INTERNATIONAL STANDARD

**ISO 10303-513**

First edition
2000-09-01

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

## Part 513: Application interpreted construct: Elementary boundary representation

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

*Partie 513: Construction interprétée d'application: Représentation des limites élémentaires*

Reference number
ISO 10303-513:2000(E)

**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 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.ch
Web www.iso.ch

Printed in Switzerland

# Contents

Page



**Figures**



**Tables**



# Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

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

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

International Standard ISO 10303-513 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data*.

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

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

This part of ISO 10303 is a member of the application interpreted constructs series.

Annexes A and B form a normative part of this part of ISO 10303. Annexes C, D and E 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 application interpreted construct series.

An application interpreted construct (AIC) provides a logical grouping of interpreted constructs that supports a specific functionality for the usage of product data across multiple application contexts. An interpreted construct is a common interpretation of the integrated resources that supports shared information requirements among application protocols.

This document specifies the application interpreted construct for the definition of a boundary representation solid with elementary geometry and explicit topology.

# Industrial automation systems and integration — Product data representation and exchange — Part 513 : Application interpreted construct: Elementary boundary representation

## 1 Scope

This part of ISO 10303 specifies the interpretation of the generic resources for the definition of an elementary boundary representation model.
The following are within the scope of this part of ISO 10303:

— the definition of an **elementary_brep_shape_representation**, this is a representation composed of one or more **manifold_solid_brep**s each of which is defined with elementary geometry and complete explicit topology;

— the definition of the unbounded geometry of curves and surfaces used in the definition of the faces of such a B-rep model;

— the definition of the topological structure of a B-rep model;

— 3D geometry;

— B-reps;

— elementary curves, these are **line**s or **conic**s;

— **elementary_surface**s;

— geometric transformations;

— **polyline**s;

— unbounded geometry;

— use of topology to bound geometric entities.

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

— 2D geometry;

— bounded curves other than **polyline**s;

— bounded surfaces;

— offset curves and surfaces.

This AIC is independent of any industrial application domain.

## 2 Normative references

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

ISO/IEC 8824-1: 1995, *Information technology - 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/TR 10303-12: 1997, *Industrial automation systems and integration - Product data representation and exchange - Part 12 : Description methods: The EXPRESS-I language reference manual.*

ISO 10303-41: 1994, *Industrial automation systems and integration - Product data representation and exchange - Part 41 : Integrated generic resources: Fundamentals of product description and support.*

ISO 10303-42: 1994, *Industrial automation systems and integration - Product data representation and exchange - Part 42 : Integrated generic resources: Geometric and topological representation.*

ISO 10303-43: 1994, *Industrial automation systems and integration - Product data representation and exchange - Part 43 : Integrated generic resources: Representation structures.*

ISO 10303-202: 1996, *Industrial automation systems and integration - Product data representation and exchange - Part 202: Application protocol: Associative draughting.*

ISO 10303-514: 1999, *Industrial automation systems and integration - Product data representation and exchange - Part 514: Application interpreted construct: Advanced boundary representation.*

# 3 Terms, definitions, and abbreviations

## 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.

— application;

— application context;

— application protocol;

— implementation method;

— integrated resource;

— interpretation;

— product data.

## 3.2 Terms defined in ISO 10303-42

For the purposes of this part of ISO 10303, the following terms defined in ISO 10303-42 apply.

— arcwise connected;

— boundary;

— bounds;

— coordinate space;

— curve;

— open curve;

— orientable;

— surface;

— topological sense.

## 3.3 Terms defined in ISO 10303-202

For the purposes of this part of ISO 10303, the following terms defined in ISO 10303-202 apply.

— AIC.

## 3.4 Terms defined in ISO 10303-514

For the purposes of this part of ISO 10303, the following terms defined in ISO 10303-514 apply.

— manifold solid.

## 3.5 Other definitions

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

**3.5.1**
**elementary B-rep shape representation**
a shape representation made up of one or more manifold solid B-reps. Each constituent B-rep is required to have its faces and edges defined by elementary geometry.

**3.5.2**
**elementary geometry**
geometry composed of **line**s, **polyline**s, **conic**s and **elementary_surface**s.

## 3.6 Abbreviations

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

| | |
|---|---|
| AIC | application interpreted construct |
| AP | application protocol |
| B-rep | boundary representation solid model |

# 4 EXPRESS short listing

This clause specifies the EXPRESS schema that uses elements from the integrated resources and contains the types, entity specializations, and functions that are specific to this part of ISO 10303.

NOTE 1 There may be subtypes and items of select lists that appear in the integrated resources that are not imported into the AIC. Constructs are eliminated from the subtype tree or select list through the use of the implicit interface rules of ISO 10303-11. References to eliminated constructs are outside the scope of the AIC. In some cases, all items of the select list are eliminated. Because AICs are intended to be implemented in the context of an application protocol, the items of the select list will be defined by the scope of the application protocol.

This application interpreted construct provides a consistent set of geometric and topological entities for the definition of manifold solid models with faces having elementary geometry and explicitly defined edges and vertices. The faces of the B-rep models are bounded by polylines, lines or conics.

The highest level entity in this AIC is the **elementary_brep_shape_representation.** This is a **shape_-representation** (see: ISO 10303-41) consisting of **manifold_solid_brep**s and **mapped_item**s defined as translated or transformed copies of **manifold_solid_brep**s having elementary geometry.

EXPRESS specification:

```
*)
SCHEMA aic_elementary_brep;
 USE FROM geometry_schema(axis2_placement_3d,
                          cartesian_point,
                          cartesian_transformation_operator_3d,
                          circle,
                          conical_surface,
                          cylindrical_surface,
                          degenerate_toroidal_surface,
                          direction,
                          ellipse,
                          hyperbola,
                          line,
                          parabola,
                          plane,
                          polyline,
                          spherical_surface,
                          toroidal_surface,
                          vector);
 USE FROM geometric_model_schema(manifold_solid_brep,
                                 brep_with_voids);

REFERENCE FROM geometric_model_schema(msb_shells);

USE FROM topology_schema(closed_shell,
                          connected_face_set,
                          edge_curve,
                          edge_loop,
                          face_bound,
                          face_outer_bound,
                          face_surface,
                          oriented_closed_shell,
                          vertex_loop,
                          vertex_point);


 USE FROM representation_schema(mapped_item);

 USE FROM product_property_representation_schema(shape_representation);

(*
```

NOTE 2 The **connected_face_set** entity is explicitly interfaced (i.e. included in the USE FROM lists) to allow rules in the **elementary_brep_shape_representation** entity to access attributes of this entity. For the use of this AIC this entity shall only be instantiated as one of its subtypes.

NOTE 3 The schemas referenced above can be found in the following parts of ISO 10303:

| | |
|---|---|
| `geometry_schema` | ISO 10303-42 |
| `geometric_model_schema` | ISO 10303-42 |
| `topology_schema` | ISO 10303-42 |
| `representation_schema` | ISO 10303-43 |
| `product_property_representation_schema` | ISO 10303-41 |

## 4.1 Fundamental concepts and assumptions

The following entities are intended to be independently instantiated in the application protocol schemas that use this AIC:

— axis2_placement_3d;

— brep_with_voids;

— cartesian_point;

— cartesian_transformation_operator_3d;

— circle;

— closed_shell;

— conical_surface;

— cylindrical_surface;

— degenerate_toroidal_surface;

— direction;

— edge_curve;

— edge_loop;

— elementary_face;

— ellipse;

— face_bound;

— face_outer_bound;

— face_surface;

— hyperbola;

— line;

— manifold_solid_brep;

— mapped_item;

— oriented_closed_shell;

— parabola;

— plane;

— polyline;

— representation_map;

— spherical_surface;

— toroidal_surface;

— vector;

— vertex_loop;

— vertex_point.

An application protocol that uses this AIC shall require that all the above entities are supported.

An application protocol that uses this AIC shall permit the **shape_representation** entity to be instantiated as an **elementary_brep_shape_representation**.

## 4.2 aic_elementary_brep schema entity definition: elementary_brep_shape_representation

The **elementary_brep_shape_representation** is a type of **shape_representation** in which the representation items are specialisations of **manifold_solid_brep** entities. These differ from the more general B-rep in having only explicit geometric forms for their faces and edges. The face geometry is restricted to **elementary_surface**s, and the edge curves to **line**s, **polyline**s or **conic**s.

EXPRESS specification:

```
*)
ENTITY elementary_brep_shape_representation
SUBTYPE OF (shape_representation);
WHERE
  WR1 : SIZEOF (QUERY (it <* SELF.items |
         NOT (SIZEOF (['AIC_ELEMENTARY_BREP.MANIFOLD_SOLID_BREP',
               'AIC_ELEMENTARY_BREP.FACETED_BREP',
               'AIC_ELEMENTARY_BREP.MAPPED_ITEM',
               'AIC_ELEMENTARY_BREP.AXIS2_PLACEMENT_3D'] *
                  TYPEOF(it)) = 1))) = 0;
  WR2 : SIZEOF (QUERY (it <* SELF.items |
        SIZEOF(['AIC_ELEMENTARY_BREP.MANIFOLD_SOLID_BREP',
        'AIC_ELEMENTARY_BREP.MAPPED_ITEM'] * TYPEOF(it)) =1 )) > 0;
  WR3 : SIZEOF (QUERY (msb <* QUERY (it <* SELF.items |
          'AIC_ELEMENTARY_BREP.MANIFOLD_SOLID_BREP' IN TYPEOF(it)) |
          NOT (SIZEOF (QUERY (csh <* msb_shells(msb) |
           NOT (SIZEOF (QUERY(fcs <* csh.cfs_faces |
             NOT('AIC_ELEMENTARY_BREP.FACE_SURFACE' IN TYPEOF(fcs)))) = 0
                ))) = 0
                  ))) = 0;
  WR4 : SIZEOF (QUERY (msb <* QUERY (it <* SELF.items |
          'AIC_ELEMENTARY_BREP.MANIFOLD_SOLID_BREP' IN TYPEOF(it)) |
          NOT (SIZEOF (QUERY (csh <* msb_shells(msb) |
           NOT (SIZEOF (QUERY(fcs <* csh\connected_face_set.cfs_faces |
             NOT(('AIC_ELEMENTARY_BREP.ELEMENTARY_SURFACE' IN
                  TYPEOF(fcs\face_surface.face_geometry))
           ))) = 0
                ))) = 0
                  ))) = 0;
  WR5 : SIZEOF (QUERY (msb <* QUERY (it <* SELF.items |
          'AIC_ELEMENTARY_BREP.MANIFOLD_SOLID_BREP' IN TYPEOF(it)) |
          NOT (SIZEOF (QUERY (csh <* msb_shells(msb) |
            NOT (SIZEOF (QUERY(fcs <* csh\connected_face_set.cfs_faces |
              NOT (SIZEOF(QUERY (elp_fbnds <* QUERY (bnds <* fcs.bounds |
        'AIC_ELEMENTARY_BREP.EDGE_LOOP' IN TYPEOF(bnds.bound)) |
               NOT (SIZEOF (QUERY (oe <* elp_fbnds.bound\path.edge_list |
                 NOT('AIC_ELEMENTARY_BREP.EDGE_CURVE' IN
           TYPEOF(oe.edge_element)))) = 0
                  ))) = 0
                ))) = 0
              ))) = 0
            ))) = 0;
  WR6 : SIZEOF (QUERY (msb <* QUERY (it <* SELF.items |
          'AIC_ELEMENTARY_BREP.MANIFOLD_SOLID_BREP' IN TYPEOF(it)) |
          NOT (SIZEOF (QUERY (csh <* msb_shells(msb) |
            NOT (SIZEOF (QUERY(fcs <* csh\connected_face_set.cfs_faces |
              NOT (SIZEOF(QUERY (elp_fbnds <* QUERY (bnds <* fcs.bounds |
        'AIC_ELEMENTARY_BREP.EDGE_LOOP' IN TYPEOF(bnds.bound)) |
               NOT (SIZEOF (QUERY (oe <* elp_fbnds.bound\path.edge_list |
```

```
                NOT (SIZEOF (['AIC_ELEMENTARY_BREP.LINE',
                      'AIC_ELEMENTARY_BREP.CONIC',
                      'AIC_ELEMENTARY_BREP.POLYLINE'] *
          TYPEOF(oe.edge_element\edge_curve.edge_geometry)) = 1 )
             )) = 0
             ))) = 0
               ))) = 0
                 ))) = 0
                  ))) = 0;
WR7 : SIZEOF (QUERY (msb <* QUERY (it <* SELF.items |
        'AIC_ELEMENTARY_BREP.MANIFOLD_SOLID_BREP' IN TYPEOF(it)) |
        NOT (SIZEOF (QUERY (csh <* msb_shells(msb) |
          NOT (SIZEOF (QUERY(fcs <* csh\connected_face_set.cfs_faces |
            NOT (SIZEOF(QUERY (elp_fbnds <* QUERY (bnds <* fcs.bounds |
      'AIC_ELEMENTARY_BREP.EDGE_LOOP' IN TYPEOF(bnds.bound)) |
             NOT (SIZEOF (QUERY (oe <* elp_fbnds.bound\path.edge_list |
         NOT(('AIC_ELEMENTARY_BREP.VERTEX_POINT' IN TYPEOF(oe.edge_start))
             AND ('AIC_ELEMENTARY_BREP.VERTEX_POINT' IN
              TYPEOF(oe.edge_end))
          ))) = 0
            ))) = 0
             ))) = 0
               ))) = 0
                 ))) = 0;
WR8 : SIZEOF (QUERY (msb <* QUERY (it <* SELF.items |
        'AIC_ELEMENTARY_BREP.MANIFOLD_SOLID_BREP' IN TYPEOF(it)) |
        NOT (SIZEOF (QUERY (csh <* msb_shells(msb) |
          NOT (SIZEOF (QUERY(fcs <* csh\connected_face_set.cfs_faces |
            NOT (SIZEOF(QUERY (elp_fbnds <* QUERY (bnds <* fcs.bounds |
      'AIC_ELEMENTARY_BREP.EDGE_LOOP' IN TYPEOF(bnds.bound)) |
             NOT (SIZEOF (QUERY (oe <* elp_fbnds.bound\path.edge_list |
                ('AIC_ELEMENTARY_BREP.POLYLINE' IN
          TYPEOF(oe.edge_element\edge_curve.edge_geometry)) AND
          (NOT (SIZEOF (oe\oriented_edge.edge_element\
                edge_curve.edge_geometry\polyline.points) >= 3))
             )) = 0
             ))) = 0
               ))) = 0
                 ))) = 0
                  ))) = 0;
WR9 : SIZEOF (QUERY (msb <* QUERY (it <* items |
       'AIC_ELEMENTARY_BREP.MANIFOLD_SOLID_BREP' IN TYPEOF(it)) |
       'AIC_ELEMENTARY_BREP.ORIENTED_CLOSED_SHELL' IN TYPEOF
           (msb\manifold_solid_brep.outer)))
         = 0;
WR10 : SIZEOF (QUERY (brv <* QUERY (it <* items |
       'AIC_ELEMENTARY_BREP.BREP_WITH_VOIDS' IN TYPEOF(it)) |
       NOT (SIZEOF (QUERY (csh <* brv\brep_with_voids.voids |
        csh\oriented_closed_shell.orientation)) = 0))) = 0;
WR11 : SIZEOF (QUERY (mi <*  QUERY (it <* items |
              'AIC_ELEMENTARY_BREP.MAPPED_ITEM' IN TYPEOF(it)) |
       NOT ('AIC_ELEMENTARY_BREP.ELEMENTARY_BREP_SHAPE_REPRESENTATION' IN
```

```
            TYPEOF(mi\mapped_item.mapping_source.
                          mapped_representation)))) = 0;
  WR12 : SIZEOF (QUERY (msb <* QUERY (it <* SELF.items |
           'AIC_ELEMENTARY_BREP.MANIFOLD_SOLID_BREP' IN TYPEOF(it)) |
           NOT (SIZEOF (QUERY (csh <* msb_shells(msb) |
             NOT (SIZEOF (QUERY(fcs <* csh\connected_face_set.cfs_faces |
              NOT (SIZEOF(QUERY (vlp_fbnds <* QUERY (bnds <* fcs.bounds |
       'AIC_ELEMENTARY_BREP.VERTEX_LOOP' IN TYPEOF(bnds.bound)) |
         NOT(('AIC_ELEMENTARY_BREP.VERTEX_POINT' IN
             TYPEOF(vlp_fbnds\face_bound.bound\vertex_loop.loop_vertex)) AND
              ('AIC_ELEMENTARY_BREP.CARTESIAN_POINT' IN
                 TYPEOF(vlp_fbnds\face_bound.bound\vertex_loop.
                   loop_vertex\vertex_point.vertex_geometry))
           ))) = 0))) = 0))) = 0))) =0;
 END_ENTITY;
(*
```

