BS ISO 10303-11:2004

BSI Standards Publication

Industrial automation systems and integration — Product data representation and exchange

Part 11: Description methods: The EXPRESS language reference manual

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

National foreword

This British Standard is the UK implementation of ISO 10303-11:2004.

The UK participation in its preparation was entrusted to Technical Committee AMT/4, Industrial data and manufacturing interfaces.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2023
Published by BSI Standards Limited 2023

ISBN 978 0 539 22242 5

ICS 25.040.40

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 March 2023.

Amendments/corrigenda issued since publication

Date	Text affected

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

INTERNATIONAL STANDARD

ISO 10303-11

Second edition
2004-11-01

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

Part 11: Description methods: The EXPRESS language reference manual

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

Partie 11: Méthodes de description: Manuel de référence du langage EXPRESS

Reference number
ISO 10303-11:2004(E)

Documents
Document Sharing Hub
© ISO 2004

Full document details: BS ISO 10303-11:2004

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 2004

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

Documents
Document Sharing Hub
© ISO 2004 – All rights reserved

Full document details: BS ISO 10303-11:2004

Contents

Page

0 Introduction . . . xii
0.1 General . . . xii
0.2 Language overview . . . xii

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/IEC 10646 . . . 2
3.3 Other terms and definitions . . . 2

4 Conformance requirements . . . 5
4.1 Formal specifications written in EXPRESS . . . 5
4.1.1 Lexical language . . . 5
4.1.2 Graphical form . . . 6
4.2 Implementations of EXPRESS . . . 6
4.2.1 EXPRESS language parser . . . 6
4.2.2 Graphical editing tool . . . 6

5 Fundamental principles . . . 7

6 Language specification syntax . . . 8
6.1 The syntax of the specification . . . 8
6.2 Special character notation . . . 9

7 Basic language elements . . . 9
7.1 Character set . . . 10
7.1.1 Digits . . . 10
7.1.2 Letters . . . 10
7.1.3 Special characters . . . 11
7.1.4 Underscore . . . 11
7.1.5 Whitespace . . . 11
7.1.6 Remarks . . . 11
7.2 Reserved words . . . 14
7.2.1 Keywords . . . 14
7.2.2 Reserved words which are operators . . . 14
7.2.3 Built-in constants . . . 15
7.2.4 Built-in functions . . . 15
7.2.5 Built-in procedures . . . 15
7.3 Symbols . . . 15
7.4 Identifiers . . . 15
7.5 Literals . . . 16
7.5.1 Binary literal . . . 16
7.5.2 Integer literal . . . 17
7.5.3 Real literal . . . 17
7.5.4 String literal . . . 18
7.5.5 Logical literal . . . 19

8 Data types . . . 19
8.1 Simple data types . . . 20

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

8.1.1 Number data type . . . 20
8.1.2 Real data type . . . 20
8.1.3 Integer data type . . . 21
8.1.4 Logical data type . . . 21
8.1.5 Boolean data type . . . 21
8.1.6 String data type . . . 21
8.1.7 Binary data type . . . 22
8.2 Aggregation data types . . . 23
8.2.1 Array data type . . . 24
8.2.2 List data type . . . 25
8.2.3 Bag data type . . . 26
8.2.4 Set data type . . . 26
8.2.5 Value uniqueness on aggregates . . . 27
8.3 Named data types . . . 28
8.3.1 Entity data type . . . 28
8.3.2 Defined data type . . . 29
8.4 Constructed data types . . . 29
8.4.1 Enumeration data type . . . 30
8.4.2 Select data type . . . 33
8.5 Generalized data types . . . 35
8.6 Data type usage classification . . . 35
8.6.1 Instantiable data types . . . 37
8.6.2 Parameter data types . . . 37
8.6.3 Underlying data types . . . 37

9 Declarations . . . 38
9.1 Type declaration . . . 38
9.2 Entity declaration . . . 40
9.2.1 Attributes . . . 41
9.2.2 Local rules . . . 45
9.2.3 Subtypes and supertypes . . . 48
9.2.4 Abstract entity data type . . . 54
9.2.5 Subtype/supertype constraints . . . 55
9.2.6 Implicit declarations . . . 60
9.2.7 Specialization . . . 61
9.3 Schema . . . 62
9.4 Constant . . . 63
9.5 Algorithms . . . 64
9.5.1 Function . . . 64
9.5.2 Procedure . . . 65
9.5.3 Parameters . . . 65
9.5.4 Local variables . . . 71
9.6 Rule . . . 72
9.7 Subtype constraints . . . 74
9.7.1 Abstract supertype constraint . . . 75
9.7.2 Total coverage subtypes . . . 75
9.7.3 Overlapping subtypes and their specification . . . 76

10 Scope and visibility . . . 78
10.1 Scope rules . . . 78
10.2 Visibility rules . . . 79
10.3 Explicit item rules . . . 81
10.3.1 Alias statement . . . 81

 ©ISO 2004 — All rights reserved

AVA Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

10.3.2 Attribute . . . 81
10.3.3 Constant . . . 81
10.3.4 Enumeration item . . . 81
10.3.5 Entity . . . 81
10.3.6 Function . . . 82
10.3.7 Parameter . . . 83
10.3.8 Procedure . . . 83
10.3.9 Query expression . . . 83
10.3.10 Repeat statement . . . 84
10.3.11 Rule . . . 84
10.3.12 Rule label . . . 85
10.3.13 Schema . . . 85
10.3.14 Subtype constraint . . . 86
10.3.15 Type . . . 86
10.3.16 Type label . . . 86
10.3.17 Variable . . . 86

11 Interface specification . . . 86
11.1 Use interface specification . . . 87
11.2 Reference interface specification . . . 87
11.3 The interaction of use and reference . . . 88
11.4 Implicit interfaces . . . 89
11.4.1 Constant interfaces . . . 89
11.4.2 Defined data type interfaces . . . 90
11.4.3 Entity data type interfaces . . . 90
11.4.4 Function interfaces . . . 91
11.4.5 Procedure interfaces . . . 91
11.4.6 Rule interfaces . . . 91
11.4.7 Subtype constraint interfaces . . . 91

12 Expression . . . 92
12.1 Arithmetic operators . . . 93
12.2 Relational operators . . . 94
12.2.1 Value comparison operators . . . 95
12.2.2 Instance comparison operators . . . 99
12.2.3 Membership operator . . . 101
12.2.4 Interval expressions . . . 102
12.2.5 Like operator . . . 102
12.3 Binary operators . . . 103
12.3.1 Binary indexing . . . 103
12.3.2 Binary concatenation operator . . . 104
12.4 Logical operators . . . 104
12.4.1 NOT operator . . . 105
12.4.2 AND operator . . . 105
12.4.3 OR operator . . . 105
12.4.4 XOR operator . . . 105
12.5 String operators . . . 106
12.5.1 String indexing . . . 106
12.5.2 String concatenation operator . . . 107
12.6 Aggregate operators . . . 107
12.6.1 Aggregate indexing . . . 107
12.6.2 Intersection operator . . . 108
12.6.3 Union operator . . . 109

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

12.6.4 Difference operator . . . 109
12.6.5 Subset operator . . . 110
12.6.6 Superset operator . . . 111
12.6.7 Query expression . . . 111
12.7 References . . . 112
12.7.1 Simple references . . . 113
12.7.2 Prefixed references . . . 113
12.7.3 Attribute references . . . 114
12.7.4 Group references . . . 115
12.8 Function call . . . 116
12.9 Aggregate initializer . . . 117
12.10 Complex entity instance construction operator . . . 118
12.11 Type compatibility . . . 119
12.12 Select data types in expressions . . . 120
12.12.1 Select data types in unary expressions . . . 120
12.12.2 Select data types in binary expressions . . . 120
12.12.3 Select data types in ternary expressions . . . 121

13 Executable statements . . . 121
13.1 Null (statement) . . . 121
13.2 Alias statement . . . 122
13.3 Assignment . . . 122
13.3.1 Assignment statement . . . 122
13.3.2 Assignment compatibility . . . 123
13.4 Case statement . . . 126
13.5 Compound statement . . . 127
13.6 Escape statement . . . 127
13.7 If . . . Then . . . Else statement . . . 127
13.8 Procedure call statement . . . 128
13.9 Repeat statement . . . 128
13.9.1 Increment control . . . 129
13.9.2 While control . . . 130
13.9.3 Until control . . . 130
13.10 Return statement . . . 131
13.11 Skip statement . . . 131

14 Built-in constants . . . 132
14.1 Constant e . . . 132
14.2 Indeterminate . . . 132
14.3 False . . . 132
14.4 Pi . . . 132
14.5 Self . . . 132
14.6 True . . . 133
14.7 Unknown . . . 133

15 Built-in functions . . . 133
15.1 Abs - arithmetic function . . . 133
15.2 ACos - arithmetic function . . . 133
15.3 ASin - arithmetic function . . . 133
15.4 ATan - arithmetic function . . . 134
15.5 BLength - binary function . . . 134
15.6 Cos - arithmetic function . . . 134
15.7 Exists - general function . . . 135

 ©ISO 2004 — All rights reserved

AVA Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

15.8 Exp - arithmetic function . . . 135
15.9 Format - general function . . . 135
15.9.1 Symbolic representation . . . 136
15.9.2 Picture representation . . . 137
15.9.3 Standard representation . . . 138
15.10 HiBound - arithmetic function . . . 138
15.11 HiIndex - arithmetic function . . . 138
15.12 Length - string function . . . 139
15.13 LoBound - arithmetic function . . . 139
15.14 Log - arithmetic function . . . 140
15.15 Log2 - arithmetic function . . . 140
15.16 Log10 - arithmetic function . . . 140
15.17 LoIndex - arithmetic function . . . 140
15.18 NVL - null value function . . . 141
15.19 Odd - arithmetic function . . . 141
15.20 RolesOf - general function . . . 142
15.21 Sin - arithmetic function . . . 143
15.22 SizeOf - aggregate function . . . 143
15.23 Sqrt - arithmetic function . . . 143
15.24 Tan - arithmetic function . . . 144
15.25 TypeOf - general function . . . 144
15.26 UsedIn - general function . . . 147
15.27 Value - arithmetic function . . . 148
15.28 Value_in - membership function . . . 148
15.29 Value_unique - uniqueness function . . . 149

