_\(10303\)](http://www.tc184-sc4.org/SC4_Open/SC4_Work_Products_Documents/STEP_(10303))

Annexes A, B and C form an integral part of this part of ISO 10303. Annexes D, E and F 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. The mechanism is suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases, and as a basis for archiving.

This part of ISO 10303 is a member of the 10 series. This part of ISO 10303 specifies a language for specifying relationships between data that is governed by EXPRESS schemas, and for specifying alternative views of data that is governed by EXPRESS schemas. The language is called EXPRESS-X.

It is assumed that readers of this part of ISO 10303 are familiar with the EXPRESS data specification language defined in ISO 10303-11 and with clear text encoding specification defined in ISO 10303-21.

Copyright International Organization for Standardization

Industrial automation systems and integration — Product data representation and exchange — Part 14: Description methods: The EXPRESS-X language reference manual

1 Scope

This part of ISO 10303 specifies a language for specifying relationships between data that is governed by EXPRESS schemas, and for specifying alternate views of data that is governed by EXPRESS schemas. The language is called EXPRESS-X.

EXPRESS-X is a structural data mapping language. It consists of language elements that allow an unambiguous specification of a relationship between EXPRESS schemas.

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

- mapping of data governed by one EXPRESS schema to data governed by another EXPRESS schema;
- mapping of data governed by one version of an EXPRESS schema to data governed by another version of that EXPRESS schema, where the two schemas have different names;
- specification of requirements for data translators for data sharing and data exchange applications;
- specification of alternate views of data defined by an EXPRESS schema;
- an alternate notation for application protocol mapping tables;
- bi-directional mappings where mathematically possible;
- specification of constraints that may be evaluated against data produced by mapping.

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

- mapping of data defined using means other than EXPRESS;
- identification of the version of an EXPRESS schema;
- graphical representation of constructs in the EXPRESS-X language.

2 Normative references

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

ISO/IEC 8824-1:2002, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation — Part 1*.

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

ISO 10303-11:2004, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual*.

ISO/IEC 10646-1:2000, *Information technology — Universal Multi-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*.

3 Terms and Definitions

3.1 Terms defined in ISO 10303-1

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

- data;
- information.

3.2 Terms defined in ISO 10303-11

For the purpose of this part of ISO 10303, the following terms defined in ISO 10303-11 apply:

- complex entity data type;
- complex entity (data type) instance;
- constant;
- entity;
- entity data type;
- entity (data type) instance;
- instance;
- partial complex entity data type;

- partial complex entity value;
- population;
- simple entity (data type) instance;
- subtype/supertype graph;
- token;
- value.

3.3 Other definitions

For the purpose of this part of ISO 10303, the following definitions apply:

3.3.1

binding extent

a set of binding instances constructed from instances in source entity data type extents and view extents

3.3.2

binding instance

a collection of references to entity data type instances and view data type instances associated with a view or map.

3.3.3

entity data type extent

the collection of instances of a given entity data type

3.3.4

EXPRESS-X parser

a tool capable of parsing a specification stated in the EXPRESS-X language

3.3.5

EXPRESS-X mapping engine

a tool that performs structural information mapping based on a specification stated in the EXPRESS-X language

3.3.6

map

the declaration of a relationship between data of one or more source entity data types or source view data types and data of one or more target entity data types

3.3.7

network mapping

a mapping to many target entity instances

3.3.8

qualified binding extent

a subset of a binding extent consisting of binding instances satisfying a set of selection criteria

NOTE A set of selection criteria is satisfied if each selection criterion individually is satisfied.

3.3.9

selection criterion

a logical expression, the criterion being satisfied only if the expression evaluates to the value TRUE

3.3.10

source data set

a collection of entity data type instances governed by an EXPRESS schema and serving as an origin of mapping

3.3.11

source extent

a view extent or entity data type extent drawn on to create a binding extent

3.3.12

target data set

a collection of entity instances produced by means of mapping

3.3.13

view

an alternative organization of the information in an EXPRESS schema

3.3.14

view data type

the representation of a view

3.3.15

view data type instance

a named unit of information established by evaluation of a view

3.3.16

view extent

an aggregate of view data type instances that contains all instances that can be constructed from the qualified binding extent

4 Fundamental principles

4.1 Overview

The following principles apply to this part of ISO 10303; the concepts described in ISO 10303-11, clause 5 also apply.

EXPRESS-X provides the specification of:

- differing views of the data governed by an EXPRESS schema, using view declarations (see 9.3) in a schema view (see 9.5);
- the mapping of data governed by one or more source EXPRESS schemas into data governed by one or more target EXPRESS schemas, using map declarations (see 9.4) in a schema map (see 9.6).