Formal propositions:

**WR1:** The **items** attribute of the **representation** supertype shall contain **manifold_solid_brep**s, **mapped_item**s and **axis2_placement_3d**s only. The use of **faceted_brep**s is excluded by this rule since an instance of **faceted_brep** would also be of type **manifold_solid_brep**.

**WR2:** At least one item in the **items** set shall be a **manifold_solid_brep** entity or a **mapped_item** (see also WR11).

**WR3:** All faces used in constructing a **manifold_solid_brep** shall be of type **face_surface**.

NOTE 1 The call to function **msb_shells** in WR3, and later rules, is correct since, although the generic type of the argument 'msb' is **representation_item**, 'msb' has been selected by QUERY to be of type **manifold_solid_brep**.

**WR4:** For each **manifold_solid_brep** in the **items** set, the associated surface for each face shall be an **elementary_surface**.

**WR5:** For each **manifold_solid_brep** in the **items** set, the edges used to define the boundaries shall all be of type **edge_curve**.

**WR6:** For each **manifold_solid_brep** in the **items** set, each curve used to define the face bounds shall be either a **conic**, a **line**, or a **polyline**.

**WR7:** For each **manifold_solid_brep** in the **items** set, the edges used to define the boundaries shall all be trimmed by vertices of type **vertex_point**.

**WR8:** For each **manifold_solid_brep** in the **items** set, each **polyline** used to define part of the face bounds shall contain 3 or more points.

**WR9:** For each **manifold_solid_brep** in the **items** set, the outer shell attribute shall not be of type **oriented_closed_shell**.

**WR10:** If a **brep_with_voids** is included in the **items** set, each shell in the **voids** set shall be an **oriented_closed_shell** with orientation value FALSE.

**WR11:** If a **mapped_item** is included in the **items** set, the **mapped_representation** of the **mapping_source** attribute shall be an **elementary_brep_shape_representation**.

NOTE If a **cartesian_transformation_operator_3d** is included as **mapped_item.mapping_target** with an **axis2_placement_3d** that corresponds to the original coordinate system as **mapped_representation.mapping_origin**, the resulting **mapped_item** is a transformed copy of the **elementary_brep_shape_representation**. The precise definition of the transformation, including translation, rotation, scaling and, if appropriate, mirroring, is given by the transformation operator.

**WR12:** For each **manifold_solid_brep** in the **items** set, any **vertex_loop** used to define a face bound shall reference a **vertex_point** with the geometry defined by a **cartesian_point**.

EXPRESS specification:

```
*)
  END_SCHEMA; -- end AIC_ELEMENTARY_BREP SCHEMA
(*
```

# Annex A
(normative)

# Short names of entities

Table A.1 provides the short names of entities specified in this part of ISO 10303. Requirements on the use of the short names are found in the implementation methods included in ISO 10303.

## Table A.1 – Short names of entities

| Entity name | Short name |
|---|---|
| ELEMENTARY_BREP_SHAPE_REPRESENTATION | EBSR |

# Annex B
(normative)

# Information object registration

## B.1 Document identification

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

{ iso standard 10303 part(513) version(1) }

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

## B.2 Schema identification

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

{ iso standard 10303 part(513) version(1) object(1) aic-elementary-brep(1) }

is assigned to the aic_elementary_brep schema (see clause 4). The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

# Annex C
(informative)

# Computer-interpretable listings

This annex provides a listing of the EXPRESS entity names and corresponding short names as specified in this Part of ISO 10303 without comments or other explanatory text. This annex is available in computer-interpretable form and can be found at the following URLs:

Short names: http://www.mel.nist.gov/div826/subject/apde/snr/
EXPRESS: http://www.mel.nist.gov/step/parts/part513/is/

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

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

# Annex D
(informative)

# EXPRESS-G diagrams

Figures D.1 through D.4 correspond to the EXPRESS generated from the short listing given in clause 4 using the interface specifications of ISO 10303-11. The diagrams use the EXPRESS-G graphical notation for the EXPRESS language. EXPRESS-G is defined in annex D of ISO 10303-11.

NOTE 1 The following select types are interfaced into the AIC expanded listing according to the implicit interface rules of ISO 10303-11. These select types are not used by other entities in this part of ISO 10303.

— geometric_set_select;

— pcurve_or_surface;

— reversible_topology;

— shell;

— trimming_select;

— vector_or_direction.

NOTE 2 The implicit interface rules of ISO 10303-11 also introduce some entities whose instantiation is prohibited by rules on the **elementary_brep_shape_representation**. These entities are marked "*" in the EXPRESS-G diagrams.

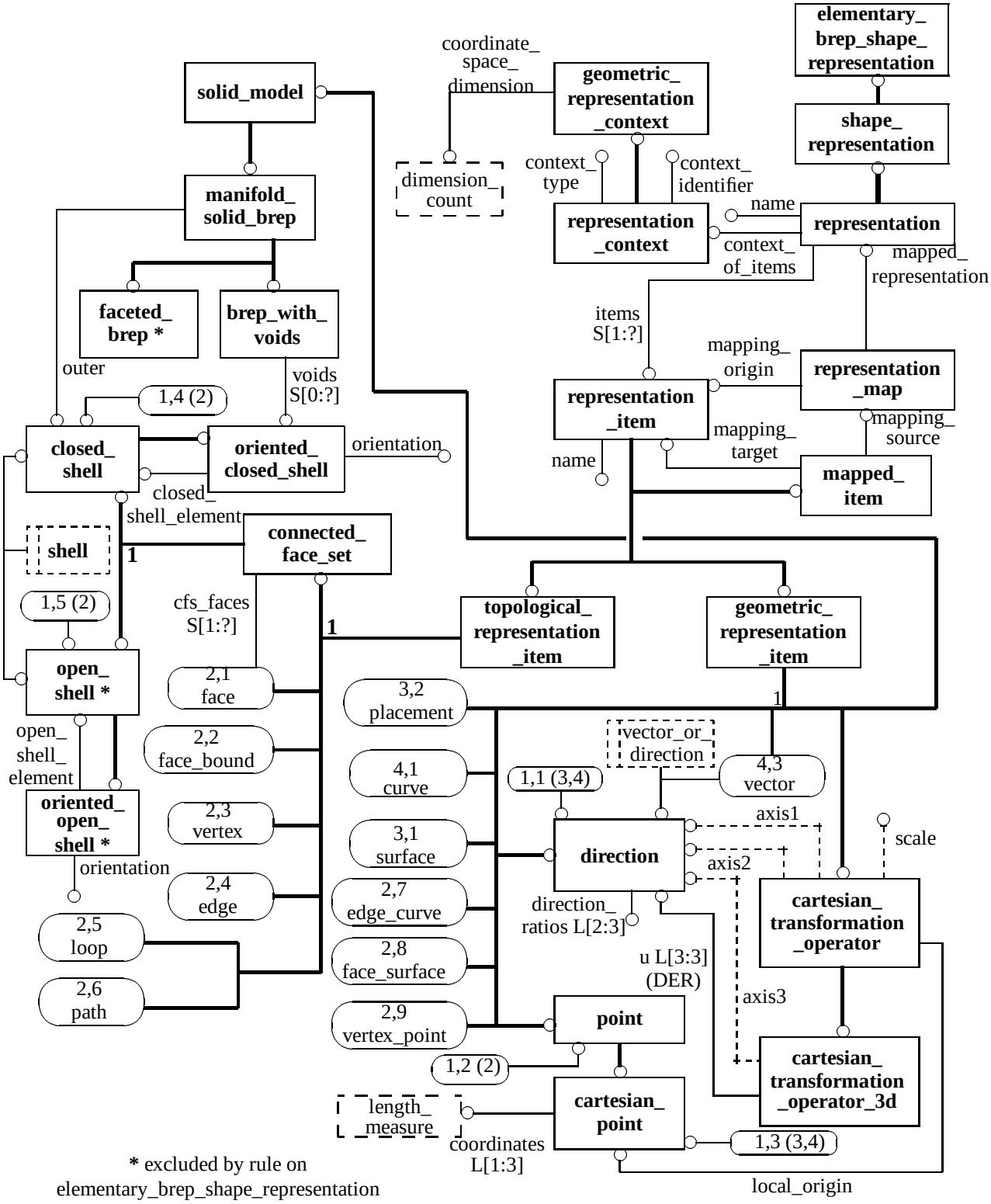
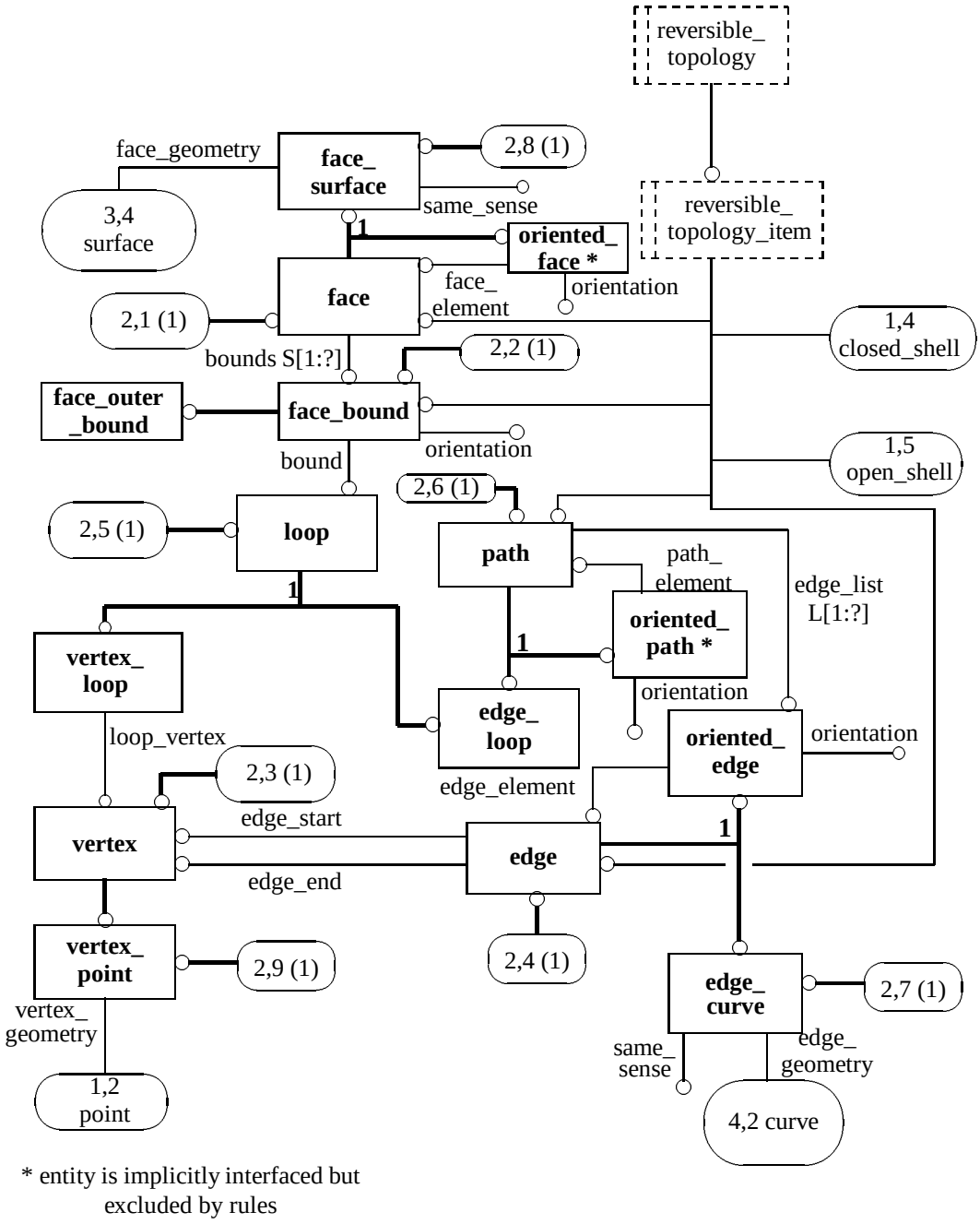