16 Built-in procedures . . . 149
16.1 Insert . . . 149
16.2 Remove . . . 150

Annex A (normative) EXPRESS language syntax . . . 151
A.1 Tokens . . . 151
A.1.1 Keywords . . . 151
A.1.2 Character classes . . . 153
A.1.3 Lexical elements . . . 154
A.1.4 Remarks . . . 154
A.1.5 Interpreted identifiers . . . 154
A.2 Grammar rules . . . 155
A.3 Cross reference listing . . . 159

Annex B (normative) Determination of the allowed entity instantiations . . . 166
B.1 Formal approach . . . 166
B.2 Supertype and subtype constraint operators . . . 167
B.2.1 OneOf . . . 168
B.2.2 And . . . 168
B.2.3 AndOr . . . 168
B.2.4 Precedence of operators . . . 168
B.3 Interpreting the possible complex entity data types . . . 168

Annex C (normative) Instance limits imposed by the interface specification . . . 181

Annex D (normative) EXPRESS-G: A graphical subset of EXPRESS . . . 185
D.1 Introduction and overview . . . 185

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

D.2 Definition symbols . . . 185
D.2.1 Symbol for simple data types . . . 186
D.2.2 Symbols for constructed data types . . . 187
D.2.3 Symbols for defined data types . . . 188
D.2.4 Symbols for entity data types . . . 188
D.2.5 Symbols for subtype_constraints . . . 188
D.2.6 Symbols for functions and procedures . . . 188
D.2.7 Symbols for rules . . . 189
D.2.8 Symbols for schemas . . . 189
D.3 Relationship symbols . . . 189
D.4 Composition symbols . . . 190
D.4.1 Page references . . . 190
D.4.2 Inter-schema references . . . 191
D.5 Entity level diagrams . . . 191
D.5.1 Role names . . . 191
D.5.2 Cardinalities . . . 192
D.5.3 Constraints . . . 192
D.5.4 Constructed and defined data types . . . 193
D.5.5 Entity data types . . . 193
D.5.6 Inter-schema references . . . 197
D.6 Schema level diagrams . . . 198
D.7 Complete EXPRESS-G diagrams . . . 198
D.7.1 Complete entity level diagram . . . 199
D.7.2 Complete schema level diagram . . . 200

Annex E (normative) Protocol implementation conformance statement (PICS) . . . 201
E.1 EXPRESS language parser . . . 201
E.2 EXPRESS-G editing tool . . . 201

Annex F (normative) Information object registration . . . 203
F.1 Document identification . . . 203
F.2 Syntax identification . . . 203

Annex G (normative) Generating a single schema from multiple schemas . . . 204
G.1 Introduction . . . 204
G.2 Fundamental concepts . . . 204
G.3 Name munging . . . 206
G.3.1 Name clashes . . . 206
G.3.2 Identifiers as strings . . . 206
G.4 Stage 1: multi-schema to intermediate schema conversion . . . 207
G.4.1 Introduction . . . 207
G.4.2 Primary population . . . 207
G.4.3 Secondary population . . . 209
G.4.4 Prune . . . 216
G.4.5 Schema names and versions . . . 222
G.5 Stage 2: convert intermediate schema to ISO 10303-11:1994 . . . 223
G.5.1 Introduction . . . 223
G.5.2 Initialisation . . . 223
G.5.3 Conversion of extensible constructed data types . . . 223
G.5.4 Conversion of subtype constraints . . . 228
G.5.5 Conversion of abstract entity and generalized types . . . 231
G.5.6 Conversion of attributes renamed in a redeclaration . . . 233

 ©ISO 2004 — All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

Annex H (informative) Relationships . . . 235
H.1 Relationships via attributes . . . 235
H.1.1 Simple relationship . . . 236
H.1.2 Collective relationship . . . 238
H.1.3 Distributive relationship . . . 239
H.1.4 Inverse attribute . . . 240
H.2 Subtype/supertype relationships . . . 241

Annex J (informative) EXPRESS models for EXPRESS-G illustrative examples . . . 242
J.1 Example single schema model . . . 242
J.2 Relationship sampler . . . 243
J.3 Simple subtype/supertype tree . . . 244
J.4 Attribute redeclaration . . . 244
J.5 Multi-schema models . . . 245

Annex K (informative) Deprecated features of EXPRESS . . . 248

Annex L (informative) Examples of the new EXPRESS constructs . . . 249
L.1 Product management example . . . 249

Bibliography . . . 251

Index . . . 252

Figures

Figure B.1 — EXPRESS-G diagram of schema for example 1 on page 171 . . . 172
Figure B.2 — EXPRESS-G diagram of schema for example 2 on page 174 . . . 174
Figure B.3 — EXPRESS-G diagram of schema for example 3 on page 176 . . . 177
Figure D.1 — Complete entity level diagram of the example in J.1 on page 242 (Page 1 of 2) 186
Figure D.2 — Complete entity level diagram of the example in J.1 on page 242 (Page 2 of 2) 186
Figure D.3 — Symbols for EXPRESS simple data types . . . 186
Figure D.4 — Symbol for EXPRESS generic_entity data type . . . 187
Figure D.5 — Symbols for EXPRESS constructed data types . . . 187
Figure D.6 — Abbreviated symbols for the EXPRESS constructed data types when used as the representation of defined data types . . . 187
Figure D.7 — Example of alternative methods for representing an enumeration data type 187
Figure D.8 — Symbols for EXPRESS extensible constructed data types . . . 188
Figure D.9 — Symbol for EXPRESS defined data type . . . 188
Figure D.10 — Symbol for an EXPRESS entity data type . . . 188
Figure D.11 — Symbol for an EXPRESS subtype_constraint . . . 188
Figure D.12 — Symbol for a schema . . . 189
Figure D.13 — Relationship line styles . . . 189
Figure D.14 — Partial entity level diagram illustrating relationship directions from the example in J.2 on page 243 (Page 1 of 1) . . . 190
Figure D.15 — Composition symbols: page references . . . 190
Figure D.16 — Composition symbols: inter-schema references . . . 191
Figure D.17 — Complete entity level diagram of the example in J.2 on page 243 (Page 1 of 1) . . . 192
Figure D.18 — Extensible select data type diagram . . . 193
Figure D.19 — Symbol for denoting an ABSTRACT SUPERTYPE if the abstract constraint is defined within a SUBTYPE_CONSTRAINT . . . 195
Figure D.20 — Symbol denoting an ABSTRACT ENTITY . . . 195

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

Figure D.21 — Example of the TOTAL_OVER coverage constraint 196
Figure D.22 — Complete entity level diagram of the inheritance graph from the example in J.3 on page 244 (Page 1 of 1) 196
Figure D.23 — Complete entity level diagram of the example in J.4 on page 245 showing attribute redeclarations in subtypes (Page 1 of 1) 197
Figure D.24 — Complete entity level diagram of the `top` schema of example 1 on page 245 illustrating inter-schema references (Page 1 of 1) 197
Figure D.25 — Complete schema level diagram of example 1 on page 245 (Page 1 of 1) . 198
Figure D.26 — Complete schema level diagram of example 2 on page 246 (Page 1 of 1) . 199

Tables

Table 1 — EXPRESS keywords 14
Table 2 — EXPRESS reserved words which are operators 14
Table 3 — EXPRESS reserved words which are constants 15
Table 4 — EXPRESS reserved words which are function names 15
Table 5 — EXPRESS reserved words which are procedure names 15
Table 6 — EXPRESS symbols 16
Table 7 — The use of data types 36
Table 8 — Supertype expression operator precedence 59
Table 9 — Scope and identifier defining items 79
Table 10 — Operator precedence 93
Table 11 — Pattern matching characters 103
Table 12 — NOT operator 105
Table 13 — AND operator 105
Table 14 — OR operator 105
Table 15 — XOR operator 106
Table 16 — Intersection operator – operand and result types 108
Table 17 — Union operator – operand and result types 110
Table 18 — Difference operator – operand and result types 110
Table 19 — Subset and superset operators - operand types 111
Table 20 — Example symbolic formatting effects 137
Table 21 — Picture formatting characters 137
Table 22 — Example picture formatting effects 137

 ©ISO 2004 — All rights reserved

AVA Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

Foreword

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

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

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

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

ISO 10303–11 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data*.

This second edition of ISO 10303–11 constitutes a minor revision of the first edition (ISO 10303-11:1994), which is provisionally retained in order to support continued use and maintenance of implementations based on the first edition and to satisfy the normative references of other parts of ISO 10303. This second edition also incorporates the Technical Corrigendum ISO 10303–11:1994/Cor.1:1999(E).

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

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

A complete list of parts of ISO 10303 is available from the Internet:
`<http://www.tc184-sc4.org/titles/>`

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

0 Introduction

0.1 General

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 the elements of the EXPRESS language. Each element of the language is presented in its own context with examples. Simple elements are introduced first, then more complex ideas are presented in an incremental manner.

The changes that lead to this edition were driven by requirements from multi-schema specifications. The new concepts constitute an architecture for extensible data models. The following keywords have been added to this edition:

— BASED_ON;

— END_SUBTYPE_CONSTRAINT;

— EXTENSIBLE;

— GENERIC_ENTITY;

— RENAMED;

— SUBTYPE_CONSTRAINT;

— TOTAL_OVER;

— WITH.

Schemas that contain these words as EXPRESS identifiers become invalid under this edition. Else, the modifications that are incorporated in this edition are upwardly compatible with the previous edition.

0.2 Language overview

EXPRESS is the name of a formal information requirements specification language. It is used to specify the information requirements of other parts of ISO 10303. It is based on a number of design goals among which are:

— the size and complexity of ISO 10303 demands that the language be parsable by both computers and humans. Expressing the information elements of ISO 10303 in a less formal manner would eliminate the possibility of employing computer automation in checking for inconsistencies in presentation or for creating any number of secondary views, including implementation views;

— the language is designed to enable partitioning of the diverse material addressed by ISO 10303. The schema is the basis for partitioning and intercommunication;

 ©ISO 2004 — All rights reserved

AVA Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

— the language focuses on the definition of entities, which represent objects of interest. The definition of an entity is in terms of its properties, which are characterized by specification of a domain and the constraints on that domain;

— the language seeks to avoid, as far as possible, specific implementation views. However, it is possible to manufacture implementation views (such as static file exchange) in an automatic and straightforward manner.

