TECHNICAL SPECIFICATION

ISO/TS 10303-304

First edition
2001-03-01

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

Part 304: Abstract test suite: Mechanical design using boundary representation

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

Partie 304: Suite d'essais abstraite: Conception mécanique utilisant une représentation de limite

Reference number
ISO/TS 10303-304:2001(E)

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

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.

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

In other circumstances, particularly when there is an urgent market requirement for such documents, a technical committee may decide to publish other types of normative document:

— an ISO Publicly Available Specification (ISO/PAS) represents an agreement between technical experts in an ISO working group and is accepted for publication if it is approved by more than 50% of the members of the parent committee casting a vote;

— an ISO Technical Specification (ISO/TS) represents an agreement between technical experts in an ISO working group and is accepted for publication if it is approved by 2/3 of the members of the parent committee casting a vote.

An ISO/PAS or ISO/TS is reviewed every three years with a view to deciding whether it can be transformed into an International Standard.

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/TS 10303–304 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC4, *Industrial data*.

This International Standard is organised as a series of parts, each published separately. The structure of this International Standard is described in ISO 10303-1.

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

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

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

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

The preparation of this part of ISO 10303 has benefitted from the technical contributions of many projects and their sponsoring organizations. The contributions of the following are acknowledged:

— Esprit project 6040 Prodex.

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 abstract test suite series.

The purpose of an abstract test suite is to provide a basis for evaluating whether a particular implementation of an application protocol actually conforms to the requirements of that application protocol. A standard abstract test suite helps ensure that evaluations of conformance are conducted in a consistent manner by different test laboratories.

This part of ISO 10303 specifies the abstract test suite for ISO 10303-204, application protocol Mechanical design using boundary representation. The abstract test cases presented here are the basis for conformance testing of implementations of ISO 10303-204.

This abstract test suite is made up of two major parts:

— the test purposes, the specific items to be covered by conformance testing;

— the set of abstract test cases that meet those test purposes.

The test purposes are statements of the application protocol requirements that are to be addressed by the abstract test cases. Test purposes are derived primarily from the application protocol's application elements and application interpreted model, as well as from other sources such as standards referenced by the application protocol and requirements stated in the application protocol conformance requirements clause.

The abstract test cases address the test purposes by:

— specifying the requirements for input data to be used when testing an implementation of the application protocol;

— specifying the verdict criteria to be used when evaluating whether the implementation successfully converted the input data to a different form.

The abstract test cases set the requirements for the executable test cases that are required to actually conduct a conformance test. Executable test cases contain the scripts, detailed values, and other ex-

plicit information required to conduct a conformance test on a specific implementation of the application protocol.

At the time of publication of this document, conformance testing requirements had been established for implementations of application protocols in combination with ISO 10303–21 and ISO 10303–22. Accordingly, this part of ISO 10303 only specifies test purposes and abstract test cases appropriate to such implementations.

For ISO 10303–21, two kinds of implementations, preprocessors and postprocessors, must be tested. Both these are addressed in this abstract test suite.

For ISO 10303–22, a class of applications will possess the capability to upload and download application protocol-compliant standard data access interface-models and/or schema instances to and from applications that implement the standard data access interface. This abstract test suite addresses such applications.

The abstract test cases presented here are the basis for conformance testing of implementations of ISO 10303-204. The test cases in this part of ISO 10303 are documented in the EXPRESS-I language and can potentially be readily adapted to other implementation methods.

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

Part 304: Abstract test suite: Mechanical design using boundary representation

1 Scope

This part of ISO 10303 specifies the abstract test suite to be used in the conformance testing of implementations of ISO 10303-204. The following are within the scope of this part of ISO 10303:

— the specification of the test purposes associated with ISO 10303-204;

— the verdict criteria to be applied during conformance testing of an implementation of ISO 10303-204 using ISO 10303-21 or ISO 10303-22;

— the abstract test cases to be used as the basis for the executable test cases for conformance testing.

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

— the creation of executable test cases;

— testing other than conformance testing;

— other implementation methods.

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

ISO 10303-22:1998, *Industrial automation systems and integration - Product data representation and exchange - Part 22 : Implementation methods: Standard data access interface.*

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

ISO 10303-32:1998, *Industrial automation systems and integration - Product data representation and exchange - Part 32: Conformance testing methodology and framework: Requirements on testing laboratories and clients.*

ISO 10303-34:2001, *Industrial automation systems and integration - Product data representation and exchange - Part 34: Conformance testing methodology and framework: Abstract test methods for application protocol implementations.*

ISO 10303-204[1)], *Industrial automation systems and integration - Product data representation and exchange - Part 204 : Application protocol: Mechanical design using boundary representation.*

ISO 10303-511:2001, *Industrial automation systems and integration - Product data representation and exchange - Part 511 : Application interpreted construct: Topologically bounded surface*

ISO 10303-512:1999, *Industrial automation systems and integration - Product data representation and exchange - Part 512 : Application interpreted construct: Faceted boundary representation*

ISO 10303-513:2000, *Industrial automation systems and integration - Product data representation and exchange - Part 513 : Application interpreted construct: Elementary boundary representation*

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

ISO 10303-517:2000, *Industrial automation systems and integration - Product data representation and exchange - Part 517 : Application interpreted construct: Mechanical design geometric presentation*

ISO 10303-518[1)], *Industrial automation systems and integration - Product data representation and exchange - Part 518 : Application interpreted construct: Mechanical design shaded presentation*

ISO/IEC 8824-1:1998, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

[1)] To be published.