An EXPRESS-X schema may contain EXPRESS function and procedure specifications in order to support the definition of views and maps.

NOTE 1 A typographical convention used throughout this specification is to contextualize parts of a sentence to either the discussion of views or maps particularly by including the terms “(views)” or “(maps)” were appropriate in the sentence.

NOTE 2 A typographical convention used throughout this specification is that a binding instance is denoted as an ordered set of entity or view instance names separated by commas “,” and enclosed in angle brackets, “<>”. The ordering of instance names corresponds to the order of appearance of the source extent in the FROM language element of the subject view or map declaration.

EXAMPLE Given the view declaration:

```
SCHEMA_VIEW my_person_org_schema_view;
REFERENCE FROM person_and_org_schema;
VIEW person_org;
  FROM p: person; o : organization; -- provides ordering
  SELECT
    name : STRING := p.last_name;
    org : STRING := o.department_name;
END_VIEW;
END_SCHEMA_VIEW;
```

the source express schema:

```
SCHEMA person_and_org_schema;
ENTITY person;
  first_name : STRING;
  last_name : STRING;
END_ENTITY;
ENTITY organization;
  department_name : STRING;
END_ENTITY;
END_SCHEMA;
```

and the data (encoded as defined in ISO 10303-21 — see [2]):

```
#1=PERSON('James','Smith');
#2=PERSON('Fredrick','Jones');
#31=ORGANIZATION('Engineering');
#32=ORGANIZATION('Sales');
```

binding instances for this view and data may be written as below. The concept of binding instances is defined in subsequent clauses and is not necessary to understand the example. Note here, however, that the first element of each binding instance is drawn from the **person** extent and the second element is drawn from the **organization** extent. This ordering corresponds to the order of appearance of **person** and **organization** in the FROM language element of the view:

```
{<#1,#31>,<#1,#32>,<#2,#31>,<#2,#32>}.
```

4.2 Fundamental principles of the execution model

4.2.1 Overview

This specification defines a language and an execution model. The execution model is composed of two phases: a binding process and an instantiation process. The evaluation of views and maps share a common binding process but differ with respect to instantiation.

4.2.2 Binding process

A binding is an environment in which values are given to variables. A binding instance is a structure that binds the variables declared in the FROM language element of a view or map declaration. The FROM language element references source entity extents and view extents. The values bound are taken from these source extents. Each binding instance is a member of the set computed as the Cartesian product of the source extents referenced. The set of binding instances thus computed is the binding extent of that view or map for the given source extents. The variable bindings of a binding instance provide an environment for the evaluation of the body of the view or map in the instantiation process, where the data referenced in the binding instance is related to structures created in the target population. Thus each

binding instance corresponds to a view data type instance (views) or target entity data type instances (maps) in the target population.

The source extents of maps and views shall be entity data type extents or view extents.

Circularity among references to source extents is prohibited.

EXAMPLE 1 The binding process applied to the view, data and schema defined in the example of 4.1 computes a binding extent of **person_org** {<#1,#31>, <#1,#32>, <#2,#31>, <#2,#32>}. This extent is depicted in tabular form below:

Binding Instance	person			organization	
	#	first_name	last_name	#	department_name
<#1,#31>	#1	'James'	'Smith'	#31	'Engineering'
<#1,#32>	#1	'James'	'Smith'	#32	'Sales'
<#2,#31>	#2	'Fredrick'	'Jones'	#31	'Engineering'
<#2,#32>	#2	'Fredrick'	'Jones'	#32	'Sales'

EXAMPLE 2 The following schema_view is invalid; it contains a cycle of references (view **a** references view **b** which references view **a**).

```
SCHEMA_VIEW invalid;
VIEW a;
  FROM some_b : b;
  attr1 : INTEGER := some_b.attr2 + 2;
END_VIEW;
VIEW b;
  FROM some_a : a;
  attr2 : INTEGER := some_a.attr1 * 3;
END_VIEW;
END_SCHEMA_VIEW;
```

4.2.3 Instantiation process

A binding is an environment in which variables are given values used during the instantiation process. Each binding instance provides a set of values to be bound to the variables. The view instantiation process is the process of evaluating the body of the view (see 9.3.2) for each binding instance in the binding extent. The order of evaluation of the binding instances is not specified.