In EXPRESS, entities are defined in terms of attributes: the traits or characteristics considered important for use and understanding. These attributes have a representation which might be a simple data type (such as integer) or another entity type. A geometric point might be defined in terms of three real numbers. Names are given to the attributes which contribute to the definition of an entity. Thus, for a geometric point the three real numbers might be named x, y and z. A relationship is established between the entity being defined and the attributes that define it, and, in a similar manner, between the attribute and its representation.

NOTE 1 A number of languages have contributed to EXPRESS, in particular, Ada, Algol, C, C^{++}, Euler, Modula-2, Pascal, PL/I and SQL. Some facilities have been invented to make EXPRESS more suitable for the job of expressing an information model.

NOTE 2 The examples of EXPRESS usage in this manual do not conform to any particular style rules. Indeed, the examples sometimes use poor style to conserve space or to show flexibility. The examples are not intended to reflect the content of the information models defined in other parts of ISO 10303. They are crafted to show particular features of EXPRESS. Any similarity between the examples and the normative information models specified in other parts of ISO 10303 should be ignored.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

Documents
Document Sharing Hub

Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual

1 Scope

This part of ISO 10303 specifies a language by which aspects of product data can be defined. The language is called EXPRESS.

This part of ISO 10303 also specifies a graphical representation for a subset of the constructs in the EXPRESS language. This graphical representation is called EXPRESS-G.

EXPRESS is a data specification language as defined in ISO 10303-1. It consists of language elements that allow an unambiguous data definition and specification of constraints on the data defined.

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

— data types;

— constraints on instances of the data types.

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

— definition of database formats;

— definition of file formats;

— definition of transfer formats;

— process control;

— information processing;

— exception handling.

EXPRESS is not a programming language.

©ISO 2004 — All rights reserved

Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

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

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

ISO/IEC 10646:2003, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*

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.

— Conformance requirement;

— Context;

— Data;

— Data specification language;

— Information;

— Information model;

— PICS proforma;

3.2 Terms defined in ISO/IEC 10646

For the purposes of this part of ISO 10303, the following term defined in ISO/IEC 10646 applies.

— Graphic character.

NOTE This definition includes only those characters in ISO/IEC 10646 that have a defined visual representation; this explicitly excludes any cells that are empty or crosshatched.

3.3 Other terms and definitions

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

 ©ISO 2004 — All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

3.3.1
complex entity data type
a representation of an entity. A complex entity data type establishes a domain of values defined by the common attributes and constraints of an allowed combination of entity data types within a particular subtype/supertype graph.

3.3.2
complex entity (data type) instance
a named complex entity data type value. The name of a complex entity instance is used for referencing the instance.

3.3.3
complex entity (data type) value
a unit of data that represents a unit of information within the class defined by a complex entity data type. It is a member of the domain established by this complex entity data type.

3.3.4
constant
a named unit of data from a specified domain. The value cannot be modified.

3.3.5
data type
a domain of values.

3.3.6
entity
a class of information defined by common properties.

3.3.7
entity data type
a representation of an entity. An entity data type establishes a domain of values defined by common attributes and constraints.

3.3.8
entity (data type) instance
a named entity data type value. The name of an entity instance is used for referencing the instance.

3.3.9
(single) entity (data type) value
a unit of data which represents a unit of information within the class defined by an entity data type. It is a member of the domain established by this entity data type.

3.3.10
instance
a named value.

3.3.11
multi-leaf complex entity (data type)
a complex entity data type that consists of more than one entity data types that do not have further subtypes within this complex entity data type.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

3.3.12
multi-leaf complex entity (data type) instance
a named multi-leaf complex entity data type value. The name of a multi-leaf complex entity instance is used for referencing the instance.

3.3.13
multi-leaf complex entity (data type) value
a unit of data that represents a unit of information within the class defined by a multi-leaf complex entity data type. It is a member of the domain established by this multi-leaf complex entity data type.

3.3.14
partial complex entity data type
a potential representation of an entity. A partial complex entity data type is a grouping of entity data types within a subtype/supertype graph which may form part or all of a complex entity data type.

3.3.15
partial complex entity value
a value of a partial complex entity data type. This has no meaning on its own and must be combined with other partial complex entity values and a name to form a complex entity instance.

3.3.16
population
a collection of entity data type instances.

3.3.17
primary schema
a schema in a group of interrelated schemas that form a, possibly cyclic, directed graph. A primary schema is a schema of interest. There can be one or more primary schemas within the graph, where the other schemas in the graph are only there to support the primary schema. The primary schema has a special role in the conversion of a shortform schema into a longform schema (see G).

3.3.18
root schema
a schema in a group of interrelated schemas that form a, possibly cyclic, directed graph. The root schema is not the target of any interface specification, but all the other schemas can be reached from the root schema. The root schema can be considered to be representative of the graph. The root schema has a special role in the conversion of a shortform schema into a longform schema (see G).

3.3.19
simple entity (data type) instance
a named unit of data which represents a unit of information within the class defined by an entity. It is a member of the domain established by a single entity data type.

3.3.20
subtype/supertype graph
a declared collection of entity data types. The entity data types declared within a subtype/supertype graph are related via the subtype statement. A subtype/supertype graph defines one or more complex entity data types.

 ©ISO 2004 — All rights reserved

AVA Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

3.3.21
token
a non-decomposable lexical element of a language.

3.3.22
value
a unit of data.

4 Conformance requirements

4.1 Formal specifications written in EXPRESS

4.1.1 Lexical language

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

Levels of checking

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 items are consistent with the scope and visibility rules defined in clauses 10 and 11.

Level 2: Type checking. This level consists of checking the formal specification to ensure that it is consistent with the following:

— expressions shall comply with the rules specified in clause 12;

— assignments shall comply with the rules specified in 13.3;

— inverse attribute declarations shall comply with the rules specified in 9.2.1.3;

— attribute redeclarations shall comply with the rules specified in 9.2.3.4.

Level 3: Value checking. This level consists of checking the formal specification to ensure that it complies with statements of the form 'A shall be greater than B' as specified in clause 7 to clause 16. This is limited to those places where both A and B can be evaluated from literals and/or constants.

Level 4: Complete checking. This level consists of checking a formal specification to ensure that it complies with all statements of requirement as specified in this part of ISO 10303.

EXAMPLE This part of ISO 10303 states that functions shall specify a return statement in each of the possible paths a process may take when that function is invoked. This would have to be checked.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

4.1.2 Graphical form

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

Levels of checking

Level 1: Symbols and scope checking. This level consists of checking the formal specification to ensure that it is consistent with either an entity level or a schema level specification as defined in D.5 and D.6 respectively. This includes checking that the formal specification uses symbols as defined in D.2, D.3 and D.4. The formal specification will also be checked to ensure that page references and redeclared attributes comply with D.4.1 and D.5.5 respectively.

Level 2: Complete checking. This level consists of checking a formal specification to identify those places which do not conform to either a complete entity level or complete schema level specification as defined in annex D and the requirements defined in clause 7 through clause 16.

4.2 Implementations of EXPRESS

4.2.1 EXPRESS language parser

An implementation of an EXPRESS language parser shall be able to parse any formal specification written in EXPRESS, consistent with the constraints as specified in the annex E associated with that implementation. An EXPRESS language parser shall be said to conform to a particular checking level (as defined in 4.1.1) if it can apply all checks required by the level (and any level below that) to a formal specification written in EXPRESS.

The implementor of an EXPRESS language parser shall state any constraints which 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 in the form specified by annex E for the purposes of conformance testing.

4.2.2 Graphical editing tool

An implementation of an EXPRESS-G editing tool shall be able to create and display a formal specification in EXPRESS-G, consistent with the constraints as specified in the annex E associated with that implementation. An EXPRESS-G editing tool shall be said to conform to a particular checking level if it can create and display a formal specification in EXPRESS-G which is consistent with the specified level of checking (and any level below that).

The implementor of an EXPRESS-G editing tool shall state any constraints which the implementation imposes on the number and length of identifiers, the number of symbols available per page of the model, and the maximum number of pages. Such constraints shall be documented in the form specified by annex E for the purposes of conformance testing.

©ISO 2004 — All rights reserved
AVA Documents Document Sharing Hub
Full document details: BS ISO 10303-11:2004

5 Fundamental principles

The reader of this document is assumed to be familiar with the following concepts.

A schema written in the EXPRESS language describes a set of conditions which establishes a domain. Instances can be evaluated to determine if they are in the domain. If the instances meet all the conditions, then they are asserted to be in the domain. If the instances fail to meet any of the conditions, then the instances have violated the conditions and thus are not in the domain. In the case where the instances do not contain values for optional attributes and some of the conditions involve those optional attributes, it may not be possible to determine whether the instances meet all the conditions. In this case, the instances are considered in the domain.

Many of the elements of the EXPRESS language are assigned names. The name allows other language elements to reference the associated representation. The use of the name in the definition of other language elements constitutes a reference to the underlying representation. While the syntax of the language uses an identifier for the name, the underlying representation must be examined to understand the structure.

The specification of an entity data type in the EXPRESS language describes a domain. The individual members of the domain are assumed to be distinguished by some associated identifier which is unique. EXPRESS does not specify the content or representation of these identifiers.

The declaration of a constant entity instance defines an identifiable member of the domain described by the entity data type. These entity instances shall not be modified or deleted by operations performed on the domain.

The procedural description of constraints in EXPRESS may declare or make reference to additional entity instances, as local variables, which are assumed to be transient identifiable members of the domain. These procedural descriptions may modify these additional entity instances, but cannot modify persistent members of the domain. These transient members of the domain are only accessible within the scope of the procedural code in which they were declared, and cease to exist upon termination of that code. Transient members may violate uniqueness constraints, global rules, and where rules. This standard does not define the behaviour of functions or procedures when instances that violate these constraints are passed to them as actual parameters.

