

TECHNICAL REPORT

**ISO/TR
10303-12**

First edition
1997-07-15

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

Part 12: Description methods: The EXPRESS-I language reference manual

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

*Partie 12: Méthodes descriptives: Le manuel de référence du langage
EXPRESS-I*

This material is reproduced from ISO documents under International Organization for Standardization (ISO) Copyright License number IHS/ICC/1996. Not for resale. No part of these ISO documents may be reproduced in any form, electronic retrieval system or otherwise, except as allowed in the copyright law of the country of use, or with the prior written consent of ISO (Case postale 56, 1211 Geneva 20, Switzerland, Fax +41 22 734 10 79), IHS or the ISO Licensor's members.



Reference number
ISO/TR 10303-12
AVA Documents
Document Sharing Hub

ISO/TR 10303-12:1997(E)

Contents	Page
1 Scope	1
2 Normative references	2
3 Definitions	3
3.1 Terms defined in ISO 10303-1	3
3.2 Terms defined in ISO 10303-11	3
3.3 Terms defined in ISO 10303-31	3
3.4 Other definitions	4
3.4.1 attribute	4
3.4.2 information base	4
3.4.3 object base	4
3.4.4 schema	4
3.4.5 type	4
3.4.6 universe of discourse	4
4 Conformance requirements	5
4.1 Formal specifications written in EXPRESS-I	5
4.1.1 Conformance levels	5
4.2 Implementations of EXPRESS-I	5
5 Fundamental principles	6
6 Language elements	7
6.1 Character set	7
6.1.1 Digits	7
6.1.2 Letters	8
6.1.3 Special characters	8
6.1.4 Underscore	8
6.1.5 Whitespace	8
6.1.6 Remarks	9
6.2 Reserved words	10
6.2.1 Keywords	10
6.2.2 Reserved words which are operators	11
6.2.3 Built-in constants	11

© ISO 1997

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 the publisher.

International Organization for Standardization
Case Postale 56 • CH-2111 Genève 20 • Switzerland

Printed in Switzerland

6.2.4	Built-in functions	11
6.2.5	Built-in procedures	11
6.3	Symbols	12
6.4	Identifiers and references	12
7	Named domains	15
7.1	Entity domain	15
7.2	Enumeration domain	15
7.3	Select domain	15
7.4	Type domain	15
8	Values and instances	16
8.1	Base values	16
8.1.1	Binary value	16
8.1.2	Boolean value	16
8.1.3	Number value	16
8.1.4	Integer value	17
8.1.5	Logical value	17
8.1.6	Real value	17
8.1.7	String value	18
8.1.8	Enumeration value	20
8.2	Aggregation values	20
8.3	Simple instance	21
8.4	Type instance	21
8.5	Select instance	22
8.6	Enumeration instance	22
8.7	Entity instance	23
8.7.1	Attributes	24
8.7.2	Supertypes and subtypes	26
8.8	Constant instance	27
8.9	Schema data instance	27
8.10	Model display	28
9	Abstract test case specification	30
9.1	Context	30
9.2	Parameters	31
9.2.1	Formal parameter	31
9.2.2	Actual parameter	32
9.3	Test case	32
9.4	Test objective	33
9.4.1	Test purpose	34
9.4.2	Test reference	34
9.4.3	Test criteria	34
9.4.4	Test notes	35
9.5	Test realization	35
10	Interfaces	36

10.1	Schema instance interface	36
10.2	Schema reference	36
10.3	Context data references	37
11	Scope and visibility	38
11.1	Scope rules	38
11.2	Visibility rules	39
11.2.1	General rules of visibility	39
11.2.2	Named data type identifier visibility rules	40
11.3	Explicit item rules	41
11.3.1	Alias statement	42
11.3.2	Attribute	42
11.3.3	Constant	42
11.3.4	Constant instance	42
11.3.5	Context	42
11.3.6	Entity	43
11.3.7	Entity instance	43
11.3.8	Enumeration item	44
11.3.9	Enumeration instance	44
11.3.10	Function	44
11.3.11	Model	44
11.3.12	Parameter	44
11.3.13	Procedure	45
11.3.14	Query expression	45
11.3.15	Repeat statement	45
11.3.16	Rule label	45
11.3.17	Schema data instance	46
11.3.18	Select instance	46
11.3.19	Simple instance	46
11.3.20	Test case	46
11.3.21	Type	47
11.3.22	Type instance	47
11.3.23	Type label	47
11.3.24	Variable	47
12	Mapping from EXPRESS to EXPRESS-I	48
12.1	Mapping of EXPRESS schema	48
12.1.1	Mapping of use and reference	49
12.2	Mapping of EXPRESS simple data types	50
12.3	Mapping of aggregation data types	51
12.4	Mapping of EXPRESS defined data type	52
12.5	Mapping of EXPRESS enumeration type	52
12.6	Mapping of EXPRESS select type	53
12.6.1	Simple select case	53
12.6.2	Complex select case	53
12.7	Mapping of EXPRESS constant	54

12.8	Mapping of EXPRESS entity	55
12.9	Mapping of EXPRESS entity attributes	55
12.9.1	Explicit attribute	56
12.9.2	Derived and inverse attributes	56
12.9.3	Attribute with a simple domain	57
12.9.4	Attribute with an entity domain	58
12.9.5	Attribute with a type, select or enumeration domain	59
12.10	Mapping of supertypes and subtypes	60
12.10.1	Mapping of redeclared attributes	63

Annexes

A	Syntax description of EXPRESS-I	65
A.1	Tokens	65
A.1.1	Keywords	65
A.1.2	Character classes	68
A.2	Lexical elements	69
A.2.1	Remarks	69
A.3	Interpreted identifiers	70
A.4	Grammar rules	70
A.5	Cross reference listing	77
B	Protocol implementation conformance statement (PICS)	88
B.1	EXPRESS-I language parser	88
C	Information object registration	89
D	Language specification syntax	90
D.1	The syntax of the specification	90
D.2	Special character notation	91
E	Example test cases	93
E.1	Test case 1	93
E.2	Test case 2	95
E.3	Test case 3	96
E.4	Test case 4	97
F	Usage notes	100
F.1	EXPRESS data examples	100
F.2	Abstract test cases	100
F.3	Object bases	101
F.3.1	Input	101
F.3.2	Output	101
F.3.3	Code testing	101
F.4	Non-EXPRESS data examples	102
G	Technical discussions	103
G.1	Abstract test cases	103
G.2	Relationship with EXPRESS	103
G.3	Object references	104
G.4	Aggregations	104
G.5	String values	104