EXAMPLE 1 The binding process applied to the schema, data, and view declaration of the example in clause 4.1 results in the binding extent of **person_org**: {<#1,#31>, <#1,#32>, <#2,#31>, <#2,#32>}. The view declaration and data used in that example is repeated here:

```
VIEW person_org;
  FROM p: person; o : organization; -- provides the illustrated ordering
  SELECT
    name : STRING := p.last_name;
    org : STRING := o.department_name;
END_VIEW;

#1=PERSON('James','Smith');
#2=PERSON('Fredrick','Jones');
#31=ORGANIZATION('Engineering');
#32=ORGANIZATION('Sales');
```

The binding instance <#1,#31> corresponds to the assignment of entity data type instance #1 to the variable p and #31 to variable o. Evaluation of the body of the view in this binding results in a view data type instance with name attribute 'Smith' and org attribute 'Engineering.' View data type instances may be encoded as though they were entity data type instances using the encoding specified in ISO 10303-21 [2]. The view extent for this example is:

```
#100=PERSON_ORG('Smith','Engineering');    /* <#1,#31> */
#101=PERSON_ORG('Smith','Sales');           /* <#1,#32> */
#102=PERSON_ORG('Jones','Engineering');     /* <#2,#31> */
#103=PERSON_ORG('Jones','Sales');           /* <#2,#32> */
```

EXAMPLE 2 A target EXPRESS schema and schema map with structure similar to that of the schema view used in the previous example can be defined as follows:

```
SCHEMA similar_target;
ENTITY person_org;
  name : STRING;
  org : STRING;
END_ENTITY;
END_SCHEMA;

SCHEMA_MAP similar;
REFERENCE FROM person_and_org_schema AS SOURCE;
REFERENCE FROM similar_target AS TARGET;
MAP person_org_map AS
  po : person_org;
FROM
  p : person;
  o : organization;
SELECT
  po.name := p.last_name;
  po.org := o.department_name;
END_MAP;
END_SCHEMA_MAP;
```

Evaluation of the data of the previous example results in the following entity data type instances governed by the schema similar_target:

```
#100=PERSON_ORG('Smith','Engineering');    /* <#1,#31> */
#101=PERSON_ORG('Smith','Sales');           /* <#1,#32> */
#102=PERSON_ORG('Jones','Engineering');     /* <#2,#31> */
#103=PERSON_ORG('Jones','Sales');           /* <#2,#32> */
```

When the expression in the right-hand-side of an assignment of the body of the map (view) declaration consists only of attribute references (.) from a source entity, and the referenced attribute is an explicit attribute, the mapping of that target entity attribute (view attribute) is bi-directional.

EXAMPLE 3

```
MAP person_org_map AS
  po : person_org;
FROM
  p : person;
  o : organization;
SELECT
  po.name := p.last_name;           -- bi-directional
  po.org := o.department_name;      -- bi-directional
  po.industry_code := o.owning_enterprise.industry.code_num; -- bi-directional
  po.dept_number := dept_func(o.department_name); -- possibly not bi-directional
END_MAP;
```

Whether or not the attribute po.dept_number is bi-directional depends on the nature of the dept_func function.

Clause 4.2 specifies only the fundamental aspects of the execution model. Details of the binding process are described in clause 9.2. Details of the instantiation process for views are described in clause 9.3. Details of the instantiation process for maps are described in clause 9.4.

4.3 Implementation environment

The EXPRESS-X language does not describe an implementation environment. In particular, EXPRESS-X does not specify:

- how references to names are resolved;
- how input and output data sets are specified;
- how maps are executed for instances that do not conform to an EXPRESS schema.

Evaluation of a view produces a view extent. Evaluation of a map may produce entity instances in the target data set. EXPRESS-X does not specify what effect modification of source data may have on view extents or target data sets after initial mapping.

5 Conformance requirements

5.1 EXPRESS-X conformance classes

5.1.1 Overview

The conformance class of an implementation of an EXPRESS-X parser or mapping engine is indicated by what portion of the language the implementation supports. Declarations are classified by language subset in Table 1 below.

Table 1: Declarations and language subset

Declaration	Subset 1	Subset 2
view declaration	•	
map declaration		•
dependent map declaration		•
constant declaration	•	•
function declaration	•	•
procedure declaration	•	•
rule declaration	•	

The implementor of an EXPRESS-X parser or mapping engine shall state any constraints that the implementation imposes on the number and length of identifiers, on the range of processed numbers, and on the maximum precision of real numbers. Such constraints shall be documented for the purpose of conformance testing.

5.1.2 EXPRESS-X parser conformance classes

An implementation of an EXPRESS-X parser shall be able to parse any formal specification written in EXPRESS-X consistent with the conformance class associated with that implementation. An EXPRESS-X parser shall be said to conform to a particular level of checking (as defined in 5.1.4) if it can apply all checks required by that level (and any level below it) to a formal specification written in EXPRESS-X.

A conformance class 1 conforming EXPRESS-X parser shall parse all declarations from language subset 1 (see Table 1).

A conformance class 2 conforming EXPRESS-X parser shall parse all declarations from language subset 2 (see Table 1).