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

— how references to names are resolved;

— how other schemas are known;

— how or when constraints are checked;

— what an implementation shall do if a constraint is not met;

— whether or not instances that do not conform to an EXPRESS schema are allowed to exist in an implementation;

— whether, when or how instances are created, modified or deleted in an implementation.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

6 Language specification syntax

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

The full syntax for the EXPRESS language is given in annex A. 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.

6.1 The syntax of the specification

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

NOTE See annex L.1 under 3 for a reference.

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 which 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 which cannot be expanded further. A literal is a case independent sequence of characters enclosed in apostrophes. Character, in this case, stands for any character as defined by ISO/IEC 10646 cells 21-7E in group 00, plane 00, row 00. For an apostrophe to appear in a literal it must be written twice.

— 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;

 ©ISO 2004 — All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

— vertical bar '|' indicates that exactly one of the terms in the expression shall be chosen.

EXAMPLE 1 The syntax for a string type is as follows:

Syntax:

```
311 string_type = STRING [ width_spec ] .
341 width_spec = '(' width ')' [ FIXED ] .
340 width = numeric_expression .
```

The complete syntax definition (annex A) contains the definitions for `STRING`, `numeric_expression` and `FIXED`.

EXAMPLE 2 Following the syntax given in example 1, the following alternatives are possible:

a) `string`

b) `string ( 22 )`

c) `string ( 19 ) fixed`

The rule for `numeric_expression` is quite complex and allows many other things to be written.

6.2 Special character notation

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

— `\a` represents characters in cells 21-7E of row 00, plane 00, group 00 of ISO/IEC 10646;

— `\n` represents a newline (system dependent) (see 7.1.5.2);

— `\q` is the quote (apostrophe) (') character and is contained within `\a`;

— `\s` is the space character;

— `\x9`, `\xA`, and `\xD` represent the characters in positions 9, 10, and 13 respectively of 00, plane 00, group 00 of ISO/IEC 10646.

7 Basic language elements

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

The basic language elements are composed into a stream of text, typically broken into physical lines. A physical line is any number (including zero) of characters ended by a newline (see 7.1.5.2).

NOTE A schema is easier to read when statements are broken into lines and whitespace is used to layout the different constructs.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

EXAMPLE The following are equivalent

```
entity point;x,y,z:real;end_entity;

ENTITY point;
  x,
  y,
  z : REAL;
END_ENTITY;
```

7.1 Character set

A schema written in EXPRESS shall use only the characters in the following character set: characters allocated to cells 09, 0A, 0D, the graphic characters lying in the range 20 to 7E of ISO/IEC 10646, and the special character `\n` signifying the newline. This set of characters is called the EXPRESS character set. A member of this set is referred to by the cell of the standard in which this character is allocated; these cell numbers are specified in hexadecimal. The printable characters from this set (cells 21–7E of ISO/IEC 10646) are combined to form the tokens for the EXPRESS language. The EXPRESS tokens are keywords, identifiers, symbols or literals. The EXPRESS character set is further classified below:

The character set thus specified is an abstract character set; it is independent of its representation in an implementation.

NOTE 1 ISO/IEC 6429 specifies semantics for the characters in positions 09, 0A and 0D in ISO/IEC 10646. This part of ISO 10303 does not require the semantics prescribed in ISO/IEC 6429; neither does it preclude them.

NOTE 2 This clause only refers to the characters used to specify an EXPRESS schema, and does not specify the domain of characters allowed within a string data type.

7.1.1 Digits

EXPRESS uses the Arabic digits 0-9 (cells 30 - 39 of the EXPRESS character set).

Syntax:

```
124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
```

7.1.2 Letters

EXPRESS uses the upper and lower case letters of the English alphabet (cells 41 - 5A and 61 - 7A of the EXPRESS character set). The case of letters is significant only within explicit string literals.

NOTE EXPRESS may be written using upper, lower or mixed case letters (see example 7).

Syntax:

```
128 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' .
```

 ©ISO 2004 — All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

7.1.3 Special characters

The special characters (printable characters which are neither letters nor digits) are used mainly for punctuation and as operators. The special characters are in cells 21-2F, 3A-3F, 40, 5B-5E, 60 and 7B-7E of the EXPRESS character set.

Syntax:

```
137 special = not_paren_star_quote_special | '(' | ')' | '*' | '''' .
132 not_paren_star_quote_special = '!' | '"' | '#' | '$' | '%' | '&' | '+' | ',' |
                                   '-' | '.' | '/' | ':' | ';' | '<' | '=' | '>' |
                                   '?' | '@' | '[' | '\' | ']' | '^' | '_' | '`' |
                                   '{' | '|' | '}' | '~' .
```

7.1.4 Underscore

The underscore character (_, cell 5F of the EXPRESS character set) may be used in identifiers and keywords, with the exception that the underscore character shall not be used as the first character.

7.1.5 Whitespace

Whitespace is defined by the following sub-clauses and by 7.1.6. Whitespace shall be used to separate the tokens of a schema written in EXPRESS.

NOTE Liberal, and consistent, use of whitespace can improve the structure and readability of a schema.

7.1.5.1 Space character

One or more spaces (cell 20 of the EXPRESS character set) can appear between two tokens. The notation `\s` may used to represent a blank space character in the syntax of the language.

7.1.5.2 Newline

A newline marks the physical end of a line within a formal specification written in EXPRESS. Newline is normally treated as a space but is significant when it terminates a tail remark or abnormally terminates a string literal. A newline is represented by the notation `\n` in the syntax of the language.

The representation of a newline is implementation specific.

7.1.5.3 Other characters

The characters in cells 09, 0A and 0D shall be treated as whitespace, unless within a string literal. The notation `\xn` where `n` is one of the characters 9, A, and D shall be used to represent these characters in the syntax of the language.

7.1.6 Remarks

A remark is used for documentation and shall be interpreted by an EXPRESS language parser as whitespace. There are two forms of remark, embedded remark and tail remark. Both forms of remark may be associated with an identified construct using a remark tag.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

7.1.6.1 Embedded remark

The character pair (* denotes the start of an embedded remark and the character pair *) denotes its end. An embedded remark may appear between any two tokens.

Syntax:

```
145 embedded_remark = '(*' [ remark_tag ] { ( not_paren_star { not_paren_star } ) |
                      lparen_then_not_lparen_star | ( '*' { '*' } ) |
                      not_rparen_star_then_rparen | embedded_remark } '*)' .
147 remark_tag = '"' remark_ref { '.' remark_ref } '"' .
148 remark_ref = attribute_ref | constant_ref | entity_ref | enumeration_ref |
                 function_ref | parameter_ref | procedure_ref | rule_label_ref |
                 rule_ref | schema_ref | subtype_constraint_ref | type_label_ref |
                 type_ref | variable_ref .
131 not_paren_star = letter | digit | not_paren_star_special .
128 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' .
124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
133 not_paren_star_special = not_paren_star_quote_special | '''' .
132 not_paren_star_quote_special = '!' | '"' | '#' | '$' | '%' | '&' | '+' | ',' |
                                   '-' | '.' | '/' | ':' | ';' | '<' | '=' | '>' |
                                   '?' | '@' | '[' | '\' | ']' | '^' | '_' | '`' |
                                   '{' | '|' | '}' | '~' .
129 lparen_then_not_lparen_star = '(' { '(' } not_lparen_star { not_lparen_star } .
130 not_lparen_star = not_paren_star | ')' .
138 not_rparen_star_then_rparen = not_rparen_star { not_rparen_star } ')' { ')' } .
135 not_rparen_star = not_paren_star | '(' .
```

Any character within the EXPRESS character set may occur between the start and end of an embedded remark including the newline character; therefore, embedded remarks can span several physical lines.

Embedded remarks may be nested.

NOTE Care must be taken when nesting remarks to ensure that there are matched pairs of symbols.

EXAMPLE The following is an example of nested embedded remarks.

```
(* The '(*' symbol starts a remark, and the '*)' symbol ends it *)
```

7.1.6.2 Tail remark

The tail remark is written at the end of a physical line. Two consecutive hyphens (--) start the tail remark and the following newline terminates it.

Syntax:

```
149 tail_remark = '--' [ remark_tag ] { \a | \s | \x9 | \xA | \xD } \n .
147 remark_tag = '"' remark_ref { '.' remark_ref } '"' .
148 remark_ref = attribute_ref | constant_ref | entity_ref | enumeration_ref |
                 function_ref | parameter_ref | procedure_ref | rule_label_ref |
                 rule_ref | schema_ref | subtype_constraint_ref | type_label_ref |
                 type_ref | variable_ref .
```

 ©ISO 2004 — All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

EXAMPLE `-- this is a remark that ends with a newline`

7.1.6.3 Remark tag

A remark may be related to a named item, that is, an item which is associated with an identifier, by placing a remark tag as the first sequence of characters within the remark. The remark tag shall immediately follow the pair of characters that is used to identify the remark. The remark tag itself consists of a reference to the identifier declared in the item that is enclosed by quotation mark signs.

Syntax:

```
147 remark_tag = '"' remark_ref { '.' remark_ref } '"' .
148 remark_ref = attribute_ref | constant_ref | entity_ref | enumeration_ref |
                 function_ref | parameter_ref | procedure_ref | rule_label_ref |
                 rule_ref | schema_ref | subtype_constraint_ref | type_label_ref |
                 type_ref | variable_ref .
```

Rules and restrictions:

a) The `remark_ref` shall follow the visibility rules defined in 10.2.

b) A qualified remark reference shall use the visibility rules defined in 10.2 in the following manner: the reference to the left of the '.' shall identify the scope in which to find the reference to the right.

NOTE A qualified remark reference is a remark reference that uses the '.'-notation (see syntax rule 147 above).

c) If a remark reference is not found according to the visibility rules identified above, the remark is not to be associated with any item.

d) A tagged remark that contains other tagged remarks (via nesting) shall be associated completely (including nested remarks) with the item referenced. The inner tagged remarks shall be associated with their identified items also.

e) If both the nested remark and the enclosing remark refer to the same identified item, the nested remark shall be associated with that item twice; once inside the enclosing remark, and once directly.