G.6	Model testing and validation	105
G.7	Enhancement of test case capabilities	105
G.8	Compatibility with EXPRESS	105
G.9	Trial Usage	105
G.10	Alphabet extensions	106
G.11	Supertype mapping	106
G.12	CD ballot comments — 1995	107
G.12.1	Test case support	107
G.12.2	Complex entity instances	107
G.12.3	Type instances	108
H	Bibliography	109
	Index	110

Figures

1	The major elements of the EXPRESS-I language	xv
---	--	----

Tables

1	Keywords common to EXPRESS-I and EXPRESS	10
2	Additional EXPRESS-I keywords	10
3	The EXPRESS-I use of EXPRESS operators	11
4	The EXPRESS-I use of EXPRESS constants	11
5	The EXPRESS-I use of EXPRESS functions	11
6	The EXPRESS-I use of EXPRESS procedures	12
7	Symbols common to EXPRESS-I and EXPRESS	12
8	Additional EXPRESS-I symbols	12
9	Scope and identifier defining EXPRESS-I items	38
10	Scope and identifier defining EXPRESS items utilised by EXPRESS-I	39
11	Scope and visibility rules	41
12	Summary overview of EXPRESS to EXPRESS-I mappings	48
13	Overview of SCHEMA mapping	49
14	Simple type mapping	51
15	Mapping of AGGREGATES	51
16	Overview of ENTITY mapping	55
17	Overview of SUPERTYPE and SUBTYPE mapping	61

Foreword

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

The main task of technical committees is to prepare International Standards, but in exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

- type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;
- type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;
- type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard (“state of the art”, for example).

Technical reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/TR 10303-12, which is a Technical Report of type 2, was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC4, *Industrial data*.

There is an urgent need for guidance on how EXPRESS can be used to represent data so that it is ‘human’ interpretable. This need is supported by the importance of involving domain experts in the development of industrial data standards and in the development of abstract test cases that can be used to verify conformance to such standards.

This document is being issued in the Technical Report (type 2) series of publications (according to subclause G.3.2.2 of part 1 of the ISO/IEC Directives as a ‘prospective standard for provisional application’ in the field of EXPRESS information modeling as there is an urgent need for guidance on how standards in this field should be used to meet an identified need.

This document is not to be regarded as an ‘International Standard’. It is proposed for provisional application so that experience of its use in practice may be gathered. Comments on the content of this document should be sent to the ISO Central Secretariat.

A review of this Technical Report (type 2) will be carried out not later than three years after its publication with the options of: extension for another three years; conversion into an International Standard; or withdrawal.

ISO 10303 consists of the following parts under the general title *Industrial automation systems and integration – Product data representation and exchange*:

- Part 1, Overview and fundamental principles;
- Part 11, Description methods: The EXPRESS language reference manual;
- Part 12, Description method: The EXPRESS-I language reference manual;
- Part 21, Implementation methods: Clear text encoding of the exchange structure;
- Part 22, Implementation method: Standard data access interface specification;
- Part 23, Implementation method: C++ language binding to the standard data access interface;
- Part 24, Implementation method: C language binding to the standard data access interface;
- Part 26, Implementation method: Interface definition language binding to the standard data access interface;
- Part 31, Conformance testing methodology and framework: General concepts;
- Part 32, Conformance testing methodology and framework: Requirements on testing laboratories and clients;
- Part 33, Conformance testing methodology and framework: Structure and use of abstract test suites;
- Part 34, Conformance testing methodology and framework: Abstract test methods;
- Part 35, Conformance testing methodology and framework: Abstract test methods for standard data access interface implementations;
- Part 41, Integrated generic resources: Fundamentals of product description and support;
- Part 42, Integrated generic resources: Geometric and topological representation;
- Part 43, Integrated generic resources: Representation structures;
- Part 44, Integrated generic resources: Product structure configuration;
- Part 45, Integrated generic resource: Materials;
- Part 46, Integrated generic resources: Visual presentation;
- Part 47, Integrated generic resource: Shape variation tolerances;

- Part 49, Integrated generic resource: Process structure and properties;
- Part 101, Integrated application resource: Draughting;
- Part 104, Integrated application resource: Finite element analysis;
- Part 105, Integrated application resource: Kinematics;
- Part 106, Integrated application resource: Building construction core model;
- Part 201, Application protocol: Explicit draughting;
- Part 202, Application protocol: Associative draughting;
- Part 203, Application protocol: Configuration controlled design;
- Part 204, Application protocol: Mechanical design using boundary representation;
- Part 205, Application protocol: Mechanical design using surface representation;
- Part 207, Application protocol: Sheet metal die planning and design;
- Part 208, Application protocol: Life cycle management - Change process;
- Part 209, Application protocol: Composite and metallic structural analysis and related design;
- Part 210, Application protocol: Electronic assembly, interconnect, and packaging design;
- Part 212, Application protocol: Electrotechnical design and installation;
- Part 213, Application protocol: Numerical control process plans for machined parts;
- Part 214, Application protocol: Core data for automotive mechanical design;
- Part 215, Application protocol: Ship arrangement;
- Part 216, Application protocol: Ship moulded forms;
- Part 217, Application protocol: Ship piping;
- Part 218, Application protocol: Ship structures;
- Part 220, Application protocol: Process planning, manufacture, and assembly of layered electronic products;
- Part 221, Application protocol: Functional data and their schematic representation for process plant;