**Figure D.1 – aic_elementary_boundary_representation EXPRESS-G diagram, page 1 of 4**




* entity is implicitly interfaced but excluded by rules

**Figure D.2 – aic_elementary_boundary_representation EXPRESS-G diagram, page 2 of 4**

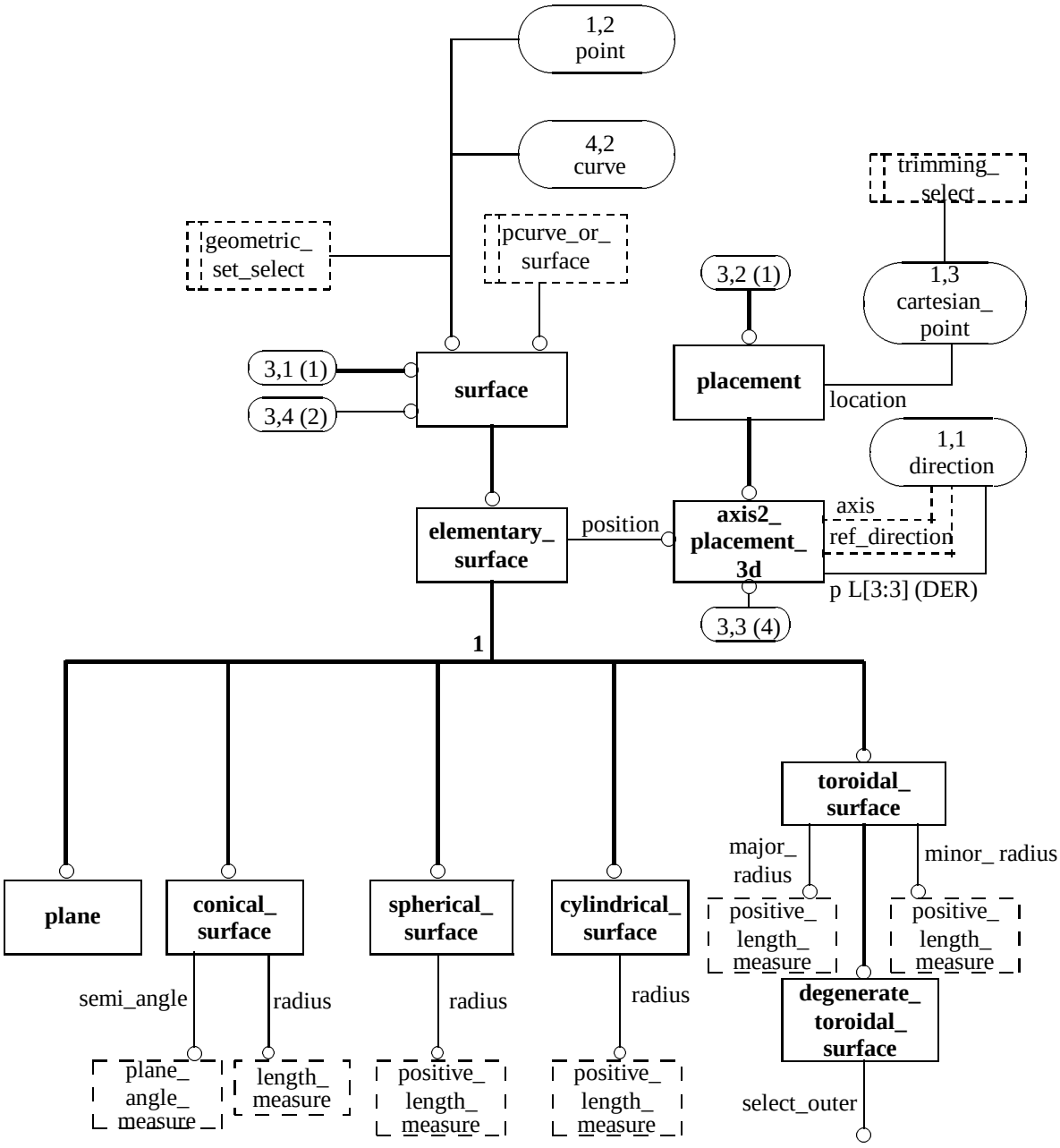



**Figure D.3 – aic_elementary_boundary_representation EXPRESS-G diagram, page 3 of 4**

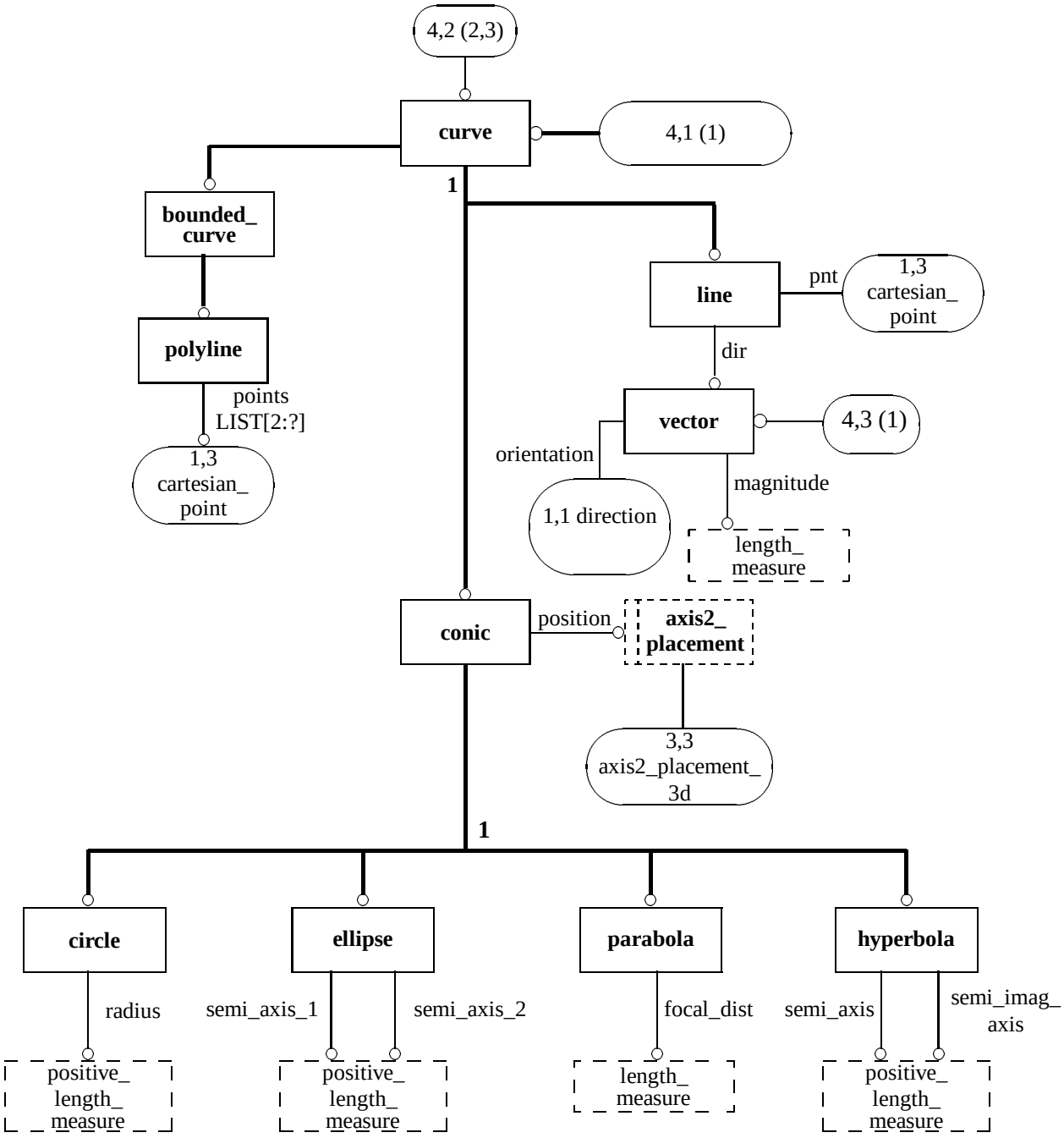



**Figure D.4 – aic_elementary_boundary_representation EXPRESS-G diagram, page 4 of 4**

# Annex E
(informative)

# AIC conformance requirements and test purposes

## E.1 AIC conformance requirements: elementary B-rep

Any application protocol that uses this AIC may require conformance to the AIC conformance requirements defined below when instantiating an **elementary_brep_shape_representation**.

Conformance to this AIC means that all the defined types and entity types defined in the EXPRESS listing are supported. The only legitimate use, within the context of this AIC, for a geometric or topological entity instance is for the purpose of defining an **elementary_brep_shape_representation**.

The following entities are instantiable as part of the definition of an **elementary_brep_shape_representation**:

— axis2_placement_3d;

— brep_with_voids;

— cartesian_point;

— cartesian_transformation_operator_3d;

— circle;

— closed_shell;

— conical_surface;

— cylindrical_surface;

— degenerate_toroidal_surface;

— direction;

— edge_curve;

— edge_loop;

— elementary_face;

— ellipse;

— face_bound;

— face_outer_bound;

— face_surface;

— hyperbola;

— line;

— manifold_solid_brep;

— mapped_item;

— oriented_closed_shell;

— parabola;

— plane;

— polyline;

— representation_map;

— spherical_surface;

— toroidal_surface;

— vector;

— vertex_loop;

— vertex_point.

## E.2 Test purposes for elementary B-rep AIC

This clause defines conformance test purposes which are appropriate for the elementary B-rep AIC. The test purposes are based on the constructs found in clause 4 of this part of ISO 10303.

### E.2.1 elementary_brep_shape_representation

The following test purposes are derived from the definition of this entity:

**EB1: representation** as **shape_representation** as **elementary_brep_shape_representation**. (see E.3.1).

**EB2: elementary_brep_shape_representation** with context as **geometric_context** with **items** as **manifold_solid_brep**. (see E.3.1).

**EB3: elementary_brep_shape_representation** with context as **geometric_context** with **items** as **mapped_item**; (see E.3.6).

**EB4: elementary_brep_shape_representation** with context as **geometric_context** with **items** as two or more items as **manifold_solid_brep**, or **mapped_item**, or **axis2_placement_3d**, including at least one **axis2_placement_3d**. (see E.3.6)

## E.2.2 manifold_solid_brep

The following test purposes are derived from the definition of this entity:

**EB5: manifold_solid_brep** with **outer** (voids absent) as **closed_shell**. (NOT **oriented_closed_-shell** subtype.) (see E.3.1).

**EB6: manifold_solid_brep** as **brep_with_voids** subtype with outer as **closed_shell** and voids as a SET of one **oriented_closed_shell**. (voids present) (see E.3.2)

**EB7: manifold_solid_brep** as **brep_with_voids** subtype with outer as **closed_shell** and voids as a SET of more than one **oriented_closed_shell**. (voids present) (see E.3.2).

## E.2.3 oriented_closed_shell

The following test purpose is derived from the definition of this entity and the constraints imposed on the **elementary_brep_shape_representation**:

**EB8: oriented_closed_shell** with **orientation** = FALSE. (see E.3.2).

## E.2.4 closed_shell

The following test purpose is derived from the definition of this entity and the constraints imposed on the **elementary_brep_shape_representation**:

**EB9: closed_shell** with **cfs_faces** as a SET of one **face_surface**. (see E.3.2).

**EB10: closed_shell** with **cfs_faces** as a SET of more than one **face_surface**. (see E.3.1).

## E.2.5 face

The following test purposes are derived from the definition of this entity and the constraints imposed on the **elementary_brep_shape_representation**:

**EB11: face** as **face_surface** with **bounds** as SET of one **face_bound** as **face_outer_bound** with **orientation** = TRUE. (see E.3.1).

**EB12:** **face** as **face_surface** with **bounds** as SET of one **face_bound** as **face_outer_bound** with **bound** as **edge_loop** (not **oriented_path**) and **orientation** = FALSE. (see E.3.1).