EXAMPLE 1 The tagged remark in the example below refers to the attribute `attr` in the scope of `ent`.

```
ENTITY ent;
  attr: INTEGER;
END_ENTITY;
(*"ent.attr" The attr attribute ... *)
```

EXAMPLE 2 The reference to a SCHEMA `my_second_schema` may in a tagged remark be followed by any identifier that is declared directly within the scope of that SCHEMA, for example, by the name of the FUNCTION `a_complicated_function` in the example below.

```
SCHEMA my_second_schema;
...
FUNCTION a_complicated_function;
```

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

```
...
END_FUNCTION;
(*"my_second_schema.a_complicated_function" This complicated function ... *)
...
END_SCHEMA;
```

7.2 Reserved words

The reserved words of EXPRESS are the keywords and the names of built-in constants, functions and procedures. The reserved words shall not be used as identifiers. The reserved words of EXPRESS are described below.

7.2.1 Keywords

The EXPRESS keywords are shown in Table 1.

NOTE Keywords have an uppercase production which represents the literal. This is to enable easier reading of the syntax productions.

Table 1 – EXPRESS keywords

ABSTRACT	AGGREGATE	ALIAS	ARRAY
AS	BAG	BASED_ON	BEGIN
BINARY	BOOLEAN	BY	CASE
CONSTANT	DERIVE	ELSE	END
END_ALIAS	END_CASE	END_CONSTANT	END_ENTITY
END_FUNCTION	END_IF	END_LOCAL	END_PROCEDURE
END_REPEAT	END_RULE	END_SCHEMA	END_SUBTYPE_CONSTRAINT
END_TYPE	ENTITY	ENUMERATION	ESCAPE
EXTENSIBLE	FIXED	FOR	FROM
FUNCTION	GENERIC	GENERIC_ENTITY	IF
INTEGER	INVERSE	LIST	LOCAL
LOGICAL	NUMBER	OF	ONEOF
OPTIONAL	OTHERWISE	PROCEDURE	QUERY
REAL	RENAMED	REFERENCE	REPEAT
RETURN	RULE	SCHEMA	SELECT
SET	SKIP	STRING	SUBTYPE
SUBTYPE_CONSTRAINT	SUPERTYPE	THEN	TO
TOTAL_OVER	TYPE	UNIQUE	UNTIL
USE	VAR	WHERE	WHILE
WITH			

7.2.2 Reserved words which are operators

The operators defined by reserved words are shown in Table 2. See clause 12 for the definition of these operators.

Table 2 – EXPRESS reserved words which are operators

AND	ANDOR	DIV	IN
LIKE	MOD	NOT	OR
XOR			

 ©ISO 2004 – All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

7.2.3 Built-in constants

The names of the built-in constants are shown in Table 3. See clause 14 for the definitions of these constants.

Table 3 – EXPRESS reserved words which are constants

?	SELF	CONST_E	PI
FALSE	TRUE	UNKNOWN	

7.2.4 Built-in functions

The names of the built-in functions are shown in Table 4. See clause 15 for the definitions of these functions.

Table 4 – EXPRESS reserved words which are function names

ABS	ACOS	ASIN	ATAN
BLENGTH	COS	EXISTS	EXP
FORMAT	HIBOUND	HIINDEX	LENGTH
LOBOUND	LOG	LOG2	LOG10
LOINDEX	NVL	ODD	ROLESOF
SIN	SIZEOF	SQRT	TAN
TYPEOF	USEDIN	VALUE	VALUE_IN
VALUE_UNIQUE			

7.2.5 Built-in procedures

The names of the built-in procedures are shown in Table 5. See clause 16 for the definitions of these procedures.

Table 5 – EXPRESS reserved words which are procedure names

INSERT	REMOVE

7.3 Symbols

Symbols are special characters or groups of special characters which have special meaning in EXPRESS. Symbols are used in EXPRESS as delimiters and operators. A delimiter is used to begin, separate or terminate adjacent lexical or syntactic elements. Interpretation of these elements would be impossible without separators. Operators denote that actions shall be performed on the operands which are associated with the operator; see clause 12 for an explanation of operators. The EXPRESS symbols are shown in Table 6.

7.4 Identifiers

Identifiers are names given to the items declared in a schema (see 9.3), including the schema itself. An identifier shall not be the same as an EXPRESS reserved word.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

Table 6 – EXPRESS symbols

.	,	;	:
*	+	-	=
%	'	\	/
<	>	[	]
{	}	|	e
(	)	<=	<>
>=	<*	:=	||
**	--	(*	*)
:=:	:<>:		

Syntax:

```
143 simple_id = letter { letter | digit | '_' } .
128 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' .
124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
```

The first character of an identifier shall be a letter. The remaining characters, if any, may be any combination of letters, digits and the underscore character.

The implementor of an EXPRESS language parser shall specify the maximum number of characters of an identifier which can be read by that implementation, using annex E.

7.5 Literals

A literal is a self-defining constant value. The type of a literal depends on how characters are composed to form a token. The literal types are binary, integer, real, string and logical.

Syntax:

```
251 literal = binary_literal | logical_literal | real_literal | string_literal .
```

7.5.1 Binary literal

A binary literal represents a value of a binary data type and is composed of the % symbol followed by one or more `bits` (0 or 1).

Syntax:

```
139 binary_literal = '%' bit { bit } .
123 bit = '0' | '1' .
```

The implementor of an EXPRESS language parser shall specify the maximum number of bits in a binary literal which can be read by that implementation, using annex E.

EXAMPLE A valid binary literal

```
%0101001100
```

 ©ISO 2004 – All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

7.5.2 Integer literal

An integer literal represents a value of an integer data type and is composed of one or more `digits`.

Syntax:

```
141 integer_literal = digits .
125 digits = digit { digit } .
124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
```

NOTE The sign of the integer literal is not modelled within the syntax since EXPRESS uses the concept of unary operators within the expression syntax.

The implementor of an EXPRESS language parser shall specify the maximum integer value of an integer literal which can be read by that implementation, using annex E.

EXAMPLE Valid integer literals

```
4016
38
```

7.5.3 Real literal

A real literal represents a value of a real data type and is composed of a mantissa and an optional exponent; the mantissa shall include a decimal point.

Syntax:

```
142 real_literal = integer_literal |
                   ( digits '.' [ digits ] [ 'e' [ sign ] digits ] ) .
125 digits = digit { digit } .
124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
304 sign = '+' | '-' .
```

NOTE The sign of the real literal is not modelled within the syntax since EXPRESS uses the concept of unary operators within the expression syntax.

The implementor of an EXPRESS language parser shall specify the maximum precision and maximum exponent of a real literal which can be read by that implementation, using annex E.

EXAMPLE 1 Valid real literals

```
1.E6        "E" may be written in upper or lower case
3.5e-5
359.62
```

EXAMPLE 2 Invalid real literals

```
.001    At least one digit must precede the decimal point
1e10    A decimal point must be part of the literal
1. e10  A space is not part of the real literal
```

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

7.5.4 String literal

A string literal represents a value of a string data type. There are two forms of string literal, the simple string literal and encoded string literal. A simple string literal is composed of a sequence of characters in the EXPRESS character set (see 7.1) enclosed by apostrophes (').
An apostrophe within a simple string literal is represented by two consecutive apostrophes. An encoded string literal is a four octet encoded representation of each character in a sequence of ISO/IEC 10646 characters enclosed in quotation marks (").
The encoding is defined as follows:

a) first octet = ISO/IEC 10646 group in which the character is defined;

b) second octet = ISO/IEC 10646 plane in which the character is defined;

c) third octet = ISO/IEC 10646 row in which the character is defined;

d) fourth octet = ISO/IEC 10646 cell in which the character is defined.

The sequence of octets shall identify one of the valid characters of ISO/IEC 10646.

A string literal shall never span a physical line boundary; that is, a newline shall not occur between the apostrophes enclosing a string literal.

Syntax:

```
310 string_literal = simple_string_literal | encoded_string_literal .
144 simple_string_literal = \q { ( \q \q ) | not_quote | \s | \x9 | \xA | \xD } \q .
134 not_quote = not_paren_star_quote_special | letter | digit | '(' | ')' | '*' .
132 not_paren_star_quote_special = '!' | '"' | '#' | '$' | '%' | '&' | '+' | ',' |
                                   '-' | '.' | '/' | ':' | ';' | '<' | '=' | '>' |
                                   '?' | '@' | '[' | '\' | ']' | '^' | '_' | '`' |
                                   '{' | '|' | '}' | '~' .
128 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' .
124 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
140 encoded_string_literal = '"' encoded_character { encoded_character } '"' .
126 encoded_character = octet octet octet octet .
136 octet = hex_digit hex_digit .
127 hex_digit = digit | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' .
```

The implementor of an EXPRESS language parser shall specify the maximum number of characters of a simple string literal which can be read by that implementation, using annex E.

The implementor of an EXPRESS language parser shall also specify the maximum number of octets (must be a multiple of four) of an encoded string literal which can be read by that implementation, using annex E.

EXAMPLE 1 Valid simple string literals

`'Baby needs a new pair of shoes!'`
Reads . . . Baby needs a new pair of shoes!

`'Ed''s Computer Store'`
Reads . . . Ed's Computer Store

 ©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

EXAMPLE 2 Invalid simple string literals

`'Ed's Computer Store'`
There must always be an even number of apostrophes.

`'Ed''s Computer`
`Store'`
Spans a physical line

EXAMPLE 3 Valid encoded string literals

`"00000041"`
Reads A

`"000000C5"`
Reads Å

`"0000795E00006238"`
These are the Japanese ideographs 神戸 for Kobe.

EXAMPLE 4 Invalid encoded string literals

`"000041"`
Octets must be supplied in groups of four

`"00000041 000000C5"`
Only hexadecimal characters are allowed between `""`

7.5.5 Logical literal

A logical literal represents a value of a logical or boolean data type and is one of the built-in constants TRUE, FALSE, or UNKNOWN.

NOTE UNKNOWN is not compatible with a boolean data type.