- Part 222, Application protocol: Exchange of product data for composite structures;
- Part 223, Application protocol: Exchange of design and manufacturing product information for cast parts;
- Part 224, Application protocol: Mechanical product definition for process plans using mechanical feature;
- Part 225, Application protocol: Building elements using explicit shape representation;
- Part 226, Application protocol: Ship mechanical systems;
- Part 227, Application protocol: Plant spatial configuration;
- Part 228, Application protocol: Building services: Heating, ventilation, and air conditioning;
- Part 229, Application protocol: Exchange of design and manufacturing product information for forged parts;
- Part 230, Application protocol: Building structural frame: Steelwork;
- Part 231, Application protocol: Process engineering data: Process design and process specification of major equipment;
- Part 232, Application Protocol: Technical data package;
- Part 301, Abstract test suite: Explicit draughting;
- Part 302, Abstract test suite: Associative draughting;
- Part 303, Abstract test suite: Configuration controlled design;
- Part 304, Abstract test suite: Mechanical design using boundary representation;
- Part 305, Abstract test suite: Mechanical design using surface representation;
- Part 307, Abstract test suite: Sheet metal die planning and design;
- Part 308, Abstract test suite: Life cycle management - Change process;
- Part 309, Abstract test suite: Composite and metallic structural analysis and related design;
- Part 310, Abstract test suite: Electronic assembly, interconnect, and packaging design;
- Part 312, Abstract test suite: Electrotechnical design and installation;

- Part 313, Abstract test suite: Numerical control process plans for machined parts;
- Part 314, Abstract test suite: Core data for automotive mechanical design;
- Part 315, Abstract test suite: Ship arrangement;
- Part 316, Abstract test suite: Ship moulded forms;
- Part 317, Abstract test suite: Ship piping;
- Part 318, Abstract test suite: Ship structures;
- Part 320, Abstract test suite: Process planning, manufacture, and assembly of layered electronic products;
- Part 321, Abstract test suite: Functional data and their schematic representation for process plant;
- Part 322, Abstract test suite: Exchange of product data for composite structures;
- Part 323, Abstract test suite: Exchange of design and manufacturing product information for cast parts;
- Part 324, Abstract test suite: Mechanical product definition for process plans using mechanical features;
- Part 325, Abstract test suite: Building elements using explicit shape representation;
- Part 326, Abstract test suite: Ship mechanical systems;
- Part 327, Abstract test suite: Plant spatial configuration;
- Part 328, Abstract test suite: Building services: Heating, ventilation, and air conditioning;
- Part 329, Abstract test suite: Exchange of design and manufacturing product information for forged parts;
- Part 330, Abstract test suite: Building structural frame: Steelwork;
- Part 331, Abstract test suite: Process engineering data: Process design and process specification of major equipment;
- Part 332, Abstract test suite: Technical data package;
- Part 501, Application interpreted construct: Edge-based wireframe;

- Part 502, Application interpreted construct: Shell-based wireframe;
- Part 503, Application interpreted construct: Geometrically bounded 2D wireframe;
- Part 504, Application interpreted construct: Draughting annotation;
- Part 505, Application interpreted construct: Drawing structure and administration;
- Part 506, Application interpreted construct: Draughting elements;
- Part 507, Application interpreted construct: Geometrically bounded surface;
- Part 508, Application interpreted construct: Non-manifold surface;
- Part 509, Application interpreted construct: Manifold surface;
- Part 510, Application interpreted construct: Geometrically bounded wireframe;
- Part 511, Application interpreted construct: Topologically bounded surface;
- Part 512, Application interpreted construct: Faceted boundary representation;
- Part 513, Application interpreted construct: Elementary boundary representation;
- Part 514, Application interpreted construct: Advanced boundary representation;
- Part 515, Application interpreted construct: Constructive solid geometry;
- Part 517, Application interpreted construct: Mechanical design geometric presentation;
- Part 518, Application interpreted construct: Mechanical design shaded representation.

The structure of this International Standard is described in ISO 10303-1. The numbering of the parts of this International Standard reflects its structure:

- Parts 11 to 12 specify the description methods,
- Parts 21 to 26 specify the implementation methods,
- Parts 31 to 35 specify the conformance testing methodology and framework,
- Parts 41 to 49 specify the integrated generic resources,
- Parts 101 to 106 specify the integrated application resources,
- Parts 201 to 232 specify the application protocols,
- Parts 301 to 332 specify the abstract test suites, and

©ISO

ISO/TR 10303-12:1997(E)

- Parts 501 to 518 specify the application interpreted constructs.

Should further parts of ISO 10303 be published, they will follow the same numbering pattern.

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

Introduction

ISO 10303 is an International Standard for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving.

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

This part of ISO 10303 specifies the elements of the EXPRESS-I 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.

Language Overview

EXPRESS-I is the name of a formal data representation and abstract test case specification language. It may be used to exemplify the information requirements of other parts of this International Standard and is a companion to the EXPRESS and EXPRESS-G languages. 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 elements of ISO 10303 in a less formal manner would eliminate the possibility of employing computer automation in checking for inconsistencies in presentation or specification.
- Focus on the display of the realisation of the properties 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.
- Avoid, as far as possible, specific implementation views.
- Provide a means of displaying small populations of EXPRESS schemas.
- Provide a means of supporting the specification of abstract test suites for information model processors.

In EXPRESS-I, entity instances are represented in terms of attribute values: 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.