A conforming class 3 conforming EXPRESS-X parser shall parse all declarations that may appear in this part of ISO 10303.

5.1.3 EXPRESS-X mapping engine conformance classes

An implementation of an EXPRESS-X mapping engine shall be able to execute any formal specification written in EXPRESS-X consistent with the conformance class associated with that implementation. The execution of a map is relative to one or more source data sets; the specification of how these data sets are made available to the mapping engine is outside the scope of this part of ISO 10303.

A conformance class 1 conforming EXPRESS-X mapping engine shall be able to execute all declarations from language subset 1 (see Table 1).

A conformance class 2 conforming EXPRESS-X mapping engine shall be able to execute all declarations from language subset 2 (see Table 1).

A conforming class 3 conforming EXPRESS-X mapping engine shall be able to execute all declarations that may appear in this part of ISO 10303.

5.1.4 Consistency checking of EXPRESS-X parsers

5.1.4.1 Overview

A formal specification written in EXPRESS-X shall be consistent with a given level of checking as specified below. A formal specification is consistent with a given level when all checks identified for that level as well as all lower levels are verified for the specification.

5.1.4.2 Level 1: reference checking

This level consists of checking the formal specification to ensure that it is syntactically and referentially valid. A formal specification is syntactically valid if it matches the syntax generated by expanding the primary syntax rule (`syntax`) given in Annex A. A formal specification is referentially valid if all references to EXPRESS-X items are consistent with the scope and visibility rules defined in clause 13.

5.1.4.3 Level 2: type checking

This level consists of Level 1 checking and checking the formal specification to ensure that it is consistent with the following:

- expressions shall comply with the rules specified in clause 10 and in ISO 10303-11 clause 12;
- assignments shall comply with the rules specified in ISO 10303-11 clause 13.3.

5.1.4.4 Level 3: value checking

This level consists of Level 2 checking and checking the formal specification to ensure that it is consistent with statements of the form, 'A shall be greater than B', as specified in clause 7 to 14 of ISO 10303-11. This is limited to those places where both A and B can be evaluated from literals and/or constants.

5.1.4.5 Level 4: complete checking

This level consists of checking the formal specification to ensure that it is consistent with all stated requirements as specified in this part of ISO 10303.

6 Language specification syntax

The notation used to present the syntax of the EXPRESS-X language is defined in this clause.

The full syntax for the EXPRESS-X language is given in Annex B. Portions of those syntax rules are reproduced in various clauses to illustrate the syntax of a particular statement. Those portions are not always complete. It will sometimes be necessary to consult Annex A for the missing rules. The syntax portions within this part of ISO 10303 are presented in a box. Each rule within the syntax box has a unique number toward the left margin for use in cross-references to other syntax rules.

The syntax of EXPRESS-X is defined in a derivative of Wirth Syntax Notation (WSN).

NOTE See the bibliography for a reference describing Wirth Syntax Notation [1].

The notational conventions and WSN defined in itself are given below.

```
syntax = { production } .
production = identifier '=' expression '.' .
expression = term { '|' term } .
term = factor { factor } .
factor = identifier | literal | group | option | repetition .
identifier = character { character } .
literal = ''' character { character } ''' .
group = '(' expression ')' .
option = '[' expression ']' .
repetition = '{' expression '}' .
```

- The equal sign '=' indicates a production. The element on the left is defined to be the combination of the elements on the right. Any spaces appearing between the elements of a production are meaningless unless they appear within a literal. A production is terminated by a period '.'.
- The use of an identifier within a factor denotes a nonterminal symbol that appears on the left side of another production. An identifier is composed of letters, digits, and the underscore character. The keywords of the language are represented by productions whose identifier is given in uppercase characters only.

- The word **literal** is used to denote a terminal symbol that cannot be expanded further. A literal is a sequence of characters enclosed in apostrophes. For an apostrophe to appear in a literal it must be written twice, that is, `' ' ' '`.
- The semantics of the enclosing braces are defined below:
 - curly brackets `' { } '` indicates zero or more repetitions;
 - square brackets `' [ ] '` indicates optional parameters;
 - parenthesis `' ( ) '` indicates that the group of productions enclosed by parenthesis shall be used as a single production;
 - vertical bar `' | '` indicates that exactly one of the terms in the expression shall be chosen.

The following notation is used to represent entire character sets and certain special characters which are difficult to display:

- \a represents any character from ISO/IEC 10646-1;
- \n represents a newline (system dependent) (see ISO 10303-1, 7.1.5.2).

7 Basic language elements

7.1 Overview

This clause specifies the basic elements from which an EXPRESS-X mapping specification is composed: the character set, remarks, symbols, reserved words, identifiers, and literals.