Syntax:

```
255 logical_literal = FALSE | TRUE | UNKNOWN .
```

8 Data types

This clause defines the data types provided as part of the language. Every attribute, local variable or formal parameter has an associated data type.

Data types are classified as simple data types, aggregation data types, named data types, constructed data types, and generalized data types. Data types are also classified according to their usage as base data types, parameter data types, and underlying data types. The relationships between these two classifications are described in 8.6.

The operations that may be performed on values of these data types are defined in clause 12.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

8.1 Simple data types

The simple data types define the domains of the atomic data units in EXPRESS. That is, they cannot be further subdivided into elements that EXPRESS recognizes. The simple data types are NUMBER, REAL, INTEGER, STRING, BOOLEAN, LOGICAL, and BINARY.

8.1.1 Number data type

The NUMBER data type has as its domain all numeric values in the language. The NUMBER data type shall be used when a more specific numeric representation is not important.

Syntax:

```
261 number_type = NUMBER .
```

EXAMPLE Since we may not know the context of `size`, we do not know how to correctly represent it. The size of the crowd at a football game, for example, would be an INTEGER, whereas the area of the pitch would be a REAL.

```
size : NUMBER ;
```

NOTE In future editions of this standard there may be further specializations of the NUMBER data type, for example, complex numbers.

8.1.2 Real data type

The REAL data type has as its domain all rational, irrational and scientific real numbers. It is a specialization of the NUMBER data type.

Syntax:

```
278 real_type = REAL [ '(' precision_spec ')' ] .
268 precision_spec = numeric_expression .
```

Rational and irrational numbers have infinite resolution and are exact. Scientific numbers represent quantities which are known only to a specified precision. The `precision_spec` is stated in terms of significant digits.

A real number literal is represented by a mantissa and optional exponent. The number of digits making up the mantissa when all leading zeros have been removed is the number of significant digits. The known precision of a value is the number of leading digits that are necessary to the application.

Rules and restrictions:

a) The `precision_spec` gives the minimum number of digits of resolution that are required. This expression shall evaluate to a positive integer value.

b) When no resolution specification is given the precision of the real number is unconstrained.

 ©ISO 2004 — All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

8.1.3 Integer data type

The INTEGER data type has as its domain all integer numbers. It is a specialization of the REAL data type.

Syntax:

```
241 integer_type = INTEGER .
```

EXAMPLE This example uses an INTEGER data type to represent an attribute named `nodes`. The domain of this attribute is all integers, with no further constraint.

```
ENTITY foo;
  nodes : INTEGER;
  ...
END_ENTITY;
```

8.1.4 Logical data type

The LOGICAL data type has as its domain the three literals TRUE, FALSE, and UNKNOWN.

Syntax:

```
256 logical_type = LOGICAL .
```

The following ordering holds for the values of the LOGICAL data type: FALSE < UNKNOWN < TRUE. The LOGICAL data type is compatible with the BOOLEAN data type, except that the value UNKNOWN cannot be assigned to a boolean variable.

8.1.5 Boolean data type

The BOOLEAN data type has as its domain the two literals TRUE and FALSE. The BOOLEAN data type is a specialization of the LOGICAL data type.

Syntax:

```
182 boolean_type = BOOLEAN .
```

The same ordering holds for values of the BOOLEAN data type as for values of the LOGICAL data type, that is: FALSE < TRUE.

EXAMPLE In this example, an attribute named `planar` is represented by the BOOLEAN data type. The value for `planar` associated with an instance of `surface` can be either TRUE or FALSE.

```
ENTITY surface;
  planar : BOOLEAN;
  ...
END_ENTITY;
```

8.1.6 String data type

The STRING data type has as its domain sequences of characters. The characters that are permitted as part of a string value are those characters allocated to cells 09, 0A, 0D and the

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

graphic characters lying in the ranges 20 to 7E and A0 to 10FFFE of ISO/IEC 10646.

Syntax:

```
311 string_type = STRING [ width_spec ] .
341 width_spec = '(' width ')' [ FIXED ] .
340 width = numeric_expression .
```

A STRING data type may be defined as either fixed or varying width (number of characters). If it is not specifically defined as fixed width (by using the FIXED reserved word in the definition) the string has varying width.

The domain of a fixed width STRING data type is the set of all character sequences of exactly the width specified in the type definition.

The domain of a varying width STRING data type is the set of all character sequences of width less than or equal to the maximum width specified in the type definition.

If no width is specified, the domain is the set of all character sequences, with no constraint on the width of these sequences.

Substrings and individual characters may be addressed using subscripts as described in 12.5.

The case (upper or lower) of letters within a string is significant.

Rules and restrictions:

The `width` expression shall evaluate to a positive integer value.

EXAMPLE 1 The following defines a varying length string; values of which have no defined maximum length.

```
string1 : STRING;
```

EXAMPLE 2 The following defines a string that is a maximum of ten characters in length; values of which may vary in actual length from zero to ten characters.

```
string2 : STRING(10);
```

EXAMPLE 3 The following defines a string that is exactly ten characters in length; values of which must contain ten characters.

```
string3 : STRING(10) FIXED;
```

8.1.7 Binary data type

The BINARY data type has as its domain sequences of bits, each bit being represented by 0 or 1.

Syntax:

```
181 binary_type = BINARY [ width_spec ] .
341 width_spec = '(' width ')' [ FIXED ] .
340 width = numeric_expression .
```

 ©ISO 2004 — All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

A BINARY data type may be defined as either fixed or varying width (number of bits). If it is not specifically defined as fixed width (by using the FIXED reserved word in the definition) the binary data type has varying width.

The domain of a fixed width BINARY data type is the set of all bit sequences of exactly the width specified in the type definition.

The domain of a varying width BINARY data type is the set of all bit sequences of width less than or equal to the maximum width specified in the type definition. If no width is specified, the domain is the set of all bit sequences, with no constraint on the width of these sequences.

Subbinaries and individual bits may be addressed using subscripts as described in 12.3.

Rules and restrictions:

The `width` expression shall evaluate to a positive integer value.

EXAMPLE The following might be used to hold character font information.

```
ENTITY character;
  representation : ARRAY [1:20] OF BINARY (8) FIXED ;
END_ENTITY;
```

8.2 Aggregation data types

Aggregation data types have as their domains collections of values of a given base data type (see 8.6.1). These base data type values are called elements of the aggregation collection. EXPRESS provides for the definition of four kinds of aggregation data types: ARRAY, LIST, BAG, and SET. Each kind of aggregation data type attaches different properties to its values. The AGGREGATE data type is the generalization of these four aggregation data types (see 9.5.3.1).

— An ARRAY is a fixed-size ordered collection. It is indexed by a sequence of integers.

  EXAMPLE 1 A transformation matrix (for geometry) may be defined as an array of arrays (of numbers).

— A LIST is a sequence of elements which can be accessed according to their position. The number of elements in a list may vary, and can be constrained by the definition of the data type.

  EXAMPLE 2 The operations of a process plan might be represented as a list. The operations are ordered, and operations can be added to or removed from a process plan.

— A BAG is an unordered collection in which duplication is allowed. The number of elements in a bag may vary, and can be constrained by the definition of the data type.

  EXAMPLE 3 The collection of fasteners used in an assembly problem could be represented as a bag. There might be a number of elements which are equivalent bolts, but which one is used in a particular hole is unimportant.

— A SET is an unordered collection of elements in which no two elements are instance equal. The number of elements in a set may vary, and can be constrained by the definition of the data type.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

EXAMPLE 4 The population of people in this world is a set.

NOTE EXPRESS aggregations are one dimensional. Objects usually considered to have multiple dimensions (such as mathematical matrices) can be represented by an aggregation data type whose base type is another aggregation data type. Aggregation data types can be thus nested to an arbitrary depth, allowing any number of dimensions to be represented.

EXAMPLE 5 One could define a `LIST[1:3] OF ARRAY[5:10] OF INTEGER`, which would in effect have two dimensions.

8.2.1 Array data type

An ARRAY data type has as its domain indexed, fixed-size collections of like elements. The lower and upper bounds, which are integer-valued expressions, define the range of index values, and thus the size of each array collection. An ARRAY data type definition may optionally specify that an array value cannot contain duplicate elements. It may also specify that an array value need not contain an element at every index position.

Syntax:

```
175 array_type = ARRAY bound_spec OF [ OPTIONAL ] [ UNIQUE ] instantiable_type .
185 bound_spec = '[' bound_1 ':' bound_2 ']' .
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
```

Given that m is the lower bound and n is the upper bound, there are exactly $n-m+1$ elements in the array. These elements are indexed by subscripts from m to n, inclusive (see 12.6.1).

NOTE 1 The bounds may be positive, negative or zero, but may not be indeterminate (`?`) (see 14.2).

Rules and restrictions:

a) Both expressions in the bound specification, `bound_1` and `bound_2`, shall evaluate to integer values. Neither shall evaluate to the indeterminate (`?`) value.

b) `bound_1` gives the lower bound of the array. This shall be the lowest index which is valid for an array value of this data type.

c) `bound_2` gives the upper bound of the array. This shall be the highest index which is valid for an array value of this data type.

d) `bound_1` shall be less than or equal to `bound_2`.

e) If the OPTIONAL keyword is specified, an array value of this data type may have the indeterminate (`?`) value at one or more index positions.

f) If the OPTIONAL keyword is not specified, an array value of this data type shall not contain an indeterminate (`?`) value at any index position.

g) If the UNIQUE keyword is specified, each element in an array value of this data type shall be different from (that is, not instance equal to) every other element in the same array value.

NOTE 2 Both OPTIONAL and UNIQUE may be specified in the same ARRAY data type definition. This does not preclude multiple indeterminate (`?`) values from occurring in a single array value. This is because

 ©ISO 2004 — All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

comparisons between indeterminate (?) values result in UNKNOWN, so the uniqueness constraint is not violated.