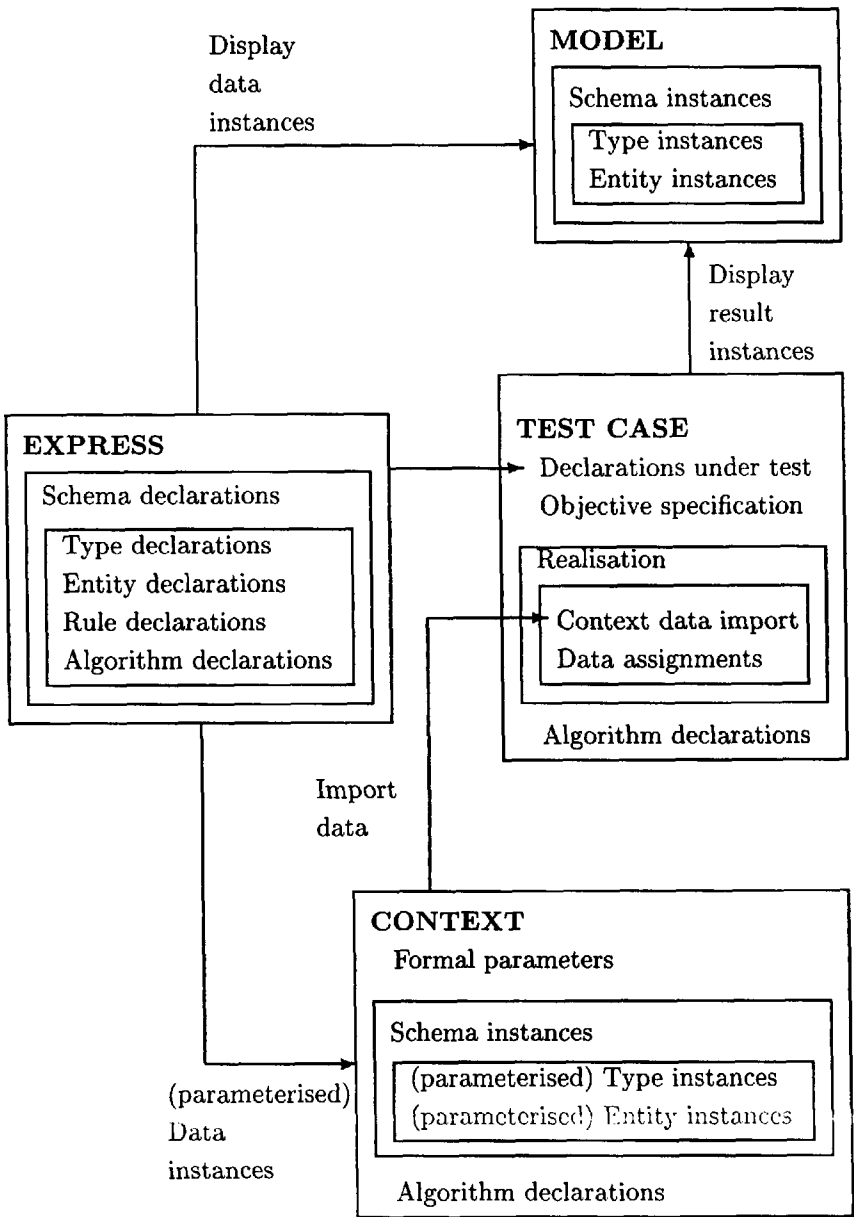


Figure 1 – The major elements of the EXPRESS-I language

The EXPRESS-I language provides a means of displaying instantiations of EXPRESS data elements. The language is designed principally for human readability and for ease of mapping between EXPRESS-I instances and the definitions in an EXPRESS schema. Elsewhere in this International Standard, for example ISO 10303-21, there are specifications for computer-efficient methods for instantiating a schema. EXPRESS-I is not intended to be a replacement for such methods.

The major elements of the language are shown in figure 1. The language has two major parts. The first part is for the display of data instances. Data may be displayed on an entity by entity

basis, on a schema basis or as a collection of schema instances which are taken to be a display of some information model of a universe of discourse. Within the EXPRESS-I language these are called *object instances*, *schema data instances* and, *model*. In figure 1 the information model is assumed to have been defined using EXPRESS.

The second part of the language is for the specification of Abstract Test Cases for the purposes of formally describing tests to be performed against an implementation of an EXPRESS-defined information model. The language constructs provided for this purpose are the *test case* and the *context*. This portion of the language also utilises the procedural aspects of the EXPRESS language. Instances of data may be parameterised and stored in a context. Many different test cases may assign values for the parameterised data in a context and use that data as part of their test specification.

The data instances resulting from the application of a test case may be displayed via the constructs provided in first part of the language.

NOTE – The examples of EXPRESS-I 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 this International Standard. They are crafted to show particular features of EXPRESS-I. Any similarity between the examples and the normative information models or abstract test cases specified in other parts of ISO 10303 should be ignored.

Industrial automation systems and integration — Product data representation and exchange — Part 12 : Description methods: The EXPRESS-I language reference manual

1 Scope

This part of ISO 10303 defines a language by which an instance of (part of) a universe of discourse can be displayed. It also provides a formal description method for supporting the specification of abstract test cases. The language is called EXPRESS-I. It is a companion language to EXPRESS which is specified in ISO 10303-11.

EXPRESS-I is an instantiation language for a conceptual schema language as defined in ISO TR 9007, and the particular conceptual schema language that formed the starting point for EXPRESS-I was EXPRESS. The EXPRESS-I language provides for the display of the state of the objects belonging to a universe of discourse and the information units pertaining to those objects.

The following are within the scope:

- display of instances of schemas;
- display of instances of types and entities;
- abstract test case data;
- mapping from EXPRESS schemas and data types to EXPRESS-I instances.

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

- mapping from other (conceptual schema) languages to EXPRESS-I;
- definition of database formats;
- definition of file formats;
- definition of transfer formats;
- process control;
- information processing;
- exception handling.

EXPRESS-I is not a programming language.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO 10303. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO 10303 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 8824-1:1995, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation*.

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

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

ISO 10303-31:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 31: Conformance testing methodology and framework: General concepts*.

ISO/IEC 10646-1:1993, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*.

3 Definitions

3.1 Terms defined in ISO 10303-1

This part of ISO 10303 makes use of the following terms defined in ISO 10303-1:

- data;
- information;
- information model.

3.2 Terms defined in ISO 10303-11

This part of ISO 10303 makes use of the following terms defined in ISO 10303-11:

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

3.3 Terms defined in ISO 10303-31

This part of ISO 10303 makes use of the following terms defined in ISO 10303-31:

- abstract test case;

- test purpose;
- verdict criteria.

3.4 Other definitions

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

3.4.1 attribute: A trait, quality, or property that is a characteristic of an entity.

3.4.2 information base: A collection of type instances, consistent with each other and with an information model, that hold for an instance of a universe of discourse.

NOTE – An information base may or may not be computer processable. For example, it would not be considered computer processable if it took the form of a handwritten document. On the other hand, if it was in the form of a data base or computer file then it would be considered computer processable, and hence also termed an object base.

3.4.3 object base: An information base that is computer processable.

3.4.4 schema: A collection of closely related items forming a part or the whole of an information model.

3.4.5 type: A representation of a domain of valid values.

3.4.6 universe of discourse: All those real-world objects that are of potential interest. These are a subset of all the real-world objects.

4 Conformance requirements

4.1 Formal specifications written in EXPRESS-I

A formal specification written in EXPRESS-I shall be consistent with a given conformance 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.

4.1.1 Conformance levels

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 given in annex A. A formal specification is referentially valid if all references to EXPRESS-I items are consistent with the scope and visibility rules defined in clause 11.

Level 2: Type checking. This level consists of checking the formal specification to ensure that type compatibility in expressions and assignments, as defined for level 2 checking in ISO 10303-11:1994, are valid.