The language elements of EXPRESS-X are those of the EXPRESS language defined in Clause 7 of ISO 10303-11, with the exceptions noted below.

7.2 Reserved words

The reserved words of EXPRESS-X are the keywords and the names of built-in constants, functions, and procedures. All reserved words of EXPRESS (ISO 10303-11) are reserved words of EXPRESS-X. The reserved words shall not be used as identifiers. The additional reserved words of EXPRESS-X are specified in Table 2 below.

Table 2: Additional EXPRESS-X keywords

DEPENDENT_MAP	EACH	ELSIF	END_DEPENDENT_MAP
END_MAP	END_SCHEMA_MAP	END_SCHEMA_VIEW	END_VIEW
EXTENT	IDENTIFIED_BY	INDEXING	MAP
ORDERED_BY	PARTITION	SCHEMA_MAP	SCHEMA_VIEW
SOURCE	TARGET	VIEW	

NOTE In the case that a legal EXPRESS identifier is a reserved word in EXPRESS-X, schemas using that identifier can be mapped by renaming the conflicting identifier using the REFERENCE language element (see 13.2).

8 Data types

8.1 Overview

The data types defined here as well as those defined in the EXPRESS language (clause 8 of ISO 10303-11) are provided as part of the language.

Every view attribute (see 9.3.2) has an associated data type.

8.2 View data type

View data types are established by view declarations (see 9.3). A view data type is assigned an identifier in the defining schema map or schema view. The view data type is referenced by this identifier.

Syntax:

```
230 view_reference = [ ( schema_map_ref | schema_view_ref ) '.' ] view_ref
      .
```

Rules and restrictions:

- a) view_reference shall be a reference to a view visible in the current scope.
- b) view_reference shall not refer to a dependent view (see 9.3.5).

EXAMPLE The following declaration defines a view data type named circle.

```
VIEW circle;
FROM e : ellipse;
WHERE (e.major_axis = e.minor_axis);
SELECT
  radius : REAL := e.minor_axis;
  centre : point := e.centre;
END_VIEW;
```

9 Declarations

9.1 Overview

This clause defines the various declarations available in EXPRESS-X. An EXPRESS-X declaration creates a new EXPRESS-X item and associates an identifier with it. The item may be referenced elsewhere by this identifier.

EXPRESS-X provides the following declarations:

- view;
- map;
- dependent map;
- schema view;
- schema map.

In addition, an EXPRESS-X specification may contain the following declarations defined in ISO 10303-11:1994:

- constant;
- function;
- procedure;
- rule.

9.2 Binding

9.2.1 Overview

A binding extent is a set of binding instances constructed from instances in source entity data type extents and view extents. A binding extent of a population is computed as the Cartesian product of the extents referenced in the FROM language element of the view or map declaration.

A qualified binding extent is a subset of the binding extent consisting of only those binding instances for which the WHERE language element, under the binding of its variables to the values in the binding instance, returns TRUE.

NOTE Provisions defined in 9.2 apply to map declarations and view declarations. Provisions applying only to view declarations are defined in 9.3. Provisions applying only to map declarations are defined in 9.4.

9.2.2 Binding extent

The FROM language element defines the elements of binding instances in the binding extent. The FROM language element consists of one or more source parameters. Each source parameter associates an identifier with an extent.

Syntax:

```

228 view_decl = ( root_view_decl | dependent_view_decl | subtype_view_decl
) .
136 map_decl = MAP map_id AS target_parameter ';' { target_parameter ';' }
( map_subtype_of_clause subtype_binding_header map_decl_body ) | (
binding_header map_decl_body { binding_header map_decl_body } )
END_MAP ';' .
47 binding_header = [ PARTITION partition_id ';' ] [ from_clause ] [
local_decl ] [ where_clause ] [ identified_by_clause ] [
ordered_by_clause ] .
90 from_clause = FROM source_parameter ';' { source_parameter ';' } .
198 source_parameter = source_parameter_id ':' extent_reference .
83 extent_reference = source_entity_reference | view_reference .

```

Rules and restrictions:

- a) `source_parameter_ids` shall be unique within the scope of the map or view declaration.

The binding extent is computed as the Cartesian product of instances in the extents referenced in the `FROM` language element.