**EB13:** **face** as **face_surface** with **bounds** as SET of at least two **face_bound**s with **bound** as **edge_loop** and **orientation** = TRUE. (see E.3.1).

**EB14:** **face** as **face_surface** with **bounds** as SET of at least two **face_bound**s with **bound** as **edge_loop** and **orientation** = FALSE. (see E.3.1).

**EB15:** **face** as **face_surface** with **bounds** as SET of at least two **face_bound**s (including one **vertex_loop**). (see E.3.5).

## E.2.6 face_surface

The following test purposes are derived from the definition of this entity and the constraints imposed on the **elementary_brep_shape_representation**:

**EB16:** **face_surface** with **face_geometry** as **surface**. (see E.3.1).

**EB17:** **face_surface** with **same_sense** = TRUE. (see E.3.1).

**EB18:** **face_surface** with **same_sense** = FALSE. (see E.3.5).

## E.2.7 surface

The following test purposes are derived from the definition of this entity and the constraints imposed on the **face_surface**:

**EB19:** **surface** as **elementary_surface**. (see E.3.1).

## E.2.8 elementary_surface

The following test purposes are derived from the definition of this entity and the constraints imposed on the **face_surface**:

**EB20:** **elementary_surface** with **position** as **axis2_placement_3d** with **axis** present. (see E.3.1).

**EB21:** **elementary_surface** with **position** as **axis2_placement_3d** with **axis** absent. (see E.3.4).

**EB22:** **elementary_surface** with **position** as **axis2_placement_3d** with **ref_direction** present. (see E.3.1).

**EB23:** **elementary_surface** with **position** as **axis2_placement_3d** with **ref_direction** absent. (see E.3.4).

**EB24:** **elementary_surface** as **plane**. (see E.3.1).

**EB25:** **elementary_surface** as **cylindrical_surface**. (see E.3.1).

**EB26:** **elementary_surface** as **conical_surface**. (see E.3.5).

**EB27:** **elementary_surface** as **spherical_surface**. (see E.3.1).

**EB28:** **elementary_surface** as **toroidal_surface**. (see E.3.3).

## E.2.9 loop

The following test purposes are derived from the definition of this entity and the constraints imposed on the **face_surface**:

**EB29:** **loop** as **edge_loop**. (see E.3.1).

**EB30:** **loop** as **vertex_loop** with **loop_vertex** as **vertex_point** with **vertex_geometry** as **cartesian_-point**. (see E.3.2).

## E.2.10 edge

The following test purposes are derived from the definition of this entity and the constraints imposed on the **face_surface**:

**EB31:** **edge** as **edge_curve** with **edge_start** as **vertex_point** and **edge_end** as **vertex_point**. (see E.3.1).

**EB32:** **edge** as **oriented_edge** with **orientation** TRUE. (see E.3.1).

**EB33:** **edge** as **oriented_edge** with **orientation** FALSE. (see E.3.3).

## E.2.11 edge_curve

The following test purposes are derived from the definition of this entity and the constraints imposed on the **face_surface**:

**EB34:** **edge_curve** with **edge_geometry** as **line**. (see E.3.3).

**EB35:** **edge_curve** with **edge_geometry** as **polyline**. (see E.3.4).

**EB36:** **edge_curve** with **edge_geometry** as **conic**. (see E.3.1).

**EB37:** **edge_curve** with **same_sense** = TRUE. (see E.3.1).

**EB38:** **edge_curve** with **same_sense** = FALSE. (see E.3.5)>

### E.2.12 conic

The following test purposes are derived from the definition of this entity and the constraints imposed on the **face_surface**:

**EB39:** **conic** as **circle**. (see E.3.1).

**EB40:** **conic** as **ellipse**. (see E.3.1).

**EB41:** **conic** as **hyperbola**. (see E.3.5).

**EB42:** **conic** as **parabola**. (see E.3.5).

### E.2.13 polyline

The following test purpose is derived from the definition of this entity and the constraints imposed on the **face_surface**:

**EB43:** **polyline** with **points** as a LIST of 3 or more **cartesian_point**s. (see E.3.4).

### E.2.14 cartesian_transformation_operator_3d

The following test purposes are derived from the definitions of this entity, the **mapped_item** entity, and the constraints imposed on the **elementary_brep_shape_representation**:

**EB44:** **mapped_item** with **mapping_target** as **cartesian_transformation_operator_3d**. (see E.3.7)

**EB45:** **cartesian_transformation_operator** as **cartesian_transformation_operator_3d** with scale as REAL not equal to 1.0. (see E.3.7).

## E.3 Abstract test cases for elementary B-rep

The post-processor abstract test cases in this clause are fully documented in EXPRESS-I.
A simple textual description is provided for each pre-processor test case to enable the creation of a model similar to that described in the EXPRESS-I documentation of the post-processor test. For each test case a number of relevant test purposes is identified.

NOTE Many of the test purposes are applicable to more than one test case, but the criteria are only defined with the first such test case. This applies in particular to many of the purposes documented in test case eb1.

### E.3.1 Test case eb1

Test case eb1 is the most basic test case consisting of the faces needed to define a single solid cylinder with hemispherical base and elliptic top. All geometry is explicitly defined with no defaults and no sense reversals of geometry required. The definition of the faces is provided by the **cylinder_sphere_shell** context using the original parameters.

### E.3.1.1 Test purpose coverage

The AIM test purposes addressed by this test case are listed below.

EB1 representation as shape_representation as elementary_brep_shape_representation;
EB2 elementary_brep_shape_representation with context as geometric_representation_context with items as manifold_solid_brep;
EB5 manifold_solid_brep with outer (voids absent) as closed_shell (NOT oriented_closed_shell subtype);
EB10 closed_shell with cfs_faces as a SET of more than one face_surface;
EB11 face as face_surface with bounds as a SET of one face_bound as face_outer_bound with orientation TRUE;
EB12 face as face_surface with bounds as a SET of one face_bound as face_outer_bound with orientation FALSE;
EB13 face as face_surface with bounds as a SET of more than one face_bound with bound as an edge_loop and orientation TRUE;
EB14 face as face_surface with bounds as a SET of more than one face_bound with bound as an edge_loop and orientation FALSE;
EB16 face_surface with face_geometry as surface;
EB17 face_surface with same_sense = TRUE;
EB19 surface as elementary_surface;
EB20 elementary_surface with position as axis2_placement_3d with axis present;
EB22 elementary_surface with position as axis2_placement_3d with ref_direction present;
EB24 elementary_surface as plane;
EB25 elementary_surface as cylindrical_surface;
EB27 elementary_surface as spherical_surface;
EB29 loop as edge_loop;
EB31 edge as edge_curve with edge_start as vertex_point and edge_end as vertex_point;
EB32 edge as oriented_edge with orientation as TRUE;
EB36 edge_curve with edge_geometry as conic;
EB37 edge_curve with same_sense as TRUE;
EB39 conic as a circle;
EB40 conic as an ellipse.

### E.3.1.2 Preprocessor input specification

Create an **elementary_brep_shape_representation** consisting of a single **manifold_solid_brep**. The **manifold_solid_brep** should be in the form of a solid cylinder with a hemi-spherical base and a sloping planar top. The centre of the hemisphere is at the origin and the Z axis is the axis of the cylinder. The B-rep object is defined by a single closed shell with 3 faces. A suitable set of dimensions is defined in the EXPRESS-I specification below.

NOTE The **cylinder_sphere_shell** context is used, in its simplest form with default values, to define the faces of the B-rep.

### E.3.1.3 Postprocessor input specification

```
*)
TEST_CASE example_ebrep_1; WITH aic_elementary_brep;


  REALIZATION

  LOCAL
    shell_object : closed_shell ;
    cysp_solid : manifold_solid_brep ;
    ebsr : elementary_brep_shape_representation ;
    its_units : named_unit ;
    its_context : representation_context ;
  END_LOCAL;

    CALL cylinder_sphere_shell ; -- uses default values, so no WITH
      IMPORT (shell_object := @cyspshell; ) ;
    END_CALL;
      cysp_solid := manifold_solid_brep ('cysp_solid', shell_object) ;

      its_units := length_unit() || si_unit ('milli', 'metre') ;

      its_context := geometric_representation_context
                      ('context_1', 'context_for_cylinder_sphere', 3) ||
                        global_unit_assigned_context ( [its_units] ) ;

      ebsr := elementary_brep_shape_representation
                  ( 'ebsr', [cysp_solid], its_context );
  END_REALIZATION;
END_TEST_CASE;
(*
```

### E.3.1.4 Postprocessor verdict criteria

**EB1:** All WRs on **elementary_brep_shape_representation** shall be verified.

**EB2:** Length units shall be correctly interpreted, model re-created shall contain no **polyloop**s or **vertex_loop**s.

**EB5:** Shell normals shall point out of solid.

**EB10:** Faces shall be connected along edges, no other face intersections shall occur.

**EB11:** Face geometry shall be correctly trimmed by **face_bound**.

**EB12:** **face_bound** with orientation FALSE shall be correctly interpreted to define correct portion of face surface.

**EB13:** Multiple bounds shall be correctly interpreted to trim face.

**EB14:** **face_bound**s with different orientations shall be correctly interpreted.

**EB16:** **edge_curve**s and vertices of bounding **edge_loop**s shall lie on surface defining **face_geometry**.

**EB20:** **axis2_placement** with **axis** present shall be correctly interpreted to locate surface.

**EB22:** **axis2_placement** with **ref_direction** present shall be correctly interpreted to locate surface.

**EB24:** Bounding loops of face with **face_geometry** as **plane** shall be co-planar.

**EB25:** Unbounded **cylindrical_surface** shall be bounded by **edge_loop**s. **[**EB27] Correct portion of **spherical_surface** shall be defined by **edge_loop**s.

**EB31:** All **vertex_point**s shall lie on **edge_curve**s.

**EB36:** **edge** with identical start and end vertices and **edge_geometry** as ellipse shall be correctly interpreted as closed ellipse.

**EB39:** **edge** with identical start and end vertices and **edge_geometry** as circle shall be correctly interpreted as a closed circle.

**EB40:** **ellipse** subtype of **conic** shall be correctly interpreted.

## E.3.2 Test case eb2

Test case eb2 is designed to test the definition of an elementary B-rep containing one or more voids. The **cylinder_sphere_shell** context is used with different parameters to define the outer shell and a void shell. The result is a hollow cylindrical solid with void(s) of a similar shape, or spherical.

NOTE If required this test can easily be modified to test geometric precision by varying the parameters to define voids that are very close to each other or to the outer shell. As defined in the current version of this test case there should be no possibility of such interference.

### E.3.2.1 Test purpose coverage

The test purposes addressed by this test case are listed below.

EB6 manifold_solid_brep as brep_with_voids subtype with outer as closed_shell and voids as a SET of one oriented_closed_shell (voids present).
EB7 manifold_solid_brep as brep_with_voids subtype with outer as closed_shell and voids as a SET of more than one oriented_closed_shell.

EB8 oriented_closed_shell with orientation as FALSE.
EB9 closed_shell with cfs_faces as a SET of one face_surface .
EB30 loop as vertex_loop.

### E.3.2.2 Preprocessor input specification

Create **elementary_brep_shape_representation**s consisting of a single **manifold_solid_brep**. The **manifold_solid_brep** should be in the form of a solid cylindrical solid with a hemi-spherical top and a sloping planar top. One such B-rep shall contain a void of a similar shape and orientation. A second example shall contain two such non-intersecting voids, one of a similar shape, the other spherical. The spherical void shall be defined with a single **face_surface** using a **vertex_loop**. The centre of the hemi-sphere for the outer shell is at the origin and the Z axis is the axis of the cylinder. Each shell is defined as a single closed shell with 3 faces. A suitable set of dimensions is defined in the EXPRESS-I specification below.

### E.3.2.3 Postprocessor input specification

NOTE 1 The **cylinder_sphere_shell** context is used, with appropriate parameters, to define the faces of the B-rep outer shells and to define the void shells.

NOTE 2 Outer shell of brep_with_voids is a closed_shell and not an oriented_closed_shell, oriented_closed_shell is used to define voids, orientation must be FALSE.

```
*)
TEST_CASE example_ebrep_2; WITH aic_elementary_brep;

  REALIZATION

  LOCAL
    shell_object, hollow1, hollow2 : closed_shell ;
    void1, void2 : oriented_closed_shell;
    cylsp_with_void : brep_with_voids ;
    cylsp_with_voids : brep_with_voids ;
    ebsr1, ebsr2 : elementary_brep_shape_representation ;
    its_units : named_unit ;
    context1, context2 : representation_context ;
    sph2 : spherical_surface ;
    l1, l2 : length_measure ;
    top_pt : cartesian_point ;
    top_vert : vertex_point ;
    v_loop : vertex_loop ;
    s_bound : face_outer_bound ;
    sp_face : face_surface ;
    sp_shell : closed_shell ;
  END_LOCAL;
```

```
CALL cylinder_sphere_shell ; -- uses default values, so no WITH
  IMPORT (shell_object := @cyspshell; );
END_CALL;

CALL cylinder_sphere_shell; -- parameters re-set for dimensions
  IMPORT (hollow1 := @cyspshell; ); --large void
  WITH (orc := 10; rad := 12; ht := 50; );
END_CALL;

CALL cylinder_sphere_shell ; -- parameters re-set for dimensions
    (sphere for spherical void)
  IMPORT (sph2 := @sphere;
          l1 := @orc;
          l2 ;= @rad ;);
  WITH (orc := -5; rad := 10 ;);
END_CALL;

void1 := oriented_closed_shell ('void1', hollow1, FALSE) ;
top_pt := cartesian_point ('top_pt', [l1, l1, (l1 + l2) ]) ;
top_v := vertex_point ('top_v', top_pt ) ;
v_loop := vertex_loop ('v_loop', top_v ) ;
s_bound := face_outer_bound ('s_bound', v_loop, TRUE ) ;
sp_face := face_surface ('sp_face', [s_bound], sph2, TRUE );
sp_shell := closed_shell ('sp_shell', [sp_face] );
void2 := oriented_closed_shell ('void2', sp_shell, FALSE) ;

cylsp_with_void :=
         manifold_solid_brep ('cylsp_w_v', shell_object) ||
                   brep_with_voids ([void1]) ;

cylsp_with_voids :=
     manifold_solid_brep ('cylsp_w_vs', shell_object)||
                    brep_with_voids ([void1, void2]) ;

its_units := length_unit() || si_unit ('milli', 'metre') ;

context1 := geometric_representation_context ,
                 ('context_1', 'context_for_cylsp_with_void', 3) ||
                     global_unit_assigned_context ( [its_units] ) ;

context2 := geometric_representation_context ,
                ('context_1', 'context_for_cylsp_with_voids', 3) ||
                     global_unit_assigned_context ( [its_units] ) ;

ebsr1 := elementary_brep_shape_representation
```

```
            ( 'ebsr1', [cylsp_with_void], context1 ) ;

    ebsr2 := elementary_brep_shape_representation
            ('ebsr2', [cylsp_with_voids], context2 ) ;

  END_REALIZATION;
END_TEST_CASE;
(*
```

### E.3.2.4 Postprocessor verdict criteria

**EB6:** Void shell shall not intersect outer shell, void shell shall be completely within outer shell.

**EB7:** Void shells shall not intersect outer shell or each other, each void shell shall be completely within outer shell, two **elementary_brep_shape_representation**s ebsr1 and ebsr2 shall not be spatially related.