Level 3: Value checking. This level consists of checking the formal specification to ensure that it complies with level 3 checking defined in ISO 10303-11.

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

4.2 Implementations of EXPRESS-I

An implementation of an EXPRESS-I language parser shall be able to parse any formal specification written in EXPRESS-I, consistent with the constraints associated with that implementation as specified in the PICS (annex B). An EXPRESS-I language parser shall be said to conform to a particular 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-I.

The implementor of an EXPRESS-I 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 B for the purposes of conformance testing.

5 Fundamental principles

It is assumed that the reader of this document is familiar with the EXPRESS language as specified in ISO 10303-11.

The use of EXPRESS-I to display instances neither requires nor implies that there is an associated set of entity, or other, definitions. That is, EXPRESS-I can be used as a language in its own right. However, there normally will be an accompanying set of definitions, typically described in a formal manner using a language such as EXPRESS.

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

- how instance data is accessed or output;
- how instance data is stored or maintained;
- how references to EXPRESS schemas are resolved;
- how or when constraints are checked or reported.

6 Language elements

This clause specifies the basic elements from which sentences in the EXPRESS-I language are composed: the character set, remarks, symbols, reserved words, and identifiers.

The boxed syntax definitions in the body of this document are excerpts from the EXPRESS-I language syntax in annex A which defines the complete syntax of the language and provides any language productions not given here. The method of specifying the syntax is a superset of that used for EXPRESS as defined in clause 6 of ISO 10303-11:1994.

NOTE 1 – For convenience of the reader, the EXPRESS definition method is repeated in annex D, together with the extensions for EXPRESS-I.

The basic language elements are composed into a stream of source text, typically broken into physical lines. A physical line is any number (including zero) of characters ended by a newline (see 6.1.5.2).

NOTE 2 – EXPRESS-I source is easier to read when statements are broken into lines and whitespace is used to set off different constructs.

6.1 Character set

EXPRESS-I source shall use only the characters in the following character set: characters allocated to cells 20 to 7E of row 00 of plane 00 of group 00 of ISO/IEC 10646-1; and the special character `\n` signifying the newline. This set of characters is called the EXPRESS-I character set. Members of this set are referred to by the cell of ISO/IEC 10646-1 in which these characters are defined; these cell numbers are specified in hexadecimal. The printable characters from this set (cells 21–7E of ISO/IEC 10647-1) are combined to form the tokens for the EXPRESS-I language. The EXPRESS-I tokens are keywords, identifiers, symbols, literals, or values. The EXPRESS-I 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. In particular, a real implementation may use some of the control codes defined in ISO/IEC 6429. Such codes are interpreted by the implementation and may result in the inclusion of one or more of the abstract characters of the EXPRESS-I character set in the resulting source.

EXAMPLE 1 – A TAB control code may be interpreted by the implementation as providing one or more space characters to the abstract characters making up the EXPRESS-I definition.

NOTE – This clause only refers to the characters used to specify EXPRESS-I source, and does not specify the domain of characters allowed within a string value.

6.1.1 Digits

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

Syntax:

120 digit = < as EXPRESS > .

6.1.2 Letters

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

NOTE – EXPRESS-I may be written using upper-, lower-, or mixed-case letters.

Syntax:

124 letter = < as EXPRESS > .

6.1.3 Special characters

The special characters (printable characters which are neither letters nor digits) are used mainly for punctuation and as operators. Some of the special characters shown are not used as part of the language. They may be used within remarks and string values, however. These special characters are in cells 21–2F, 3A–3F, 40, 5B–5E, 60, and 7B–7E of the EXPRESS-I character set.

Syntax:

134 special = < as EXPRESS > .

6.1.4 Underscore

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

6.1.5 Whitespace

Whitespace is defined by the following sub-clauses and by 6.1.6. Whitespace shall be used to separate the tokens in EXPRESS-I source.

NOTE – Liberal, and consistent, use of whitespace can improve the structure and readability of EXPRESS-I source.

6.1.5.1 Space character

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

6.1.5.2 Newline

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

The representation of a newline is implementation-defined.

6.1.6 Remarks

A remark is used for documentation and shall be interpreted by an EXPRESS-I parser as whitespace. There are two forms of remark: embedded remark and tail remark.

6.1.6.1 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:

```
142 embedded_remark = < as EXPRESS > .
```

Any character within the EXPRESS-I 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 2 – The following is an example of embedded nested remarks.

```
(* The '(' symbol starts an embedded remark, and the '*)' symbol ends it. *)
```

6.1.6.2 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:

```
144 tail_remark = < as EXPRESS > .
```

EXAMPLE 3 – A tail remark

```
-- This is a tail remark and is ended by a newline
```

6.2 Reserved words

The reserved words of EXPRESS-I 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-I are described below.

6.2.1 Keywords

EXPRESS-I uses a subset of the EXPRESS keywords, together with some additional ones.

Table 1 lists the keywords that are common to both EXPRESS-I and EXPRESS. Table 2 lists the additional EXPRESS-I keywords.

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

Table 1 – Keywords common to EXPRESS-I and EXPRESS

ABSTRACT	AGGREGATE	ALIAS	ARRAY
BAG	BEGIN	BINARY	BOOLEAN
BY	CASE	CONSTANT	CONTEXT
DERIVE	ELSE	END	END_ALIAS
END_CASE	END_CONSTANT	END_CONTEXT	END_ENTITY
END_FUNCTION	END_IF	END_LOCAL	END_MODEL
END_PROCEDURE	END_REPEAT	END_TYPE	ENTITY
ENUMERATION	ESCAPE	FIXED	FOR
FUNCTION	GENERIC	IF	INTEGER
INVERSE	LIST	LOCAL	LOGICAL
MODEL	NUMBER	OF	ONEOF
OPTIONAL	OTHERWISE	PROCEDURE	QUERY
REAL	REPEAT	RETURN	SELECT
SET	SKIP	STRING	SUBTYPE
SUPERTYPE	THEN	TO	TYPE
UNIQUE	UNTIL	VAR	WHERE
WHILE			

Table 2 – Additional EXPRESS-I keywords

CALL	CRITERIA	END_CALL	END_CRITERIA
END_NOTES	END_OBJECTIVE	END_PARAMETER	END_PURPOSE
END_REALIZATION	END_REFERENCES	END_SCHEMA_DATA	END_TEST_CASE
IMPORT	NOTES	OBJECTIVE	PARAMETER
PURPOSE	REALIZATION	REFERENCES	SCHEMA_DATA
SUBOF	SUPOF	TEST_CASE	USING
WITH			

6.2.2 Reserved words which are operators

The operators defined by reserved words are shown in table 3. These are the same as the EXPRESS operators and are defined in clause 12 of ISO 10303-11:1994.