EXAMPLE A binding extent is constructed over the populations of entity data types **item** and **person**.

```

SCHEMA source_schema; -- An EXPRESS schema
ENTITY item;
    item_number : INTEGER;
    approved_by : STRING;
END_ENTITY;
ENTITY person;
    name : STRING;
END_ENTITY;
END_SCHEMA;

SCHEMA_VIEW example;
REFERENCE FROM source_schema;
VIEW items_and_persons;
    FROM i : item; p : person;
    SELECT
        item_number : INTEGER := i.item_number;
        responsible : STRING := p.name;
END_VIEW;
END_SCHEMA_VIEW;

```

Given a population (written as ISO 10303-21 entity instances — see [2]):

```

#1=ITEM(123, 'Smith');
#2=ITEM(234, 'Smith');
#33=PERSON('Jones');
#44=PERSON('Smith');

```

the corresponding binding extent of `items_and_person` is: {<#1,#33>,<#1,#44>,<#2,#33>,<#2,#44>}.

9.2.3 Qualification of the binding extent

The `WHERE` language element defines selection criteria on binding instances in the binding extent. The `WHERE` language element, together with the source extents identified in the `FROM` language element define the qualified binding extent.

A binding is an environment in which values are given to variables. The source parameters of the `FROM` language element are bound to the binding instance. The domain rule expressions of the `WHERE` language element are evaluated with this binding. A binding instance in the binding extent is a member of the qualified binding extent if all the domain rule expressions of the `WHERE` language element evaluate to `TRUE`.

The syntax of the `WHERE` language element is as defined in ISO 10303-11, 9.2.2.2.

EXAMPLE The qualified binding extent consists of those pairs of **item** and **person** of the binding extent for which **person.name** is 'Smith' or 'Jones' and **item.approved_by** is 'Smith' or 'Jones'.

```
SCHEMA_VIEW example;
REFERENCE FROM source_schema;
VIEW items_and_persons;
  FROM i : item; p : person;
  WHERE (p.name = 'Smith') OR (p.name = 'Jones');
        (i.approved_by = p.name);
  SELECT
    name : STRING := p.name;
END_VIEW;
END_SCHEMA_VIEW;
```

The qualified binding extent corresponding to the data in example 1 above is: {<#1,#44>,<#2,#44>}.

9.2.4 Identification of view and target instances

A binding instance of a map or view not containing the `identified_by_clause` language element is identified by the values (entity data type instances) it takes from the extents referenced in the `FROM` language element. A binding instance of a map or view containing the `identified_by_clause` language element is identified by the value(s) of the `expression` language element in syntax rule 108. The view call (see 10.2) and map call (see 10.3) use these identification schemes.

The `identified_by_clause` language element defines an equivalence relation among instances in a binding extent.

Syntax:

```
107 identified_by_clause = IDENTIFIED_BY id_parameter ';' { id_parameter
    ';' } .
108 id_parameter = [ id_parameter_id ':' ] expression .
109 id_parameter_id = simple_id .
    90 from_clause = FROM source_parameter ';' { source_parameter ';' } .
198 source_parameter = source_parameter_id ':' extent_reference .
```

Rules and restrictions:

- a) When used in a map declaration, an `expression` in the `id_parameter` language syntax shall not refer, through any level of indirection, to the target entity instances of the map or any of their attributes.

Two binding instances are in the same equivalence class if, for each `expression` of the `identified_by_clause` language element, evaluating the expression in the context of each of those instances produces result that are instance equal (ISO 10303-11, 12.2.2). The instantiation process produces one view instance (views) or target network (maps) for each equivalence class.

EXAMPLE This example illustrates the use of IDENTIFIED_BY.

```

SCHEMA_VIEW example;
REFERENCE FROM some_schema;
VIEW department;
  FROM e : employee;
  IDENTIFIED_BY e.department_name;
  SELECT
    name : STRING := e.department_name;
END_VIEW;
END_SCHEMA_VIEW;

SCHEMA some_schema;
ENTITY employee;
  name : STRING;
  department_name : STRING;
END_ENTITY;
END_SCHEMA;

#1=EMPLOYEE('Jones','Engineering');
#2=EMPLOYEE('Smith','Sales');
#3=EMPLOYEE('Doe','Engineering');

```

Given the view and population above, there are two equivalence classes: {<#1>,<#3>} and {<#2>}, corresponding to binding instances with e.department_name = 'Engineering' and e.department_name = 'Sales' respectively.

9.2.5 Equivalence classes and the instantiation process

View attributes (see 9.3.2) and target entity attributes (see 9.4.2) represent properties of the corresponding view (view declaration) and target network entities (map declaration). These attributes are provided values by evaluation of the corresponding expression (syntax rule 224). The expressions are evaluated in the context of a binding instance in the qualified binding extent.

Syntax:

```

224 view_attribute_decl = view_attribute_id ':' [ OPTIONAL ] [
  source_schema_ref '.' ] base_type ':' expression ';' .
134 map_attribute_declaration = [ target_parameter_ref [ index_qualifier ]
  [ group_qualifier ] '.' ] attribute_ref [ index_qualifier ] ':' expression ';' .