**EB8:** Normal to void shells shall point into voids.

**EB9:** **closed_shell** with single face with **spherical_surface** geometry shall be correctly processed.

**EB30:** **vertex_loop** shall be correctly processed as **face_bound** to define face as complete spherical surface.

## E.3.3 Test case eb3

Test case eb3 is a simple test case consisting of the faces needed to define a single solid segment of a torus bounded by planes. One of the plane/torus intersections is represented by a planar polyline. The definition of the shell is provided by the **toroidal_segment** context using the original parameters.

### E.3.3.1 Test purpose coverage

The test purposes specifically addressed by this test case are listed below.

EB28 elementary_surface as toroidal_surface;
EB33 edge as oriented_edge with orientation as FALSE;
EB34 edge_curve with edge_geometry as line;
EB35 edge_curve with edge_geometry as polyline.

### E.3.3.2 Preprocessor input specification

Create an **elementary_brep_shape_representation** consisting of a single **manifold_solid_brep**. The **manifold_solid_brep** should be in the form of a toroidal segment centred at origin with z axis as central axis. The segment is created by intersecting the torus with three planes, one of which (z = 0) is through the centre and normal to the central axis. Other two planes are parallel to each other with one (x =

0) passing through the centre. Intersection curves are circular arcs or a polyline. The B-rep object is defined by a single closed shell with 4 faces. A suitable set of dimensions is defined in the EXPRESS-I specification below.

### E.3.3.3 Postprocessor input specification

NOTE The **toroidal_segment** context is used, with default parameters, to define the faces and all geometry and topology of the B-rep.

```
*)
TEST_CASE example_ebrep_3; WITH aic_elementary_brep;

  REALIZATION

  LOCAL
    shell_object : closed_shell ;
    torus_solid : manifold_solid_brep ;
    ebsr : elementary_brep_shape_representation ;
    its_units : named_unit ;
    its_context : representation_context ;
  END_LOCAL;

    CALL toroidal_segment ; -- uses default values, so no WITH
      IMPORT (shell_object := @torshell; ) ;
    END_CALL;
      torus_solid := manifold_solid_brep ('torus_solid', shell_object) ;

    its_units := length_unit() || si_unit ('milli', 'metre') ;

    its_context := geometric_representation_context
                     ('context_1', 'context_for_torshell', 3) ||
                     global_unit_assigned_context ( [its_units] ) ;

      ebsr := elementary_brep_shape_representation
                  ('ebsr', [torus_solid], its_context );
  END_REALIZATION;
END_TEST_CASE;
(*
```

### E.3.3.4 Postprocessor verdict criteria

**EB28:** **toroidal_surface** face shall be processed and bounded correctly.

**EB33:** When edge is re-used with FALSE orientation it shall be correctly interpreted.

**EB34:** **line** shall be correctly trimmed by vertices.

**EB35:** All **polyline** points shall lie on **toroidal_surface** with a tolerance of less than 0.0000001. Polyline points shall be coplanar.

### E.3.4 Test case eb4

Test case eb4 is a test case consisting of the faces needed to define a single solid resulting from the union of two cylinders of different radii that have orthogonal axes. The intersection curve is a closed 3D curve represented by a polyline. The definition of the shell is provided by the **cylinder_union_polyline** context using the original parameters.

#### E.3.4.1 Test purpose coverage

The test purposes specifically addressed by this test case are listed below.

EB21 elementary_surface with position as axis2_placement_3d with axis absent;
EB23 elementary_surface with position as axis2_placement_3d with ref_direction absent;
EB35 edge_curve with edge_geometry as (closed 3D) polyline.

#### E.3.4.2 Preprocessor input specification

Create an **elementary_brep_shape_representation** consisting of a single **manifold_solid_brep**. The **manifold_solid_brep** should be in the form of two perpendicular intersecting cylinders. Defaults shall be used in defining the **axis2_placement_3d** to locate one of these cylinders. The intersection curve shall be represented by a polyline. A suitable set of dimensions is defined in the EXPRESS-I specification below.

#### E.3.4.3 Postprocessor input specification

NOTE The **cylinder_union_polyline** context is used, with default parameters, to define the faces and all geometry and topology of the B-rep.

```
*)
TEST_CASE example_ebrep_4; WITH aic_elementary_brep;

  OBJECTIVE

  REALIZATION

  LOCAL
    shell_object : closed_shell ;
    cylxcyl_solid : manifold_solid_brep ;
    ebsr : elementary_brep_shape_representation ;
    its_units : named_unit ;
    its_context : geometric_representation_context ;
  END_LOCAL;
```

```
    CALL cylinder_union_polyline ; -- uses default values, so no WITH
      IMPORT (shell_object := @cxcshell ;) ;
    END_CALL;
      cylxcyl_solid := manifold_solid_brep ('cylxcyl_solid',
                                              shell_object) ;
      its_units := length_unit() || si_unit ('milli', 'metre') ;

      its_context := geometric_representation_context
                       ('context_1', 'context_for_cylxcyl', 3) ||
                     global_unit_assigned_context ( [its_units] ) ;

      ebsr := elementary_brep_shape_representation
                  ('ebsr',  [cylxcyl_solid], its_context );
  END_REALIZATION;
END_TEST_CASE;
(*
```

### E.3.4.4 Postprocessor verdict criteria

**EB21:** Default value of **axis** attribute shall be correctly supplied.

**EB23:** Default value of **ref_direction** attribute shall be correctly supplied.

**EB35:** All polyline points shall lie on BOTH **cylindrical_surface**s with a tolerance of less than 0.000001.

## E.3.5 Test case eb5

Test case eb5 is a test case consisting of the faces needed to define solids resulting from the intersection of inclined planes with a cone. Face boundary curves are ellipse, hyperbola, parabola, circular arc, and line segments. The definition of the shells is provided by the **cone_faces** context using the original parameters.

### E.3.5.1 Test purpose coverage

The test purposes specifically addressed by this test case are listed below.

EB2 elementary_brep_shape_representation with context as geometric_representation_context with items as a SET of more than one manifold_solid_brep;
EB15 face as face_surface with bounds as a SET of at least two face_bounds including one vertex_loop;
EB26 elementary_surface as conical_surface;
EB40 conic as a ellipse;
EB41 conic as hyperbola;
EB42 conic as parabola.

### E.3.5.2 Preprocessor input specification

Create an **elementary_brep_shape_representation** consisting of two **manifold_solid_brep**s. The **manifold_solid_brep**s should be in the form of cones bounded by inclined planes. The planes are chosen to produce intersection curves in the form of elliptic, parabolic, hyperbolic and circular arcs. One cone has a **vertex_loop** at the apex and an elliptic base. The second has the same elliptic curve as its top profile. A suitable set of dimensions is defined in the EXPRESS-I specification below.

### E.3.5.3 Postprocessor input specification

NOTE The **cone_shell** context is used, with default parameters, to define the faces and all geometry and topology of the B-reps.

```
*)
TEST_CASE example_ebrep_5; WITH aic_elementary_brep;

  REALIZATION

  LOCAL
    shell1, shell2 : closed_shell ;
    cone1, cone2 : manifold_solid_brep ;
    ebsr : elementary_brep_shape_representation ;
    angle_u, len_u, angle_c_u : named_unit ;
    ang_m_wu : plane_angle_measure_with_unit ;
    its_context : geometric_representation_context ;
  END_LOCAL;

    CALL cone_shell ; -- uses default values, so no WITH
      IMPORT (shell1 := @vconeshell ;
              shell2 := @con4fshell ; ) ;
    END_CALL;
      cone1 := manifold_solid_brep ('cone1', shell1) ;
      cone2 := manifold_solid_brep ('cone2', shell2) ;

      angle_c_u := plane_angle_unit() || si_unit ( , 'radian' ) ;
      ang_m_wu := plane_angle_measure_with_unit(.017453293, angle_c_u);
      angle_u := plane_angle_unit() ||
                    conversion_based_unit('degree', ang_m_wu ) ;
      len_u  := length_unit() || si_unit ('milli', 'metre') ;

      its_context := geometric_representation_context
                         ('context_1', 'context_for_cones', 3) ||
                     global_unit_assigned_context ([len_u, angle_u]) ;

      ebsr := elementary_brep_shape_representation
                 ('ebsr', [cone1, cone2], its_context );
```

```
  END_REALIZATION;
END_TEST_CASE;
(*
```

### E.3.5.4 Postprocessor verdict criteria

**EB2:** Two B-reps should exactly fit together with a common face surface, two distinct B-reps shall not intersect.

**EB15:** **face** with **face_bound** as **vertex_loop** shall be correctly processed.

**EB26:** **conical_surface**s shall be correctly trimmed by bounding edges.

**EB40:** All points on the ellipse shall lie exactly on BOTH **conical_surface**s and on intersecting plane.

**EB41:** All hyperbolic edge points shall lie exactly on BOTH **conical_surface** and on intersecting **plane**, **hyperbola** shall be properly trimmed by **vertex_point**s.

**EB42:** All parabolic edge points shall lie exactly on BOTH **conical_surface** and on intersecting **plane**, **parabola** shall be properly trimmed by **vertex_point**s.

## E.3.6 Test case eb6

Test case eb6 is designed to test the use of **mapped_item**s in the creation of a simple assembly of elementary B-reps. It also provides a test of the consistent behaviour of **geometric_representation_context**s in distinguishing between coordinate spaces. This test makes use of the **cylinder_union_polyline** context to define the geometry and topology.

### E.3.6.1 Test purpose coverage

The test purposes addressed by this test case are listed below.

EB3 elementary_brep_shape_representation with context as geometric_representation_context with items as mapped_item;
EB4 elementary_brep_shape_representation with context as geometric_representation_context with two or more items as manifold_solid_brep, or mapped_item, or axis2_placement_3d, including at least one axis2_placement_3d.

### E.3.6.2 Preprocessor input specification

Create a basic **elementary_brep_shape_representation** consisting of a **manifold_solid_brep** in the form of 2 intersecting cylinders as defined in eb4. A **mapped_item** is then defined as a translated and rotated copy of this representation. Two further **elementary_brep_shape_representation**s are then defined; one, in the same context, consists of the **mapped_item** only; the other, in a distinct context

contains the original **manifold_solid_brep**, the **mapped_item** and an **axis2_placement_3d**. Full details of dimensions and of the mapping are defined in the EXPRESS-I specification below.

### E.3.6.3 Postprocessor input specification

NOTE 1 In the specification below two distinct **geometric_representation_context**s are created for the geometric definitions.

NOTE 2 The **cylinder_union_polyline** context is used, with default parameters, to define the faces and all geometry and topology of the B-reps.

NOTE 3 **ebsr1** should be a rotated and translated copy of **ebsr**.

NOTE 4 **ebsrass** should be equivalent to 2 copies of **ebsr** 'glued' together.

```
 *)

TEST_CASE example_ebrep_6; WITH aic_elementary_brep;

  REALIZATION

  LOCAL
    origin : cartesian_point ;
    pos_z, neg_y  : direction ;
    refaxes, topaxes, baseaxes : axis_placement_3d;
    shell_object  : closed_shell ;
    cylxcyl : manifold_solid_brep ;
    ebsr, ebsr1, ebsrass : elementary_brep_shape_representation ;
    grc1, grc2 : geometric_representation_context ;
    transrot1, trans2  : mapped_item ;
    mapping1, mapping2   : representation_map ;
  END_LOCAL;

    CALL cylinder_union_polyline ; -- uses default values, so no WITH
      IMPORT (shell_object := @cycshell;
              origin := @origin;
              baseaxes := @ ab; refaxes := @ar; topaxes := @at; );
    END_CALL;


cylxcyl := manifold_solid_brep ('cylxcyl', shell_object) ;

grc1 = geometric_representation_context ('ctx1',
                            'context for cylinder union', 3) ;

grc2 = geometric_representation_context ('ctx2',
```

```
                    'context for rotated cylinder union', 3) ;

ebsr := elementary_brep_shape_representation ('ebsr', [cylxcyl], grc1);

mapping1 := representation_map (baseaxes, ebsr );
transrot1 := mapped_item ('transrot1', mapping1, refaxes );

(* Define representation using transrot1 only  *)
ebsr1 := elementary_brep_shape_representation ('ebsr1',
                                          [transrot1], grc1) ;
(* Define representation that is an assembly of intersecting cylinders
      +  mapped  (translated) copy.
*)

mapping2 := representation_map (baseaxes, ebsr );
trans2 := mapped_item ('trans2', mapping2, topaxes );

ebsrass := elementary_brep_shape_representation
         ('ebsrass', [ebsr1, ebsrot2, baseaxes], grc2 ) ;

  END_REALIZATION;
END_TEST_CASE;
(*
```

### E.3.6.4 Postprocessor verdict criteria

**EB3:** After processing the mapped_item shall be correctly interpreted, result is a rotated and translated copy of original B-rep.