Table 3 – The EXPRESS-I use of EXPRESS operators

AND	ANDOR	DIV	IN
LIKE	MOD	NOT	OR
XOR			

6.2.3 Built-in constants

The names of the EXPRESS-I built-in constants are given in table 4. These are the same as the EXPRESS constants and are defined in clause 14 of ISO 10303-11:1994.

Table 4 – The EXPRESS-I use of EXPRESS constants

?	CONST_E	FALSE	PI
SELF	TRUE	UNKNOWN	

The question mark character (?) represents the notion of a nil, or unspecified, value.

6.2.4 Built-in functions

The names of the EXPRESS functions that may be used within EXPRESS-I are given in table 5.

Table 5 – The EXPRESS-I use of EXPRESS functions

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

The definitions of these functions are given in clause 15 of ISO 10303-11:1994.

6.2.5 Built-in procedures

The names of the EXPRESS procedures that may be used within EXPRESS-I are given in table 6. The procedures are defined in clause 16 of ISO 10303-11:1994.

Table 6 – The EXPRESS-I use of EXPRESS procedures

INSERT	REMOVE
--------	--------

6.3 Symbols

Symbols are special characters or groups of special characters which have a special meaning in EXPRESS-I. Symbols are used in EXPRESS-I 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. The EXPRESS-I symbols are shown in table 7 and table 8.

Table 7 – Symbols common to EXPRESS-I and EXPRESS

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

Table 8 – Additional EXPRESS-I symbols

@	!	->	<-
==	"		

6.4 Identifiers and references

Identifiers are names given to the elements declared in an EXPRESS-I instantiation. An identifier shall not be the same as an EXPRESS-I or EXPRESS reserved word.

Syntax:

```

187 constant_id = < as EXPRESS > .
198 entity_id = < as EXPRESS > .
282 schema_id = < as EXPRESS > .
140 simple_id = < as EXPRESS > .
51i ComplexEntityInstanceId = SimpleEntityInstanceId '[' SupSubId ']' .
58i ContextId = simple_id .
69i EntityInstanceId = ComplexEntityInstanceId |
    SimpleEntityInstanceId .
73i EnumerationId = type_ref .
75i EnumerationInstanceId = simple_id .
92i ModelId = simple_id .
100i ParameterId = simple_id .
115i SelectId = type_ref .
117i SelectInstanceId = simple_id .
120i SimpleEntityInstanceId = simple_id .
122i SimpleInstanceId = simple_id .
125i SupSubId = digits .
129i TestCaseId = simple_id .
136i TypeId = type_ref .
138i TypeInstanceId = simple_id .

```

The first character of a simple identifier shall be a letter. The remaining characters, if any, may be any combination of letters, digits, and the underscore character. Identifiers shall not have any embedded white space.

The implementor of an EXPRESS-I language parser shall specify the maximum number of characters of an identifier which can be read by that implementation (see annex B).

NOTE – The letters used to form identifiers are not case sensitive as upper- and lower-case letters are treated as equal.

EXAMPLE 4 – Valid simple identifiers

```
POINT line Circle AnEntity item507 An_integer
```

EXAMPLE 5 – Invalid simple identifiers

```

_POINT      underscore cannot be first character
line?       ? cannot be part of identifier
3dThing     digit cannot be first character
Pi          Pi is an EXPRESS-I keyword

```

EXAMPLE 6 – Valid complex entity instance identifiers

```
complex[101] complex[12] an_ent[23] an_ent[77]
```

Syntax:

```

146 constant_ref = < as EXPRESS > .
154 type_ref = < as EXPRESS > .
36i ContextRef = ContextId .
39i ParameterRef = ParameterId .

```

An element may be referenced via its identifier. Constant and parameter elements are referenced via the corresponding identifier.

Syntax:

34i ComplexEntityInstanceRef = '@' SimpleEntityInstanceId .
37i EntityInstanceRef = ComplexEntityInstanceRef |
SimpleEntityInstanceRef .
38i EnumerationInstanceRef = '@' EnumerationInstanceId .
96i ObjectInstanceRef = EntityInstanceRef | EnumerationInstanceRef |
SelectInstanceRef | SimpleInstanceRef |
TypeInstanceRef .
40i SelectInstanceRef = '@' SelectInstanceId .
41i SimpleEntityInstanceRef = '@' SimpleEntityInstanceId .
42i SimpleInstanceRef = '@' SimpleInstanceId .
43i SupSubRef = '@' SubSubId .
44i TypeInstanceRef = '@' TypeInstanceId .

The first character of an entity, enumeration, type or select instance reference shall be @ followed by at least one character. The characters after the initial @ can be any combination of letters, digits, and the underscore character which form a valid entity, enumeration, simple instance, select or type instance identifier. Collectively, these are termed object instance references.

EXAMPLE 7 – Valid object instance references

@POINT @line @Circle @AnEntity @item567

EXAMPLE 8 – Invalid object instance references

@line? ? cannot be part of identifier
3dThing @ must be first character
@subof subof is an EXPRESS-I keyword
@@Circle @ must appear only as the first character
@567 characters following the @ must begin with a letter
@complex[82] only alphanumeric and the underscore allowed

7 Named domains

This clause defines the domain types provided as part of the language. Domains are used to delineate the allowable instance values. A named domain is an entity, a type, an enumeration, or a select domain.

7.1 Entity domain

An entity domain represents a class of objects which have common attributes.

Syntax:

```
66i EntityDomain = [ SchemaId '.' ] EntityId .
```

NOTE – An entity domain corresponds to an EXPRESS ENTITY data type.

7.2 Enumeration domain

An enumeration domain has as its domain an ordered set of names.

Syntax:

```
72i EnumerationDomain = [ SchemaId '.' ] EnumerationId .
```

NOTE – An enumeration domain corresponds to an EXPRESS ENUMERATION data type.