```

If an equivalence class defined by an `identified_by_clause` language element contains more than one qualified binding instance, then the value of the `expression` is computed as follows:

- if there are binding instances for which `expression` (syntax rule 224) evaluates to a non-indeterminate value, and if all such non-indeterminate values are the same (instance equal) or if there is only one such value, that value is assigned to the attribute.
- If for two or more binding instances, the evaluation of the `expression` of the attribute produces non-indeterminate, unequal values (instance equal), or if all evaluations produce indeterminate values, an indeterminate value is assigned to the attribute.

EXAMPLE This example illustrates the assignment of values where an equivalence class contain more than one qualified binding instance. The map declaration is described in clause 9.4.

```
(* source schema *)          (* target schema *)
SCHEMA src;                   SCHEMA tar;
ENTITY employee;              ENTITY department;
  name : STRING;               employee : STRING;
  manager : STRING;            manager : STRING;
  dept : STRING;               dept_name : STRING;
END_ENTITY;                   END_ENTITY;
END_SCHEMA;                   END_SCHEMA;

(* mapping schema *)
SCHEMA_MAP example;
REFERENCE FROM src AS SOURCE;
REFERENCE FROM tar AS TARGET;
MAP department_map AS d : department;
  FROM e : src.employee;
  IDENTIFIED_BY e.dept;
  SELECT
    d.employee := e.name;
    d.manager := e.manager;
    d.dept_name := e.dept;
END_MAP;
END_SCHEMA_MAP;

#1=EMPLOYEE('Smith','Jones','Marketing');
#2=EMPLOYEE('Doe','Jones','Marketing');
```

Given the data above, the target data set contains one entity instance, #1=DEPARTMENT(\$, 'Jones','Marketing'). The attribute **department.dept_name** is indeterminate because the expression for this attribute evaluates to two different values ('Smith' and 'Doe').

9.2.6 Ordering of view and target instances

The `ORDERED_BY` language element defines an ordering on the binding instances of a qualified binding extent. Partial binding calls (see 10.4) on the corresponding view or map partition evaluate to aggregates ordered according to the `ORDERED_BY` language element, if present.

The `expression` in an `ORDERED_BY` language element shall evaluate to values of the same order-comparable type for all bindings in the partition. The order-comparable types are `NUMBER`, `BINARY`, `STRING`, and `ENUMERATION` and specializations of these. A resulting ordering on the binding extent shall be such that the expression ($e < f$) shall not evaluate to `FALSE`, where e and f are respectively the values produced by evaluation of the `expression` (see syntax rule 148) for any two successive elements of the binding extent, and `<` is the `EXPRESS` value comparison operator (ISO 10303-11, 12.2.1). If additional `expression` language elements are specified, then for each subsequent `expression`, the process describe is applied to the result of the previous `expression`.

Syntax:

```
47 binding_header = [ PARTITION partition_id ';' ] [ from_clause ] [
  local_decl ] [ where_clause ] [ identified_by_clause ] [
  ordered_by_clause ] .
148 ordered_by_clause = ORDERED_BY expression { ',' expression } ';' .
```

Rules and restrictions:

- a) The `expression` of syntax rule 148 shall not evaluate to an indeterminate value for any bindings in the partition.

- b) Subtype view and map partitions shall not specify an `ORDERED_BY` language element. Subtype views and map partitions shall inherit ordering from parent partitions defining `ORDERED_BY`, if present in a parent.
- c) A collection of view partitions related through type hierarchy shall contain no more than one `ORDERED_BY` language element.
- d) A collection of map partitions related through the `map_subtype_of_clause` (see syntax rule 141) shall contain no more than one `ORDERED_BY` language element.

9.3 View declaration

9.3.1 Overview

A view declaration creates a view data type and declares an identifier to refer to it.

EXAMPLE The following view defines a view data type `arm_person_role_in_organization`.

```
VIEW arm_person_role_in_organization;
FROM
  pao : person_and_organization;
  ccdpaoa : cc_design_person_and_organization_assignment;
WHERE ccdpaoa.assigned_person_and_organization :=: pao;
SELECT
  person : person := pao.the_person;
  org : organization := pao.the_organization;
  role : label := ccdpaoa.role.name;
END_VIEW;
```

Syntax:

```
228 view_decl = ( root_view_decl | dependent_view_decl | subtype_view_decl
  ) .
177 root_view_decl = VIEW view_id [ supertype_constraint ] ';'
  binding_header SELECT view_attr_decl_stmt_list { binding_header SELECT
  view_attr_decl_stmt_list } END_VIEW ';' .
  47 binding_header = [ PARTITION partition_id ';' ] [ from_clause ] [
  local_decl ] [ where_clause ] [ identified_by_clause ] [
  ordered_by_clause ] .
  90 from_clause = FROM source_parameter ';' { source_parameter ';' } .
198 source_parameter = source_parameter_id ':' extent_reference .
  83 extent_reference = source_entity_reference | view_reference .
```

9.3.2 View attributes

An attribute of a view data type represents a property of the view. The value of the attribute of a view instance is computed as the evaluation of the expression of syntax rule 224.