EXAMPLE This example shows how a multi-dimensioned array is declared.

```
sectors : ARRAY [ 1 : 10 ] OF       -- first dimension
          ARRAY [ 11 : 14 ] OF      -- second dimension
          UNIQUE something;
```

The first array has 10 elements of data type `ARRAY[11:14] OF UNIQUE something`. There is a total of 40 elements of data type `something` in the attribute named `sectors`. Within each `ARRAY[11:14]`, no duplicates may occur; however, the same `something` instance may occur in two different `ARRAY[11:14]` values within a single value for the attribute named `sectors`.

8.2.2 List data type

A LIST data type has as its domain sequences of like elements. The optional lower and upper bounds, which are integer-valued expressions, define the minimum and maximum number of elements that can be held in the collection defined by a LIST data type. A LIST data type definition may optionally specify that a list value cannot contain duplicate elements.

Syntax:

```
250 list_type = LIST [ bound_spec ] OF [ UNIQUE ] instantiable_type .
185 bound_spec = '[' bound_1 ':' bound_2 ']' .
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
```

Rules and restrictions:

a) The `bound_1` expression shall evaluate to an integer value greater than or equal to zero. It gives the lower bound, which is the minimum number of elements that can be in a list value of this data type. `bound_1` shall not produce the indeterminate (?) value.

b) The `bound_2` expression shall evaluate to an integer value greater than or equal to `bound_1`, or an indeterminate (?) value. It gives the upper bound, which is the maximum number of elements that can be in a list value of this data type.

   If this value is indeterminate (?) the number of elements in a list value of this data type is not bounded from above.

c) If the `bound_spec` is omitted, the limits are `[0:?]`.

d) If the UNIQUE keyword is specified, each element in a list value of this data type shall be different from (that is, not instance equal to) every other element in the same list value.

EXAMPLE This example defines a list of arrays. The list can contain zero to ten arrays. Each array of ten integers shall be different from all other arrays in a particular list.

```
complex_list : LIST[0:10] OF UNIQUE ARRAY[1:10] OF INTEGER;
```

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

8.2.3 Bag data type

A BAG data type has as its domain unordered collections of like elements. The optional lower and upper bounds, which are integer-valued expressions, define the minimum and maximum number of elements that can be held in the collection defined by a BAG data type.

Syntax:

```
180 bag_type = BAG [ bound_spec ] OF instantiable_type .
185 bound_spec = '[' bound_1 ':' bound_2 ']' .
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
```

Rules and restrictions:

a) The `bound_1` expression shall evaluate to an integer value greater than or equal to zero. It gives the lower bound, which is the minimum number of elements that can be in a bag value of this data type. `bound_1` shall not produce the indeterminate (`?`) value.

b) The `bound_2` expression shall evaluate to an integer value greater than or equal to `bound_1`, or an indeterminate (`?`) value. It gives the upper bound, which is the maximum number of elements that can be in a bag value of this data type.

   If this value is indeterminate (`?`), the number of elements in a bag value of this data type is not be bounded from above.

c) If the `bound_spec` is omitted, the limits are `[0:?]`.

EXAMPLE This example defines an attribute as a bag of point (where point is a named data type assumed to have been declared elsewhere).

```
a_bag_of_points : BAG OF point;
```

The value of the attribute named `a_bag_of_points` can contain zero or more points. The same point instance may appear more than once in the value of `a_bag_of_points`.

If the value is required to contain at least one element, the specification can provide a lower bound, as in:

```
a_bag_of_points : BAG [1:?] OF point;
```

The value of the attribute named `a_bag_of_points` now must contain at least one point.

8.2.4 Set data type

A SET data type has as its domain unordered collections of like elements. The SET data type is a specialization of the BAG data type. The optional lower and upper bounds, which are integer-valued expressions, define the minimum and maximum number of elements that can be held in the collection defined by a SET data type. The collection defined by SET data type shall not contain two or more elements which are instance equal.

©ISO 2004 — All rights reserved
AVA Documents Document Sharing Hub
Full document details: BS ISO 10303-11:2004

Syntax:

```
303 set_type = SET [ bound_spec ] OF instantiable_type .
185 bound_spec = '[' bound_1 ':' bound_2 ']' .
183 bound_1 = numeric_expression .
184 bound_2 = numeric_expression .
```

Rules and restrictions:

a) The `bound_1` expression shall evaluate to an integer value greater than or equal to zero. It gives the lower bound, which is the minimum number of elements that can be in a set value of this data type. `bound_1` shall not produce the indeterminate (`?`) value.

b) The `bound_2` expression shall evaluate to an integer value greater than or equal to `bound_1`, or an indeterminate (`?`) value. It gives the upper bound, which is the maximum number of elements that can be in a set value of this data type.

   If this value is indeterminate (`?`), the number of elements in a set value of this data type is not be bounded from above.

c) If the `bound_spec` is omitted, the limits are `[0:?]`.

d) Each element in an occurrence of a SET data type shall be different from (that is, not instance equal to) every other element in the same set value.

EXAMPLE This example defines an attribute as a set of points (a named data type assumed to have been declared elsewhere).

```
a_set_of_points : SET OF point;
```

The attribute named `a_set_of_points` can contain zero or more points. Each point instance (in the set value) is required to be different from every other point in the set.

If the value is required to have no more than 15 points, the specification can provide an upper bound, as in:

```
a_set_of_points : SET [0:15] OF point;
```

The value of the attribute named `a_set_of_points` now may contain no more than 15 points.

8.2.5 Value uniqueness on aggregates

Uniqueness among the elements of an aggregation is based upon instance comparison (see 12.2.2). Aggregates can be constrained to be value unique among their elements through the use of the VALUE_UNIQUE function (see 15.29).

EXAMPLE A set is constrained to be value unique.

```
TYPE value_unique_set = SET OF a;
WHERE
   wr1 : value_unique(SELF);
END_TYPE;
```

NOTE Modeller-defined value uniqueness can be specified via a pair of functions, called, for example, `my_equal` and `my_unique`, as shown in the following pseudo-code.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

```
FUNCTION my_equal(v1,v2: GENERIC:gen): LOGICAL;
 (*"my_equal" Returns TRUE if v1 'equals' v2 *)
END_FUNCTION;

FUNCTION my_unique(c: AGGREGATE OF GENERIC): LOGICAL;
  (*"my_unique" Returns FALSE if two elements of c have the same 'value'
     Else returns UNKNOWN if any element comparison is UNKNOWN
     Otherwise returns TRUE *)
LOCAL
  result   : LOGICAL;
  unknownp : BOOLEAN := FALSE;
END_LOCAL;
IF (SIZEOF(c) = 0) THEN
  RETURN(TRUE); END_IF;
REPEAT i := LOINDEX(c) TO (HIINDEX(c) - 1);
  REPEAT j := (i+1) TO HIINDEX(c);
    result := my_equal(c[i], c[j]);
    IF (result = TRUE) THEN
      RETURN(FALSE); END_IF;
    IF (result = UNKNOWN) THEN
      unknownp := TRUE; END_IF;
  END_REPEAT;
END_REPEAT;
IF unknownp THEN
  RETURN(UNKNOWN);
ELSE
  RETURN(TRUE);
END_IF;
END_FUNCTION;
```

The function `my_equal` should have the following properties which enable the building of equivalence classes. In the following $\mathcal{S}$ is the set of objects under consideration and `my_equal(i, j)`, where `i` and `j` are in $\mathcal{S}$, returns one of [FALSE, UNKNOWN, TRUE].

a) `my_equal(i, i)` is TRUE for all `i` in $\mathcal{S}$ (since indeterminate (`?`) is not in $\mathcal{S}$, this does not require `my_equal(?, ?)` to be TRUE);

b) `my_equal(i, j)` = `my_equal(j, i)` for all `i, j` in $\mathcal{S}$;

c) (`my_equal(i, j)` = TRUE) AND (`my_equal(j, k)` = TRUE) implies (`my_equal(i, k)` = TRUE) for all `i, j, k` in $\mathcal{S}$.

8.3 Named data types

The named data types are the data types that may be declared in a formal specification. There are two kinds of named data types: entity data types and defined data types. This subclause covers the referencing of named data types; the declaration of these data types is covered in clause 9.

8.3.1 Entity data type

Entity data types are established by ENTITY declarations (see 9.2). An entity data type is assigned an entity identifier by the user. An entity data type is referenced by this identifier.

Syntax:

```
152 entity_ref = entity_id .
```

 ©ISO 2004 — All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

Rules and restrictions:

`entity_ref` shall be a reference to an entity which is visible in the current scope (see clause 10).

EXAMPLE 1 This example uses a `point` entity data type as the representation of an attribute.

```
ENTITY point;
    x, y, z : REAL;
END_ENTITY;

ENTITY line;
    p0, p1 : point;
END_ENTITY;
```

The line entity has two attributes named `p0` and `p1`. The data type of each of these attributes is `point`.

8.3.2 Defined data type

Defined data types are declared by TYPE declarations (see 9.1). A defined data type is assigned a type identifier by the user. A defined data type is referenced by this identifier.

Syntax:

```
162 type_ref = type_id .
```

Rules and restrictions:

`type_ref` shall be the name of a defined data type which is visible in the current scope (see clause 10).

EXAMPLE The following is a defined data type used to indicate the units of measure associated with an attribute.

```
TYPE volume = REAL;
END_TYPE;

ENTITY PART;
...
  bulk : volume;
END_ENTITY;
```

The attribute named `bulk` is represented as a real number, but the use of the defined data type, `volume`, helps to clarify the meaning and context of the real number. The real number means volume, rather than some other thing that might be represented by a REAL.

8.4 Constructed data types

There are two kinds of constructed data types in EXPRESS: ENUMERATION data types and SELECT data types. These data types have similar syntactic structures and may only be used to provide underlying representations of defined data types (see 9.1).

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

8.4.1 Enumeration data type

An ENUMERATION data type has as its domain a set of names. The extent of this set of names is determined depending on the type of ENUMERATION data type. The following types of ENUMERATION data types are distinguished:

— ENUMERATION that is extensible;

— ENUMERATION that is extending an extensible ENUMERATION, in other words: that is based on an extensible ENUMERATION;

— ENUMERATION that is neither extensible nor extending.

The names that an ENUMERATION declares are the only valid values of the ENUMERATION data type. Each name in the domain is referred to as an enumeration item, and is designated by an `enumeration_id`.