7.3 Select domain

A select domain has as its domain a union of domains.

Syntax:

```
114i SelectDomain = [ SchemaId '.' ] SelectId .
```

NOTE – A select domain corresponds to an EXPRESS SELECT data type.

7.4 Type domain

A type domain is an extension to the other domains in the language.

Syntax:

```
135i TypeDomain = [ SchemaId '.' ] TypeId .
```

NOTE – A type domain corresponds to an EXPRESS defined data TYPE which is neither an ENUMERATION nor a SELECT.

8 Values and instances

This clause describes the EXPRESS instantiation capabilities.

8.1 Base values

Syntax:

```
48i BaseValue = SimpleValue | EnumerationValue .
123i SimpleValue = BinaryValue | BooleanValue | LogicalValue |
                NumberValue | StringValue .
```

A simple value is a self-defining constant value. The domain of the value depends on how characters are composed to form a token.

8.1.1 Binary value

A binary value represents a value of a binary domain.

Syntax:

```
25i BinaryValue = binary_literal .
136 binary_literal = < as EXPRESS > .
```

A binary value is composed of the % character followed by one or more bits (0 or 1).

The implementor of an EXPRESS language parser shall specify the maximum number of bits in a binary value which can be read by that implementation (see annex B).

EXAMPLE 9 – A valid binary value

```
%10100110000101
```

8.1.2 Boolean value

A boolean value represents a value of a boolean domain.

Syntax:

```
50i BooleanValue = TRUE | FALSE .
```

A boolean value is one of the built-in constants FALSE or TRUE.

8.1.3 Number value

A number value is either an integer value or a real value.

Syntax:

```
94i NumberValue = IntegerValue | RealValue .
```

8.1.4 Integer value

An integer value represents a value of an integer domain.

Syntax:

```
29i IntegerValue = [ sign ] integer_literal .  
138 integer_literal = < as EXPRESS > .  
286 sign = < as EXPRESS > .
```

An integer literal is composed entirely of digits. An integer value is composed of an integer literal, optionally preceded by a sign. It defines a positive, negative or zero integer (whole) number.

The implementor of an EXPRESS language parser shall specify the maximum value of an integer value which can be read by that implementation (see annex B).

EXAMPLE 10 – Valid integer values

```
0 1 -1 891562934527619
```

EXAMPLE 11 – Invalid integer values

```
1.0 cannot include a decimal point
```

8.1.5 Logical value

A logical value represents a value of a logical domain.

Syntax:

```
88i LogicalValue = logical_literal .  
242 logical_literal = < as EXPRESS > .
```

A logical value is one of the built-in constants FALSE, TRUE, or UNKNOWN.

8.1.6 Real value

A real value represents a value of a real domain.

A real value is either a signed mathematical constant or a signed real literal.

Syntax:

```

104i RealValue = SignedMathConstant | SignedRealLiteral .
31i SignedMathConstant = [ sign ] MathConstant .
89i MathConstant = CONST_E | PI .
32i SignedRealLiteral = [ sign ] real_literal .
139 real_literal = < as EXPRESS > .

```

A signed mathematical constant is one of the built-in mathematical constants (i.e e or π) optionally preceded by a sign.

The mathematical constant $e = 2.7182\dots$ is represented by the EXPRESS constant CONST_E.

The mathematical constant $\pi = 3.1415\dots$ is represented by the EXPRESS constant PI.

EXAMPLE 12 – Signed mathematical constants

```
-const_e      Pi
```

A signed real literal is composed of a (signed) mantissa and an optional exponent. It defines a rational number.

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

EXAMPLE 13 – Valid real values

```
0.0  -1.E6  1.e-6  8915629.34527619
```

EXAMPLE 14 – Invalid real values

.001	must have at least one digit before the point
1e10	must have a decimal point in the mantissa
1.0e-12.0	cannot have a decimal point in the exponent
CONSTE	misspelled built-in constant

8.1.7 String value

A string value represents a value of a string domain. There are two forms of string value, the explicit string value and encoded string value. An explicit string value is composed of a sequence of characters in the EXPRESS-I character set enclosed by apostrophes ('). An apostrophe within an explicit string value is represented by two consecutive apostrophes. An encoded string value is a four octet encoded representation of a sequence of characters in ISO/IEC 10646-1 enclosed in quotation marks ("). The encoding is defined as follows:

- first octet = ISO/IEC 10646-1 group in which the character is defined;
- second octet = ISO/IEC 10646-1 plane in which the character is defined;
- third octet = ISO/IEC 10646-1 row in which the character is defined;
- fourth octet = ISO/IEC 10646-1 cell in which the character is defined.

The sequence of octets shall identify one of the valid characters of ISO/IEC 10646-1.

Syntax:

```

124i StringValue = SimpleStringValue | EncodedStringValue .
33i SimpleStringValue = \q { ( \q \q ) | not_quote | \s | \o | \n } \q .
130 not_quote = < as EXPRESS > .
27i EncodedStringValue = ''' { encoded_character | \n } ''' .
122 encoded_character = < as EXPRESS > .
    
```

The implementor of an EXPRESS language parser shall specify the maximum number of characters of a string value which can be read by that implementation (see annex B).

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 value which can be read by that implementation (see annex B).

NOTE – An EXPRESS string value differs from an EXPRESS string literal, as in the former case a string value may span more than one physical line, whereas an EXPRESS string literal cannot span more than one physical line.

EXAMPLE 15 – Valid explicit string values

```
'This is a string on one line.'
```

Reads ... This is a string on one line.

```
'This
    is
      a
        multiline
          string.'
```

```

This
is
Reads ... a
          multiline
            string.
    
```

```
'This string''s got a single apostrophe embedded in it.'
```

Reads ... This string's got a single apostrophe embedded in it.

EXAMPLE 16 – Invalid explicit string values

```
'This string is invalid because there is no closing apostrophe.
```

EXAMPLE 17 – Valid encoded string values

```
"00000041"
```

Reads ... A.

```
"000000C5"
```

Reads ... Å

EXAMPLE 18 – Invalid encoded string values

"000041"

Octets must be supplied in groups of four

"00000041 000000C5"

Cannot have a space between octets

8.1.8 Enumeration value

An enumeration value represents a value of an enumeration domain.