**EB4:** The **elementary_brep_shape_representation** containing a **mapped_item**, B-rep and **axis2_placement_3d** shall be correctly interpreted, the result is the original B-rep and a rotated and translated copy of original B-rep which touches over a face.

**EB3 and EB4:** The two distinct **elementary_brep_shape_representation**s shall not be spatially related.

## E.3.7 Test case eb7

Test case eb7 is designed to test the use of **mapped_item**s in conjunction with a **cartesian_transformation_operator** in the creation of a simple assembly of faceted B-reps. The use of a scaling factor is tested. This test makes use of the **cylinder_sphere_shell** context to define the geometry and topology.

### E.3.7.1 Test purpose coverage

The test purposes specifically addressed by this test case are listed below.

EB44 mapped_item with mapping_target as cartesian_transformation_operator_3d.
EB45 cartesian_transformation_operator as cartesian_transformation_operator_3d with scale as REAL not equal to 1.0.

### E.3.7.2 Preprocessor input specification.

Create an **elementary_brep_shape_representation** consisting of a single **manifold_solid_brep**. The B-rep should be in the form of a solid cylinder with a hemispherical base and a sloping planar top. The hemisphere is centred at the origin and the axis of the cylinder should lie along the z coordinate axis. This representation is then used in conjunction with the **mapped_item** entity and a **cartesian_transformation_operator**, to create, in the same **representation_context**, a representation consisting of a rotated, translated and scaled (not by 1.0) copy of the original representation and the original B-rep. The translation and the magnitude of the scaling factor should be chosen to ensure that the cylinders of the two B-reps touch, but do not intersect.

### E.3.7.3 Postprocessor input specification

NOTE The **cylinder_sphere_shell** context is used, with default parameters, to define the faces and all geometry and topology of the B-rep. **csrans** should be a rotated scaled and translated copy of **cysp_solid**.

```
  *)

TEST_CASE example_ebrep_7; WITH elementary_brep_aic;

REALIZATION

  LOCAL
    radius : length_measure ;
    origin, neworigin : cartesian_point ;
    pos_x, pos_y, pos_z, neg_z  : direction ;
    oldaxes : axis_placement_3d;
    transform : cartesian_transformation_operator_3d;
    shell_object  : closed_shell ;
    cysp_solid : manifold_solid_brep ;
    ebsr, ebsrass : elementary_brep_shape_representation ;
    its_units : named_unit ;
    grc1, grc2 : representation_context ;
    cstrans  : mapped_item ;
    mapping1  : representation_map ;
  END_LOCAL;
   CALL cylinder_sphere_shell ; -- uses default values, so no WITH
      IMPORT (shell_object := @cyspshell;
                    pos_x  := @pos_x ;
                    pos_y  := @pos_y ;
                    pos_z  := @pos_z ;
                    origin := @origin ;
```

```
                    radius := @rad ; ) ;
   END_CALL;

cysp_solid := manifold_solid_brep ('cysp_solid', shell_object) ;

its_units := length_unit() || si_unit ('milli', 'metre') ;

grc1 = geometric_representation_context ('ctx1',
                            'context for cysp_solid', 3) ||
  global_unit_assigned_context ( [its_units] ) ;

grc2 = geometric_representation_context ('ctx2',
      'context for assembly', 3) ||
      global_unit_assigned_context ( [its_units] ) ;

(*  Define axis_placement and cartesian_transformation_operator for use
     in mapping *)

neworigin := cartesian_point ([1.8*radius, 0.0, 0.0] );
neg_z := direction ([0.0, 0.0, -1.0] );

oldaxes := axis2_placement_3d ('oldaxes', origin, pos_z, pos_x) ;

transform := cartesian_transformation_operator_3d ('transform',
                             pos_x, neg_z, neworigin, 0.8, pos_y );

ebsr := elementary_brep_shape_representation ('ebsr',
                                 [cysp_solid, oldaxes], grc1 ) ;

mapping1 := representation_map (oldaxes, ebsr );
(* cysptrans is an 80% scaled copy of original rotated about x axis and
  translated along this axis *)
cstrans := mapped_item ('cstrans', mapping1, transform );

(* Define representation that is an assembly of original B-rep +
transformed (scaled and translated and rotated) copy.*)


ebsrass := elementary_brep_shape_representation
         ('ebsrass', [cysp_solid, cstrans], grc2 ) ;

END_REALIZATION;
END_TEST_CASE;
(*
```

### E.3.7.4 Preprocessor verdict criteria

**EB44:** After processing the B-rep solid defined by **mapped_item** shall be correctly scaled and positioned.

**EB45:** After processing absrass should consist of two B-rep solids of cylindrical form with a hemispherical base which are in tangential contact in the plane x = 25. One is a 4/5 size copy of the other after translation and rotation.

## E.4 Contexts defined for test cases of elementary B-rep

The EXPRESS-I contexts below are used in the test cases in clause E.3.

### E.4.1 Cylinder_sphere_shell context

This context provides the faces needed to define a simple **closed_shell** of cylindrical shape with hemispherical base centred at (orc,orc,orc), radius rad and axial height h to oblique face.

All bounds are defined by **edge_loop**s and **conic**s.

```
*)
CONTEXT cylinder_sphere_shell ;
  WITH aic_elementary_brep;
PARAMETER
orc       : length_measure := 0.0;
h         : length_measure := 100.0;
rad       : length_measure := 25.0;
majrad    : length_measure := rad*rt2;
origin : cartesian_point := cartesian_point ('origin', [orc, orc, orc]);
ctop   : cartesian_point := cartesian_point ('ctop', [orc, orc,
                                                      orc + h]);

 pos_x  : direction := direction ('pos_x', [1, 0, 0]);
 pos_y  : direction := direction ('pos_y', [0, 1, 0]);
 pos_z  : direction := direction ('pos_z', [0, 0, 1]);
 dslope : direction := direction ('dslope', [1, 0, -1]);
 dperp  : direction := direction ('dperp', [1, 0, 1]);

 a1     : axis2_placement_3d := axis2_placement_3d ('a1', origin,
                                                    pos_z, pos_x) ;
 a2     : axis2_placement_3d := axis2_placement_3d ('a2', ctop,
                                                   dperp, dslope) ;

 pl     : plane := plane ('pl', a2) ;
```

```
 cyl    : cylindrical_surface := cylindrical_surface ('cyl', a1, rad);
 sphere : spherical_surface := spherical_surface ('sphere', a1, rad);
 circ   : circle := circle ('circ', a1 , rad);
 elli   : ellipse := ellipse('elli', a2 , majrad ,rad);

 cpoint : cartesian_point := cartesian_point ('cpoint', [(orc + rad),
                                                          orc, orc]) ;
 epoint : cartesian_point :=
              cartesian_point ('epoint', [(orc + rad), orc,
                                                  (orc + h - rad)]);

 vertc  : vertex_point := vertex_point ('vertc', cpoint);
 verte  : vertex_point := vertex_point ('verte', epoint);

 edge1  : edge_curve := edge_curve ('edge1', vertc, vertc, circ, TRUE);
 edge2  : edge_curve := edge_curve ('edge2', verte, verte, elli, TRUE);
 oe1    : oriented_edge := oriented_edge ('oe1', edge1, TRUE);
 oe2    : oriented_edge := oriented_edge ('oe2', edge2, TRUE);
 loopc  : edge_loop := edge_loop ('loopc', [oe1]);
 loope  : edge_loop := edge_loop ('loope', [oe2]);

 bc     : face_outer_bound := face_outer_bound ('bc', loopc, FALSE) ;
 be     : face_outer_bound := face_outer_bound ('be', loope, TRUE) ;
 bcylbot : face_bound := face_bound ('bcylbot', loopc, TRUE);
 bcyltop : face_bound := face_bound ('bcyltop', loope, FALSE);

 curved_face : face_surface := face_surface('curved_face', [bcylbot,
                                    bcyltop], cyl, TRUE);
 top_face : face_surface := face_surface ('top_face', [be], pl, TRUE);
 bottom_face : face_surface := face_surface ('bottom_face', [bc],
                                                       sphere, TRUE);
END_PARAMETER;

SCHEMA_DATA cyl_sph_shell_ctxt;
CONSTANT
  rt2 == sqrt(2.0);
  rt3 == sqrt(3.0);
END_CONSTANT;
cfs = connected_face_set {SUBOF(@tri);
               cfs_faces ->  (@curved_face, @top_face, @bottom_face);
                                    SUPOF(@cyspshell);} ;

tri = topological_representation_item {SUBOF(@ri); SUPOF(@cfs);} ;

ri = representation_item {name -> 'cyspshell'; SUPOF(@tri); };
```

```
cyspshell = closed_shell {SUBOF(@cfs); };

END_SCHEMA_DATA;

END_CONTEXT ;
(*
```

## E.4.2 Cone_shell context

This context provides the faces needed to define closed shells of conical shape with circular, elliptic, hyperbolic or parabolic faces. The cone has vertex at (orc,orc,orc), semi-angle 30 degrees and axis parallel to z axis. The normal to each plane face is orthogonal to y axis direction, and is at a fixed angle.

NOTE **plane_angle_units** are required to be degrees.

The dimensions of the resulting shell can be controlled by varying the values of the distances dc, de, dh, dp from the apex, of the intercepts of the planes of the circle, ellipse, hyperbola, parabola respectively with the cone axis.

2 adjacent shells can be defined, a simple conical shell with elliptic base and a more complex conical shape with planar faces elliptic top, circular base and hyperbolic and parabolic sides. Base has 2 straight edges parallel to y axis.