An ENUMERATION data type that is neither an extensible nor an extending ENUMERATION data type has as its domain the ordered set of enumeration items in its declaration.

An extensible ENUMERATION data type has as its domain the set of the enumeration items in its declaration plus the union of the sets of enumeration items comprising the domains of all extending enumeration data types. An extensible enumeration data type is a generalization of the enumeration data types that are based on it. An extensible ENUMERATION data type is specified using the EXTENSIBLE reserved word.

An extending enumeration data type has as its domain the set of enumeration items in its declaration plus the enumeration items specified directly, not via extension, in the extensible enumeration data type upon which it is based. An extending ENUMERATION data type is specified using the BASED_ON reserved word.

An enumeration data type may be both an extensible and an extending enumeration data type. An extensible ENUMERATION may be specified without enumeration items and may be based on another extensible ENUMERATION and not specify any enumeration items extending that base ENUMERATION. The based-on relationship is transitive, that is, an extending enumeration is still based on the highest level extensible enumeration even through several levels of based-on relationships; it includes all enumeration items of both the intermediate extensible enumerations and the highest level extensible enumeration.

NOTE 1 An extensible enumeration that is extended two or more times in a single context may have a larger domain than its extensions, in which case it really is a generalization.

NOTE 2 In the first edition of this part of ISO 10303 the ordering of the enumeration items defined a value ordering. In this edition of this part of ISO 10303, no ordering is defined except as noted in rule (d) below. This is to allow for extensible enumeration data types, in which the ordering of the extensions cannot be determined.

Syntax:

```
213 enumeration_type = [ EXTENSIBLE ] ENUMERATION [ ( OF
                        enumeration_items ) | enumeration_extension ] .
211 enumeration_items = '(' enumeration_id { ',' enumeration_id } ')' .
209 enumeration_extension = BASED_ON type_ref [ WITH enumeration_items ] .
```

 ©ISO 2004 — All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

Rules and restrictions:

a) An enumeration data type shall only be used as the underlying data type of a defined data type.

b) An enumeration data type may be extended only if the reserved word EXTENSIBLE is specified in its definition.

c) The `type_ref` in an `enumeration_extension` shall refer to an extensible enumeration type.

d) For comparison purposes, the ordering of the values of an enumeration data type that is neither extensible nor extending may be determined by their relative position in the `enumeration_id` list; the first occurring item shall be less than the second; the second less than the third, and so on.

e) There is no ordering of the values of an extensible enumeration or an extending enumeration.

f) An enumeration that is neither extensible nor extending shall specify enumeration items as its domain.

g) An enumeration that is not extensible, but extending shall specify enumeration items that extend the domain of the extensible enumeration upon which it is based.

h) Two different ENUMERATION data types may contain the same `enumeration_id` in their sets of names. If these enumeration data types are not extensions to the same extensible enumeration data type, their `enumeration_id`s describe different concepts, even though their local names may be identical. In this case, any reference to the `enumeration_id` (for example, in an expression) shall be qualified with the data type identifier to ensure that the reference is unambiguous. The reference then appears as: `type_id.enumeration_id`.

NOTE 3 A `type_id` for use in qualifying an `enumeration_id` is always defined as an ENUMERATION data type.

EXAMPLE 1 This example uses ENUMERATION data types to show how different kinds of vehicles might travel.

```
TYPE car_can_move = ENUMERATION OF
     (left, right, backward, forward);
END_TYPE;

TYPE plane_can_move = ENUMERATION OF
     (left, right, backward, forward, up, down);
END_TYPE;
```

The enumeration item `left` has two independent definitions, one being given by each data type of which it is a component. There is no connection between these two definitions of the identifier `left`. A reference to `left` or `right`, by itself, is ambiguous. To resolve the ambiguity, a reference to either of these values is qualified by the data type name, for example, `car_can_move.left`.

i) An extensible enumeration and its extensions define a domain consisting of `enumeration_id`s. Within that domain all occurrences of the same `enumeration_id` designate the same value even when the `enumeration_id` is specified in multiple enumeration data types contributing to that domain.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO 10303-11:2004

EXAMPLE 2 The following EXPRESS results in a single enumeration item named `red` as both `stop_light` and `canadian_flag` extend the domain of colour.

```
TYPE colour = EXTENSIBLE ENUMERATION; END_TYPE;

TYPE stop_light = ENUMERATION BASED_ON colour WITH (red, yellow,
green); END_TYPE;

TYPE canadian_flag = ENUMERATION BASED_ON colour WITH (red,
white); END_TYPE;
```

j) The type declaration which declares the enumeration data type shall not include a domain (where) rule.

NOTE 4 The above rules ensure that a defined data type names an enumeration data type, and the defined data type is not a specialization of the enumeration data type.

EXAMPLE 3 The following example shows how an extensible enumeration may be used to model a context dependent concept of approval. `general_approval`, as its name suggests, is the most general concept of approval explicitly defining only two values. By declaring `general_approval` to be an extensible enumeration allows it to take on context dependent values in schemas which declare extensions to it. If used to represent the domain of an attribute, the allowed values of the attribute are context dependent.

```
SCHEMA s1;

TYPE general_approval = EXTENSIBLE ENUMERATION OF (approved, rejected);
END_TYPE;

END_SCHEMA;

SCHEMA s2;

USE FROM s1 (general_approval);

TYPE domain2_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH (pending);
END_TYPE;

END_SCHEMA;

SCHEMA s3;

USE FROM s1 (general_approval);

TYPE domain3_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH (cancelled);
END_TYPE;

END_SCHEMA;

SCHEMA s4;

USE FROM s2 (domain2_approval);

REFERENCE FROM s3 (domain3_approval);

TYPE specific_approval = ENUMERATION BASED_ON domain2_approval WITH (rework);
END_TYPE;
```

 ©ISO 2004 — All rights reserved

AVA Documents Document Sharing Hub

Full document details: BS ISO 10303-11:2004

```
END_SCHEMA;

SCHEMA s5;

USE FROM s1 (general_approval);

TYPE redundant_approval = ENUMERATION BASED_ON general_approval WITH (approved);
END_TYPE;

END_SCHEMA;
```

In the context of schema `s1`:

— `general_approval` has the domain (`approved`, `rejected`).

In the context of schema `s2`:

— `general_approval` has the domain (`approved`, `rejected`, `pending`);

— `domain2_approval` has the domain (`approved`, `rejected`, `pending`).

In the context of schema `s3`:

— `general_approval` has the domain (`approved`, `rejected`, `cancelled`);

— `domain3_approval` has the domain (`approved`, `rejected`, `cancelled`).

In the context of schema `s4`:

— `general_approval` has the domain (`approved`, `rejected`, `pending`, `cancelled`, `rework`);

— `domain2_approval` has the domain (`approved`, `rejected`, `pending`, `rework`);

— `domain3_approval` has the domain (`approved`, `rejected`, `cancelled`);

— `specific_approval` has the domain (`approved`, `rejected`, `pending`, `rework`).

In the context of schema `s5`:

— `general_approval` has the domain (`approved`, `rejected`).

— `redundant_approval` has the domain (`approved`, `rejected`).

8.4.2 Select data type

A SELECT data type defines a data type that enables a choice among several named data types. The SELECT data type is a generalization of these named data types in its domain. The defined type for which the SELECT data type is the underlying representation, may add constraints on its domain by declaring local rules. A SELECT data type may be EXTENSIBLE or not.

A SELECT data type that is neither EXTENSIBLE nor extending has as its domain the union of the domains of the named data types in its select list.

An EXTENSIBLE SELECT data type has as its domain the union of the domains of the named data types in its own select list, plus the union of the domains of all extending SELECT data types. An EXTENSIBLE SELECT data type is specified using the EXTENSIBLE reserved word.

©ISO 2004 — All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004

An extending SELECT data type has as its domain the named data types in its own select list, plus the named data types specified directly, not via extension, in the EXTENSIBLE SELECT data type upon which it is based. An extending SELECT data type is specified using the BASED_ON reserved word.

A SELECT data type may be both an EXTENSIBLE and an extending SELECT data type. An EXTENSIBLE SELECT may be specified without a select list and may be based on another EXTENSIBLE SELECT and not specify a select list extending that base SELECT.

An EXTENSIBLE SELECT data type and only an EXTENSIBLE one may be constrained to have only ENTITY instances in its domain by using the reserved word GENERIC_ENTITY. In this case, all SELECT elements shall be generic-entity elements, where generic-entity element is defined as being either an ENTITY data type or a SELECT of generic-entity elements. All extensions of that SELECT data type shall be generic-entity SELECT data types and shall specify the GENERIC_ENTITY reserved word.

Syntax:

```
302 select_type = [ EXTENSIBLE [ GENERIC_ENTITY ] ] SELECT
                  [ select_list | select_extension ] .
301 select_list = '(' named_types { ',' named_types } ')' .
300 select_extension = BASED_ON type_ref [ WITH select_list ] .
```

Rules and restrictions:

a) Each item in the select list shall be an ENTITY data type or a defined data type.

b) A SELECT data type shall only be used as the underlying type of a defined data type.

c) A SELECT data type may be extended only if the reserved word EXTENSIBLE is specified in its definition.

d) The `type_ref` in an `select_extension` shall refer to an extensible select type.

e) A SELECT that is neither extensible nor extending shall specify a non-empty select list as its domain.

f) A SELECT that is not extensible, but extending shall specify a non-empty select list that extends the domain of the extensible SELECT upon which it is based.

NOTE The value of a SELECT data type may be a value of more than one of the named data types specified in the select list for that SELECT data type.

EXAMPLE 1 If *a* and *b* are subtypes of *c*, and if they are related by an ANDOR expression, and if there is a data type defined by `SELECT (a,b)`, then it may be that the value of the SELECT data type is an *a* and a *b* at the same time.

EXAMPLE 2 A choice must be made among several types of things in a given context.

```
TYPE attachment_method = EXTENSIBLE SELECT(nail, screw);
END_TYPE;

TYPE permanent_attachment = SELECT BASED ON attachment_method WITH (glue, weld);
END_TYPE;
```

 ©ISO 2004 — All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO 10303-11:2004