Syntax:

```
28i EnumerationValue = '!' simple_id .
```

An enumeration value is a simple identifier prepended with an exclamation mark (!). A simple identifier is a character sequence of letters, digits and underscore, with the first character being a letter.

EXAMPLE 19 – Valid enumeration values

!red !green !forward

8.2 Aggregation values

EXPRESS distinguishes two forms of aggregation of values — fixed and dynamic. A fixed aggregation is an aggregation of like things, where the number of storage locations is independent of the number of elements actually stored in the aggregation. A dynamic aggregation is an aggregation of like things, where the number of storage locations is dependent upon the number of elements actually stored in the aggregation. Aggregation values may be nested.

Syntax:

```
46i AggregationValue = DynamicAggr | FixedAggr .
61i DynamicAggr = '(' [ DynamicList ] ')' .
63i DynamicList = DynamicMember { ',' DynamicMember } .
64i DynamicMember = AggregationValue | ConstantValue |
                    DerattValue | ParmValue | ReqattValue |
                    TypeValue .
79i FixedAggr = '[' FixedList ']' .
80i FixedList = FixedMember { ',' FixedMember } .
81i FixedMember = DynamicMember | Nil .
```

The allowable domains of the elements within the aggregation depend on the domain context. These contexts are:

- Constants (see 8.8);
- Derived attributes (see 8.7.1.2);
- Explicit attributes (see 8.7.1.1);
- Parameters (see 9.2.2);
- Defined data types (see 8.4).

Rules and restrictions:

- a) Elements within a dynamic aggregation shall not be Nil.
- b) Elements within a fixed aggregation may be Nil.
- c) The element values within an aggregation shall be compatible with the aggregation domain.

EXAMPLE 20 – Aggregation values

(10,-10,0)	a dynamic aggregation of 3 integer values
(1,1,2,2,3,3)	a dynamic aggregation of 6 integer values
()	an empty dynamic aggregation
[1,2,3,4]	a fixed aggregation of 4 integer values
([1,2],[3,?])	a dynamic aggregation of a fixed aggregation of 2 values

8.3 Simple instance

A simple instance is a representation of the value of one instance of a simple value.

Syntax:

```

121i SimpleInstance = SimpleInstanceId '=' SimpleValue ';' .
122i SimpleInstanceId = simple_id .
123i SimpleValue = BinaryValue | BooleanValue | LogicalValue |
                  NumberValue | StringValue .
42i SimpleInstanceRef = '@' SimpleInstanceId .
    
```

EXAMPLE 21 – Some simple instances

```

r1 = 27.0;
s1 = 'A string';
    
```

8.4 Type instance

A type instance is a representation of the value of one instance of a TYPE domain.

Syntax:

```

137i TypeInstance = TypeInstanceId '=' TypeInstanceValue ';' .
138i TypeInstanceId = simple_id .
139i TypeInstanceValue = TypeDomain '{' TypeValue '}' .
140i TypeValue = AggregationValue | BaseValue | ConstantRef |
                EntityInstanceValue | NamedInstanceValue |
                ObjectInstanceRef | ParameterRef .
44i TypeInstanceRef = '@' TypeInstanceId .

```

Rules and restrictions:

- a) The value of the instance shall be either a simple value, an entity instance reference, a type instance reference, or aggregations of these.

EXAMPLE 22 – Some type instances

```

t1 = a_real{27.0};
t2 = an_array_of_string[['one', 'two']];
t3 = a_dynamic_aggregate_of_integer{(1,1,2,3,5,8,13)};

```

8.5 Select instance

A select instance is a representation of the value of one instance of a SELECT domain.

Syntax:

```

116i SelectInstance = SelectInstanceId '=' SelectInstanceValue ';' .
117i SelectInstanceId = simple_id .
118i SelectInstanceValue = SelectDomain '{' SelectValue '}' .
119i SelectValue = EnumerationValue | NamedInstanceValue |
                ObjectInstanceRef | TypeValue .
40i SelectInstanceRef = '@' SelectInstanceId .

```

Rules and restrictions:

- a) The value of the instance shall be either a type instance reference, a select instance reference, an enumeration instance reference, or an entity instance reference.

EXAMPLE 23 – A select instance

```

s1 = type_or_entity{@e27};

```

8.6 Enumeration instance

An enumeration instance is a representation of the value of one instance of an ENUMERATION domain.

Syntax:

```

74i EnumerationInstance = EnumerationInstanceId '='
                        EnumerationInstanceValue ';' .
75i EnumerationInstanceId = simple_id .
76i EnumerationInstanceValue = EnumerationDomain
                        '{' EnumerationValue '}' .
28i EnumerationValue = '!' simple_id .
38i EnumerationInstanceRef = '@' EnumerationInstanceId .

```

Rules and restrictions:

- a) The value of the instance shall be an enumeration value.

EXAMPLE 24 – Some enumeration instances

```

enum1 = an_enum{!first};
enum2 = an_enum{!second};

```

8.7 Entity instance

An entity instance is a representation of one instantiation of an ENTITY domain.

Syntax:

```

68i EntityInstance = EntityInstanceId '=' EntityInstanceValue ';' .
69i EntityInstanceId = ComplexEntityInstanceId |
                        SimpleEntityInstanceId .
70i EntityInstanceValue = EntityDomain '{' [ InheritsFrom ]
                        { ExplicitAttr } { DerivedAttr }
                        { InverseAttr } [ BequeathesTo ] '}' .
37i EntityInstanceRef = ComplexEntityInstanceRef |
                        SimpleEntityInstanceRef .

```

EXPRESS distinguishes two forms of an entity instance:

Simple entity instance: The instance is not part of an inheritance tree.

Complex entity instance: The instance is of an inheritance tree. It is composed of component (entity) instances which together form all the nodes of the tree.

Syntax:

```

51i ComplexEntityInstanceId = SimpleEntityInstanceId '[' SupSubId ']' .
34i ComplexEntityInstanceRef = '@' SimpleEntityInstanceId .
120i SimpleEntityInstanceId = simple_id .
41i SimpleEntityInstanceRef = '@' SimpleEntityInstanceId .
125i SupSubId = digits .

```

The identifier of a simple entity instance is a simple identifier.

The identifier of a complex entity instance has two parts. The first part is the same as the simple entity instance identifier. The second part is a string of digits enclosed in square brackets. The