All bounds are defined by **edge_loop**s using **line**s or **conic**s, or by a **vertex_loop**.

```
*)
CONTEXT cone_shell ;
  WITH aic_elementary_brep;

PARAMETER
 orc       : length_measure := 0.0;
 dc        : length_measure := 200.0;
 de        : length_measure := 20.0;
(* Note: dp and dh should be greater than the distance dc to the base
 circle.  To avoid intercepts with top ellipse dh > 7.47 de,
          and  dp > 1.27 de   *)
 dh      : length_measure := 300.0;
 dp      : length_measure := 250.0;
 emaj    : length_measure := de*(rt3/rt2);
 emin    : length_measure := de*(rt2/2.0);
 haxis   : length_measure := 0.5*dh*rt3/rt2;
 himag   : length_measure := dh*sqrt((rt3 - 1.0)/8.0);
 yh      : length_measure := sqrt((rt3 - 1.0)*((2.5 - 1.5*rt3)*dh*dh +
             dc*dh*(3.0*rt3 - 5.0) + 4.0*dc*dc*(2.0 - rt3)/3.0));
(* location points for cone, base circle, ellipse, hyperbola and
```

```
parabola respectively:
*)
origin : cartesian_point := cartesian_point('origin',[orc, orc, orc]);
 cbase  : cartesian_point := cartesian_point ('cbase',
                                          [orc, orc, orc - dc]);
 ecent   : cartesian_point := cartesian_point('ecent',
                             [orc+0.5*de, orc, orc-1.5*de]);
 hcent   : cartesian_point := cartesian_point('hcent',
           [orc+dh*(rt3 + 1.0)/8.0, orc, orc + dh*0.375*(rt3 - 1.0)]);
 ppoint  : cartesian_point := cartesian_point('ppoint',
                            [(orc - 0.5*dp/rt3), orc, (orc - 0.5*dp)]);
 epoint  : cartesian_point := cartesian_point('epoint',
            [orc + 0.5*de*(rt3 + 1.0), orc, orc - 0.5*de*(3.0 + rt3)]);
(* intersection points of conics with plane of base: *)
 ppb1    : cartesian_point := cartesian_point('ppb1', [orc +
   (dc - dp)/rt3, orc  -(0.5*dp/rt3)*sqrt(8.0*dc/dp - dp), orc - dc]);
 ppb2 : cartesian_point := cartesian_point('ppb2',[orc + (dc - dp)/rt3,
              orc + (0.5*dp/rt3)*sqrt(8.0*dc/dp - dp), orc - dc]);
 phb1    : cartesian_point := cartesian_point('phb1',
                   [orc + (dp - dc)*(2.0 - rt3), orc - yh, orc - dc]);
 phb2    : cartesian_point := cartesian_point('phb2',
                   [orc + (dp - dc)*(2.0 - rt3), orc + yh, orc - dc]);

 pos_x   : direction := direction ('pos_x', [1, 0, 0]);
 pos_y   : direction := direction ('pos_y', [0, 1, 0]);
 vec_y   : vector := vector ('vec_y', pos_y, 1.0);
 pos_z   : direction := direction ('pos_z', [0, 0, 1]);
 denorm  : direction := direction ('denorm', [1.0, 0, 1.0]);
 dhnorm  : direction := direction ('dhnorm',
                       [(rt3 + 1.0) , 0, -(rt3 - 1.0)]);
 dpnorm  : direction := direction ('dpnorm', [rt3, 0, 1]);
(* planes of ellipse, parabola and hyperbola are set at angles of 45
degrees, 30 degrees and 15 degrees to axis of cone *)
 dir_e : direction := direction ('dir_e', [1.0, 0, -1.0]);
 dir_h : direction := direction ('dir_h',
                                [-(rt3 - 1.0) , 0, -(rt3 + 1.0)]);
 dir_p : direction := direction ('dir_p', [1, 0, -rt3]);

 a1   : axis2_placement_3d := axis2_placement_3d ('a1', origin,
                                                    pos_z, pos_x) ;
 ac   : axis2_placement_3d := axis2_placement_3d ('ac', cbase, pos_z,
                                                          pos_x);
 ae   : axis2_placement_3d := axis2_placement_3d ('ae', ecent, denorm,
                                                           dir_e);
 ah   : axis2_placement_3d := axis2_placement_3d ('ah', hcent, dhnorm,
                                                          dir_h);
```

```
ap   : axis2_placement_3d := axis2_placement_3d ('ap', ppoint, dpnorm,
                                                              dir_p) ;
a2   : axis2_placement_3d := axis2_placement_3d ('a2', cbase, pos_z,
                                                          pos_x) ;

ple  : plane := plane ('ple', ae) ;
plc  : plane := plane ('plc', ac) ;
plh  : plane := plane ('plh', ah) ;
plp  : plane := plane ('plp', ap) ;
cone : conical_surface := conical_surface ('cone', a1, 0.0, 30.0);

circ  : circle := circle ('circ', ac , (dc/rt3);
elli  : ellipse := ellipse('elli', ae, emaj, emin);
hyp   : hyperbola := hyperbola('hyp', ah, haxis, himag);
parab : parabola := parabola('parab', ap, 0.25*dp/rt3);
linpb : line := line('linpb', ppb1, vec_y);
linph : line := line('linph', phb1, vec_y);

vertorc : vertex_point := vertex_point ('vertorc', origin);
verte   : vertex_point := vertex_point ('verte', epoint);
vertpb1 : vertex_point := vertex_point ('vertpb1', ppb1);
vertpb2 : vertex_point := vertex_point ('vertpb2', ppb2);
verthb1 : vertex_point := vertex_point ('verthb1', phb1);
verthb2 : vertex_point := vertex_point ('verthb2', phb2);

edge1 : edge_curve := edge_curve ('edge1', verte, verte, elli, TRUE);
edge2 : edge_curve := edge_curve ('edge2', vertpb1, vertpb2,
                                                    parab, TRUE);
edge3 : edge_curve := edge_curve ('edge3',verthb1, verthb2, hyp,TRUE);
edgeb1 : edge_curve := edge_curve ('edgeb1', vertpb1, verthb1,
                                                      circ, TRUE);
edgeb2 : edge_curve := edge_curve ('edgeb2', verthb1, verthb2,
                                                     linph, TRUE);
edgeb3 : edge_curve := edge_curve ('edgeb3', verthb2, vertpb2,
                                                       circ, TRUE);
edgeb4 : edge_curve := edge_curve ('edgeb4', vertpb2, vertpb1,
                                                     linpb, FALSE);
oe1     : oriented_edge := oriented_edge ('oe1', edge1, TRUE);
oe2t    : oriented_edge := oriented_edge ('oe2t', edge2, TRUE);
oe2f    : oriented_edge := oriented_edge ('oe2f', edge2, FALSE);
oe3t    : oriented_edge := oriented_edge ('oe3t', edge3, TRUE);
oe3f    : oriented_edge := oriented_edge ('oe3f', edge3, FALSE);
oeb1t   : oriented_edge := oriented_edge ('oeb1t', edgeb1, TRUE);
oeb1f   : oriented_edge := oriented_edge ('oeb1f', edgeb1, FALSE);
oeb2t   : oriented_edge := oriented_edge ('oeb2t', edgeb2, TRUE);
oeb2f   : oriented_edge := oriented_edge ('oeb2f', edgeb2, FALSE);
```

```
 oeb3t   : oriented_edge := oriented_edge ('oeb3t', edgeb3, TRUE);
 oeb3f   : oriented_edge := oriented_edge ('oeb3f', edgeb3, FALSE);
 oeb4t   : oriented_edge := oriented_edge ('oeb4t', edgeb4, TRUE);
 oeb4f   : oriented_edge := oriented_edge ('oeb4f', edgeb4, FALSE);

 loope   : edge_loop := edge_loop ('loope',  [oe1]);
 looppar : edge_loop := edge_loop ('looppar', [oeb4t, oe2t]);
 loophyp : edge_loop := edge_loop ('loophyp', [oeb2t, oe3f]);
 loopbase: edge_loop := edge_loop ('loopbase', [oeb4f, oeb3f,
                                                oeb2f, oeb1f]);
 loopcone: edge_loop := edge_loop ('loopcone', [oe2f, oeb1t,
                                                 oe3t, oeb3t]);
 apexloop: vertex_loop := vertex_loop ('apexloop', vertorc) ;

 bc1    : face_bound := face_bound ('bc1', loopcone, TRUE) ;
 bc2    : face_bound := face_bound ('bc2', loope, FALSE) ;
 be_t   : face_outer_bound := face_outer_bound ('be_t', loope, TRUE);
 be_f   : face_bound := face_bound ('be_f', loope, FALSE) ;
 bpar   : face_outer_bound := face_outer_bound ('bpar', looppar, TRUE);
 bhyp   : face_outer_bound := face_outer_bound ('bhyp', loophyp, TRUE);
 bbase : face_outer_bound := face_outer_bound('bbase', loopbase, TRUE);
 bcone   : face_bound := face_bound ('bcone', loopcone, TRUE) ;
 vbound  : face_bound := face_bound ('vbound', apexloop, TRUE) ;
(* Faces for cone with 4 planar faces *)
  curved_face  : face_surface := face_surface ('curved_face', [bcone,
                                  be_f], cone, TRUE);
  tope_face  : face_surface := face_surface ('tope_face', [be_t],
                                                      ple, TRUE);
  bottomc_face : face_surface := face_surface
                       ('bottomc_face', [bbase],  plc, FALSE);
 par_face    : face_surface :=
                      face_surface ('par_face', [bpar], plp, TRUE);
 hyp_face    : face_surface :=
                      face_surface ('hyp_face', [bhyp], plh, TRUE);

(* Faces for cone with elliptic base, vertex loop at apex *)
  top_face   : face_surface := face_surface
                         ('top_face', [be_t, vbound], cone, TRUE);
  bottome_face : face_surface := face_surface ('bottome_face',
                                               [be_f], ple, FALSE);
END_PARAMETER;

SCHEMA_DATA cone_shell_ctxt;

CONSTANT
  rt2 == sqrt(2.0);
```

```
  rt3 == sqrt(3.0);
END_CONSTANT;

ri1 = representation_item {name -> 'vconeshell'; SUPOF(@tri1);} ;

tri1 = topological_representation_item {SUBOF(@ri1); SUPOF(@cfs1);} ;

cfs1 = connected_face_set {SUBOF(@tri);
                           cfs_faces ->  (@top_face, @bottome_face);
                           SUPOF(@vconeshell); } ;

ri2 = representation_item {name -> 'con4fshell' ; SUPOF(@tri2);} ;

tri2 = topological_representation_item {SUBOF(@ri2); SUPOF(@cfs2);} ;

cfs2 = connected_face_set {SUBOF(@tri2);
        cfs_faces -> (@tope_face, @bottomc_face, curved_face, par_face,
                      hyp_face); SUPOF(@con4fshell); };

vconeshell = closed_shell {SUBOF(@cfs1); };

con4fshell = closed_shell {SUBOF(@cfs2); };

END_SCHEMA_DATA;

END_CONTEXT ;

(*
```

### E.4.3 Toroidal_segment context

This context provides the faces needed to define a segment of a torus bounded by planes. **edge_curve**s are **line**s, circular arcs or **polyline**s.

Torus is centred at origin with z axis as central axis and has major and minor radii of 100 and 20. Bounding planes are z = 0, x = 0 and x = 50. All polyline points are on toroidal surface with tolerance of less than 10E(-6).

All bounds are defined by **edge_loop**s.

Basic dimensional parameters should not be varied.

```
*)
CONTEXT toroidal_segment ;
  WITH aic_elementary_brep;
PARAMETER
```

```
orc       : length_measure := 0.0;
rad1      : length_measure := 100.0;
rad2      : length_measure := 20.0;
rci       : length_measure := 80.0;
rco       : length_measure := 120.0;
origin    : cartesian_point := cartesian_point ('origin', [orc, orc,
                                                  orc]);
p1        : cartesian_point := cartesian_point ('p1', [50.0, 62.44998,
                                                    0.0]);
pcleft    : cartesian_point := cartesian_point ('pcleft', [0.0, 100.0,
                                                    0.0]);
pbleft    : cartesian_point := cartesian_point ('pbleft', [0.0, 80.0,
                                                    0.0]);
ptleft    : cartesian_point := cartesian_point ('ptleft', [0.0, 120.0,
                                                     0.0]);

pos_x     : direction := direction ('pos_x', [1, 0, 0]);
pos_y     : direction := direction ('pos_y', [0, 1, 0]);
pos_z     : direction := direction ('pos_z', [0, 0, 1]);
neg_x     : direction := direction ('neg_x', [-1, 0, 0]);
vec_y     : vector := vector ('vec_y', pos_y, 1.0) ;

a1        : axis2_placement_3d := axis2_placement_3d ('a1', origin,
                                                    pos_z, pos_x) ;
a2        : axis2_placement_3d := axis2_placement_3d ('a2', pcleft,
                                                    neg_x, pos_y) ;
a3        : axis2_placement_3d := axis2_placement_3d ('a3', p1,
                                                    pos_x, pos_z) ;

base   : plane := plane ('base', a1) ;
pleft  : plane := plane ('pleft', a2) ;
pright : plane := plane ('pright', a3) ;
torus  : toroidal_surface := toroidal_surface ('torus', a1, rad1,
                                                        rad2);

circin    : circle := circle ('circin', a1 , rci);
circout   : circle := circle ('circout', a1 , rco);
circleft  : circle := circle ('circleft', a2 , rad2);

p2        : cartesian_point :=
                cartesian_point ('p2', [50.0, 62.633918, 2.392932]);
p3        : cartesian_point :=
                cartesian_point ('p3', [50.0, 64.92632, 8.609]);
p4        : cartesian_point :=
                cartesian_point ('p4', [50.0, 67.325057, 11.8123625]);
p5        : cartesian_point :=
```

```
                cartesian_point ('p5', [50.0, 69.839261, 14.1766914]);
p6        : cartesian_point :=
                cartesian_point ('p6', [50.0, 72.479126, 16.03916]);
p7        : cartesian_point :=
                cartesian_point ('p7', [50.0, 75.25605, 17.518988]);
p8        : cartesian_point :=
                cartesian_point ('p8', [50.0, 78.182821, 18.660529]);
p9        : cartesian_point :=
                cartesian_point ('p9', [50.0, 81.27384, 19.469096]);
p10       : cartesian_point :=
              cartesian_point ('p10', [50.0, 84.5453828, 19.920975]);
p11       : cartesian_point :=
             cartesian_point ('p11', [50.0, 88.0159173, 19.9623578]);
p12        : cartesian_point :=
               cartesian_point ('p12', [50.0, 91.706487, 19.498351]);
p13        : cartesian_point :=
              cartesian_point ('p13', [50.0, 95.6411789, 18.343994]);
p14        : cartesian_point :=
               cartesian_point ('p14', [50.0, 99.8476928, 16.24428]);
p15        : cartesian_point :=
              cartesian_point ('p15', [50.0, 104.3580503, 12.3673625]);
p16        : cartesian_point :=
               cartesian_point ('p16', [50.0, 107.225346, 8.046179]);
p17        : cartesian_point :=
               cartesian_point ('p17', [50.0, 109.008344, 1.692583]);
p18        : cartesian_point :=
                cartesian_point ('p18', [50.0, 109.0871212, 0.0]);

poly       : polyline := polyline('poly', [p1, p2, p3,
                                           p4, p5, p6, p7, p8, p9,
                      p10, p11, p12, p13, p14, p15, p16, p17, p18]);
l1         : line := line ('l1', p1, vec_y);
l2         : line := line ('l2', pcleft, vec_y);

v1         : vertex_point := vertex_point ('v1', p1);
v2         : vertex_point := vertex_point ('v2', p18);
v3         : vertex_point := vertex_point ('v3', pbleft);
v4         : vertex_point := vertex_point ('v4', ptleft);

edgeb1 : edge_curve := edge_curve ('edgeb1', v1, v2, l1, TRUE);
edget1 : edge_curve := edge_curve ('edget1', v1, v2, poly, TRUE);
edgeb2 : edge_curve := edge_curve ('edgeb2', v1, v3, circin, TRUE);
edgeb3 : edge_curve := edge_curve ('edgeb3', v2, v4, circout, TRUE);
edgeb4 : edge_curve := edge_curve ('edgeb4', v3, v4, l2, TRUE);
edget2 : edge_curve := edge_curve ('edget2', v3, v4, circleft, TRUE);
```

```
 oeb1t   : oriented_edge := oriented_edge ('oeb1t', edgeb1, TRUE);
 oeb1f   : oriented_edge := oriented_edge ('oeb1f', edgeb1, FALSE);
 oeb2t   : oriented_edge := oriented_edge ('oeb2t', edgeb2, TRUE);
 oeb2f   : oriented_edge := oriented_edge ('oeb2f', edgeb2, FALSE);
 oeb3t   : oriented_edge := oriented_edge ('oeb3t', edgeb3, TRUE);
 oeb3f   : oriented_edge := oriented_edge ('oeb3f', edgeb3, FALSE);
 oeb4t   : oriented_edge := oriented_edge ('oeb4t', edgeb4, TRUE);
 oeb4f   : oriented_edge := oriented_edge ('oeb4f', edgeb4, FALSE);
 oet1t   : oriented_edge := oriented_edge ('oet1t', edget1, TRUE);
 oet1f   : oriented_edge := oriented_edge ('oet1f', edget1, FALSE);
 oet2t   : oriented_edge := oriented_edge ('oet2t', edget2, TRUE);
 oet2f   : oriented_edge := oriented_edge ('oet2f', edget2, FALSE);

 loopb  : edge_loop := edge_loop ('loopb', [oeb4t, oeb3f, oeb1f,
                                                        oeb2t]);
 loopt  : edge_loop := edge_loop ('loopt', [oeb2f, oet1t, oeb3t,
                                                        oet2f]);
 loopleft  : edge_loop := edge_loop ('loopleft', [oeb4f, oet2t]);
 loopright : edge_loop := edge_loop ('loopright', [oeb1t, oet1f]);

 bbase : face_outer_bound := face_outer_bound ('bbase', loopb, TRUE) ;
 btop   : face_outer_bound := face_outer_bound ('btop', loopt, TRUE) ;
 bleft  : face_outer_bound := face_outer_bound ('bleft', loopleft,
                                                          TRUE) ;
 bright : face_outer_bound := face_outer_bound ('bright', loopright,
                                                          TRUE) ;


 curved_face : face_surface := face_surface ('curved_face', [btop],
                                                    torus, TRUE);
 base_face   : face_surface :=
                     face_surface ('base_face', [bbase], base, FALSE);
 left_face : face_surface :=
                    face_surface ('left_face', [bleft],  pleft, TRUE);
 right_face : face_surface :=
                   face_surface ('right_face', [bright],  pright, TRUE);
END_PARAMETER;

SCHEMA_DATA tor_shell_ctxt;

ri = representation_item {name -> 'torshell' ; SUPOF(@tri);} ;

tri = topological_representation_item {SUBOF(@ri); SUPOF(@cfs);} ;

cfs = connected_face_set {SUBOF(@tri);
     cfs_faces -> (@curved_face, @base_face, @left_face, @right_face);
```

```
                                SUPOF(@torshell); };

torshell = closed_shell {SUBOF(@cfs);} ;

END_SCHEMA_DATA;

END_CONTEXT ;

(*
```