The name of a view attribute, `view_attribute_id` of syntax rule 224, represents the role played by its associated value in the context of the view in which it appears.

Syntax:

```

226 view_attr_decl_stmt_list = { view_attribute_decl } .
224 view_attribute_decl = view_attribute_id ':' [ OPTIONAL ] [
    source_schema_ref '.' ] base_type ':= ' expression ';' .

```

Rules and restrictions:

- a) The value resulting from evaluation of the expression of syntax rule 224 shall be assignment compatible with the `base_type` of the view attribute.
- b) Each `view_attribute_id` declared in the view declaration shall be unique within that declaration.
- c) `OPTIONAL` indicates that the value of the attribute may be indeterminate. Use of `OPTIONAL` has no effect on the execution model.

9.3.3 View partitions

A view partition is a subset of a view extent. A view extent is the union of its partitions. A view declaration consists of one or more partition declarations, each partition declaration having its own `FROM` and `WHERE` language elements.

EXAMPLE In ISO 10303-201, the application object **ORGANIZATION** may be mapped to from a **PERSON**, an **ORGANIZATION**, or a **PERSON_AND_ORGANIZATION** entity. A view from this schema to the `arm_organization` view is specified as follows:

```

VIEW arm_organization;
PARTITION a_single_person;
    FROM p : person;
    ...
PARTITION a_single_organization;
    FROM o: organization;
    ...
PARTITION a_person_in_an_organization;
    FROM po: person_and_organization;
    ...
END_VIEW;

```

Syntax:

```

228 view_decl = ( root_view_decl | dependent_view_decl | subtype_view_decl
    ) .
177 root_view_decl = VIEW view_id [ supertype_constraint ] ';'
    binding_header SELECT view_attr_decl_stmt_list { binding_header SELECT
    view_attr_decl_stmt_list } END_VIEW ';' .
67 dependent_view_decl = VIEW view_id ':' base_type ';' binding_header
    RETURN expression { binding_header RETURN expression } END_VIEW ';' .
206 subtype_view_decl = VIEW view_id subtype_declaration ';'
    subtype_binding_header SELECT view_attr_decl_stmt_list {
    subtype_binding_header SELECT view_attr_decl_stmt_list } END_VIEW ';'
    .
203 subtype_binding_header = [ PARTITION partition_id ';' ] where_clause .
47 binding_header = [ PARTITION partition_id ';' ] [ from_clause ] [
    local_decl ] [ where_clause ] [ identified_by_clause ] [
    ordered_by_clause ] .

```

Rules and restrictions:

- a) If more than one partition exists in the view declaration, a `partition_id` shall be provided for each partition.
- b) Each `partition_id` within a view declaration shall be unique.
- c) All partitions of a view declaration shall define the same attributes (including names and types).
- d) The attributes of a view declaration shall appear in the same order in each of its partitions.

9.3.4 Constant partitions

A partition that omits the `FROM`, `WHERE`, and `IDENTIFIED_BY` language elements is called a constant partition. Such a partition represents a single view instance with no correspondence to the source data.

EXAMPLE This example illustrates the use of constant partitions.

```
VIEW person;
PARTITION mary;
  SELECT
    name : STRING := 'Mary';
    age : INTEGER := 22;
PARTITION john;
  SELECT
    name : STRING := 'John';
    age : INTEGER := 23;
END_VIEW;
```

9.3.5 Dependent views

A dependent view is a view that does not define attributes. Partitions of the dependent view specify `RETURN expression` (see syntax rule 67). Evaluation of `expression` shall not produce a value of type `AGGREGATE`. The value computed shall be type compatible with `base_type` (see syntax rule 67).

Syntax:

```
228 view_decl = ( root_view_decl | dependent_view_decl | subtype_view_decl
  ) .
67  dependent_view_decl = VIEW view_id ':' base_type ';' binding_header
  RETURN expression { binding_header RETURN expression } END_VIEW ';' .
```

If an equivalence class defined by an `IDENTIFIED_BY` language element of a dependent view contains more than one qualified binding instance, then the value returned is computed as follows:

- If for each such binding, the `RETURN expression` (syntax rule 67) produces an equal value, that value is returned;
- If for two or more bindings, the `RETURN expression` (syntax rule 67) produces unequal values, an indeterminate value is returned.