### E.4.4 Cylinder_union_polyline context

This context provides the faces needed to define the faces of a union of two cylinders of differing radii.

It provides an example of a non-planar **polyline** and of a **face** with 3 bounding loops.

All bounds are defined by **edge_loop**s. Basic dimensional parameters should not be varied.

```
*)
CONTEXT cylinder_union_polyline ;
  WITH aic_elementary_brep;
PARAMETER
 orc       : length_measure := 0.0;
 rad1      : length_measure := 50.0;
 rad2      : length_measure := 20.0;
 l1        : length_measure := 80.0;
 l2        : length_measure := -80.0;

origin : cartesian_point := cartesian_point ('origin',[orc, orc, orc]);
ptop   : cartesian_point := cartesian_point ('ptop', [orc, orc, l1]);
pbase  : cartesian_point := cartesian_point ('pbase', [orc, orc, l2]);
pright : cartesian_point := cartesian_point ('pright', [orc, l1, orc]);
pte    : cartesian_point := cartesian_point ('pte', [rad1, orc, l1]);
pbe    : cartesian_point := cartesian_point ('pbe', [rad1, orc, l2]);
pre    : cartesian_point := cartesian_point ('pre', [rad2 , l1, orc]);

pos_x  : direction := direction ('pos_x', [1, 0, 0]);
pos_y  : direction := direction ('pos_y', [0, 1, 0]);
pos_z  : direction := direction ('pos_z', [0, 0, 1]);

a1     : axis2_placement_3d := axis2_placement_3d ('a1', origin, pos_z,
                                                   pos_x) ;
a2     : axis2_placement_3d := axis2_placement_3d ('a2', origin, pos_y,
                                                   pos_x) ;
at     : axis2_placement_3d := axis2_placement_3d ('at', ptop, ?, ? );
ab     : axis2_placement_3d := axis2_placement_3d ('ab', pbase, ?, ?);
```

```
ar     : axis2_placement_3d := axis2_placement_3d ('ar', pright,
                                                   pos_y, pos_x) ;

base    : plane := plane ('base', ab) ;
top     : plane := plane ('top', at) ;
plright : plane := plane ('plright', ar) ;
cyl1  : cylindrical_surface := cylindrical_surface ('cyl1', a1, rad1);
cyl2  : cylindrical_surface := cylindrical_surface ('cyl2', a2, rad2);

circtop  : circle := circle ('circtop', at , rad1);
circbase  : circle := circle ('circbase', ab , rad1);
circright : circle := circle ('circright', ar , rad2);

p1    : cartesian_point :=
            cartesian_point ('p1', [0.0, 50.0, 20.0]);
p2    : cartesian_point :=
          cartesian_point ('p2', [3.4729636, 49.8792394, 19.6961551]);
p3    : cartesian_point :=
            cartesian_point ('p3', [6.8404029, 49.529879, 18.793852]);
p4    : cartesian_point :=
            cartesian_point ('p4', [10.0, 48.9897949, 17.3205081]);
p5    : cartesian_point :=
            cartesian_point ('p5', [12.8557522, 48.31904, 15.320889]);
p6    : cartesian_point :=
          cartesian_point ('p6', [15.3208889, 47.5948565, 12.8557522]);
p7    : cartesian_point :=
          cartesian_point ('p7', [17.3205081, 46.904158, 10.0]);
p8    : cartesian_point :=
          cartesian_point ('p8', [18.7938524, 46.3334772, 6.84040287]);
p9    : cartesian_point :=
            cartesian_point ('p9', [19.6961551, 45.95717, 3.4729635]);
p10   : cartesian_point :=
            cartesian_point ('p10', [20.0, 45.8257569, 0.0]);
p11   : cartesian_point :=
          cartesian_point ('p11', [19.6961551, 45.95717, -3.4729635]);
p12   : cartesian_point :=
         cartesian_point ('p12', [18.7938524, 46.3334772, -6.84040287]);
p13   : cartesian_point :=
            cartesian_point ('p13', [17.3205081, 46.904158, -10.0]);
p14   : cartesian_point :=
         cartesian_point ('p14', [15.3208889, 47.5948565, -12.8557522]);
p15   : cartesian_point :=
          cartesian_point ('p15', [12.8557522, 48.31904, -15.320889]);
p16   : cartesian_point :=
            cartesian_point ('p16', [10.0, 48.9897949, -17.3205081]);
p17   : cartesian_point :=
```

```
         cartesian_point ('p17', [6.8404029, 49.529879, -18.793852]);
p18   : cartesian_point :=
        cartesian_point ('p18', [3.4729636, 49.8792394, -19.6961551]);
p19   : cartesian_point :=
           cartesian_point ('p19', [0.0, 50.0, -20.0]);
p20   : cartesian_point :=
       cartesian_point ('p20', [-3.4729636, 49.8792394, -19.6961551]);
p21   : cartesian_point :=
         cartesian_point ('p21', [-6.8404029, 49.529879, -18.793852]);
p22   : cartesian_point :=
           cartesian_point ('p22', [-10.0, 48.9897949, -17.3205081]);
p23   : cartesian_point :=
         cartesian_point ('p23', [-12.8557522, 48.31904, -15.320889]);
p24   : cartesian_point :=
       cartesian_point ('p24', [-15.3208889, 47.5948565, -12.8557522]);
p25   : cartesian_point :=
           cartesian_point ('p25', [-17.3205081, 46.904158, -10.0]);
p26   : cartesian_point :=
       cartesian_point ('p26', [-18.7938524, 46.3334772, -6.84040287]);
p27   : cartesian_point :=
         cartesian_point ('p27', [-19.6961551, 45.95717, -3.4729635]);
p28   : cartesian_point :=
           cartesian_point ('p28', [-20.0, 45.8257569, 0.0]);
p29   : cartesian_point :=
         cartesian_point ('p29', [-19.6961551, 45.95717, 3.4729635]);
p30   : cartesian_point :=
      cartesian_point ('p30', [-18.7938524, 46.3334772, 6.84040287]);
p31   : cartesian_point :=
           cartesian_point ('p31', [-17.3205081, 46.904158, 10.0]);
p32   : cartesian_point :=
      cartesian_point ('p32', [-15.3208889, 47.5948565, 12.8557522]);
p33   : cartesian_point :=
           cartesian_point ('p33', [-12.8557522, 48.31904, 15.320889]);
p34   : cartesian_point :=
           cartesian_point ('p34', [-10.0, 48.9897949, 17.3205081]);
p35   : cartesian_point :=
           cartesian_point ('p35', [-6.8404029, 49.529879, 18.793852]);
p36   : cartesian_point :=
         cartesian_point ('p36', [-3.4729636, 49.8792394, 19.6961551]);

poly  : polyline := polyline ('poly',  [p1, p2, p3, p4, p5, p6,
                  p7, p8, p9, p10, p11, p12, p13, p14, p15, p16, p17,
                  p18, p19, p20, p21, p22, p23, p24, p25, p26, p27,
                  p28, p29, p30, p31, p32, p33, p34, p35, p36, p1]);

v1    : vertex_point := vertex_point ('v1',  p1);
```

```
v2    : vertex_point := vertex_point ('v2', pte);
v3    : vertex_point := vertex_point ('v3', pbe);
v4    : vertex_point := vertex_point ('v4', pre);

edgem0  : edge_curve := edge_curve ('edgem0', v1, v1, poly, TRUE);
edget1  : edge_curve := edge_curve ('edget1', v2, v2, circtop, TRUE);
edgeb2  : edge_curve := edge_curve ('edgeb2', v3, v3, circbase, TRUE);
edger3  : edge_curve := edge_curve ('edger3', v4, v4, circright, TRUE);

oem0t   : oriented_edge := oriented_edge ('oem0t', edgem0, TRUE);
oem0f   : oriented_edge := oriented_edge ('oem0f', edgem0, FALSE);
oet1t   : oriented_edge := oriented_edge ('oet1t', edget1, TRUE);
oet1f   : oriented_edge := oriented_edge ('oet1f', edget1, FALSE);
oeb2t   : oriented_edge := oriented_edge ('oeb2t', edgeb2, TRUE);
oeb2f   : oriented_edge := oriented_edge ('oeb2f', edgeb2, FALSE);
oer3t   : oriented_edge := oriented_edge ('oer3t', edger3, TRUE);
oer3f   : oriented_edge := oriented_edge ('oer3f', edger3, FALSE);

loopb  : edge_loop := edge_loop ('loopb', [oeb2f]);
loopt  : edge_loop := edge_loop ('loopt', [oet1t]);
loopmidt : edge_loop := edge_loop ('loopmidt', [oem0t]);
looprt   : edge_loop := edge_loop ('looprt', [oer3t]);
loopbt   : edge_loop := edge_loop ('loopbt', [oeb2t]);
looptf   : edge_loop := edge_loop ('looptf', [oet1f]);
loopmidf : edge_loop := edge_loop ('loopmidf', [oem0f]);
looprf   : edge_loop := edge_loop ('looprf', [oer3f]);

bbase  : face_outer_bound := face_outer_bound ('bbase', loopb, TRUE);
btop   : face_outer_bound := face_outer_bound ('btop', loopt, TRUE) ;
bright : face_outer_bound := face_outer_bound ('bright', looprt,TRUE);
bmidt  : face_bound := face_bound ('bmidt', loopmidt, TRUE) ;
bmidf  : face_bound := face_bound ('bmidf', loopmidf, TRUE) ;
bcy1top : face_bound :=  face_bound ('bcy1top', looptf, TRUE) ;
bcy1b  : face_bound :=  face_bound ('bcy1b', loopbt, TRUE) ;
bcy1m  : face_bound :=  face_bound ('bcy1m', loopmidt, TRUE) ;
bcy2m  : face_bound :=  face_bound ('bcy2m', loopmidf, TRUE) ;
bcy2r  : face_bound :=  face_bound ('bcy2r', looprf, TRUE) ;

cyl_face1  : face_surface := face_surface ('cyl_face1',
                                [bcy1top, bcy1m, bcy1b], cyl1, TRUE);
cyl_face2  : face_surface := face_surface ('cyl_face2',
                               [bcy2m, bcy2r], cyl2, TRUE);
base_face  : face_surface :=
                     face_surface ('base_face', [bbase], base, FALSE);
top_face   : face_surface :=
             face_surface ('top_face', 'top_face', [btop],  top, TRUE);
```

```
 right_face : face_surface :=
               face_surface (’right_face’, [bright],  plright, TRUE);
END_PARAMETER;

SCHEMA_DATA cyl_un_poly_ctxt;

ricx = representation_item {name -> ’cxcshell’ ; SUPOF(@tricx);} ;

tricx = topological_representation_item {SUBOF(@ricx); SUPOF(@cfscx);};

cfscx = connected_face_set {SUBOF(@tricx);
        cfs_faces -> (@cyl_face1, @cyl_face2, @base_face, @top_face,
                              @right_face); SUPOF(@csxshell); } ;

cxcshell = closed_shell {SUBOF(@cfscx);} ;
END_SCHEMA_DATA;

END_CONTEXT ;

(*
```

# Index



**ICS 25.040.40**

Price based on 56 pages