3 Definitions

3.1 Terms defined in ISO 10303-1

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

— abstract test suite;

— application;

— application activity model (AAM);

— application context;

— application interpreted model (AIM);

— application object;

— application protocol (AP);

— application reference model (ARM);

— conformance class;

— conformance requirement;

— context;

— data;

— data exchange;

— implementation method;

— interpretation;

— model;

— product data;

— unit of functionality (UoF).

3.2 Terms defined in ISO 10303-31

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

— abstract test case;

— conformance testing;

— executable test case;

— executable test suite;

— test purpose;

— (test) verdict;

— verdict criterion.

3.3 Terms defined in ISO 10303-204

This part of ISO 10303 makes use of the following terms defined in ISO 10303-204.

— advanced B-rep;

— elementary B-rep;

— faceted B-rep.

3.4 Other definitions

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

3.4.1
minimal entity set
the set of entities which shall be present in every instantiated model under this application protocol.

3.5 Abbreviations

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

AIC: Application Interpreted Construct;

B-rep: Boundary representation solid;

IP: Informal Proposition;

IUT: Implementation Under Test;

UoF: Unit of Functionality.

4 Test purposes

This clause specifies the test purposes for this part of ISO 10303. Test purposes are derived from the information requirements contained in clause 4 of ISO 10303-204, the AIM EXPRESS schema in annex A of ISO 10303-204, and AIC parts referenced by ISO 10303-204. The test purposes are organized in this clause by type.

AE test purposes are individually identified by the prefix “ae” in the test purpose number. Each test purpose derived from the information requirements shall be interpreted as:

> Correctly instantiate in the implementation under test the semantic associated with the unique application concept corresponding to *(insert test purpose here)* in at least one test case within the test suite.

AE test purposes apply to the input specifications of both preprocessor and postprocessor test cases. AE test purposes are derived from the AP information requirements as follows:

— application objects (4.2 of ISO 10303-204). A test purpose derived from an application object is a simple statement of the object’s name. Each application object test purpose is documented in a separate subclause.

— application objects with categorisations (subtypes) (4.2 of ISO 10303-204). Test purposes derived from application objects with categorisations are statements of the application object name as a specific subtype.

— application object attributes (4.2 of ISO 10303-204). Test purposes derived from application object attributes are statements of the application object name with a specific attribute name.

— application asertions (4.3 of ISO 10303-204). Test purposes derived from application assertions are statements describing the relationship between two application objects. Application assertion test purposes address the directions of relationships as well as the number (cardinality) of relationships.

Each application object test purpose is listed as a separate subclause, with its related application object attribute test purposes. The application assertion test purposes form another subclause.

AIM test purposes are identified by the prefix “aim” in the test purpose number. Each test purpose derived from the AIM EXPRESS shall be interpreted as follows:

> Correctly instntiate in the implementation under test the AIM element associated with the unique AIM entity corresponding to *(insert test purpose here)* in at least one test case within the test suite.

AIM test purposes apply to the input specifications of postprocessor test cases only. AIM test purposes are derived directly from the expanded EXPRESS listing contained in annex A of ISO 10303-204 as follows:

— AIM entities. A test purpose derived from an AIM element is a simple statement of the entity name.

— AIM entity attributes. Test purposes derived from AIM entity attributes are statements of the AIM entity with a given attribute.

Each AIM entity test purpose is grouped with its attribute test purposes, all of which are contained in 4.2.

Other test purposes are derived from implicit domain requirements, rule checking requirements and from the AICs referenced by ISO 10303-204. Each other test purpose is a statement of some requirement that shall be met by any conforming implementation. Other test purposes are grouped in clause 4.4 and later.

AE and AIM test purposes are individually identified by the prefix 'ae' or 'AIM' and a number which relates to the UoF of origin. The codes used for UoF identification are given below.

Table 1 – Test purpose source identification

Source UoF	AE test purpose number	AIM identifier code
Advanced B-rep UoF and AIC	ae2 - ae99	aim8 - aim105, aim107 - aim172
Elementary B-rep UoF and AIC	ae100 - ae199	aim8 - aim106
Faceted B-rep UoF and AIC	ae200 - ae299	aim1 - aim7
Name Preservation UoF	ae300 - ae399	aim173 - aim198
Product Structure UoF	ae400 - ae499	aim199 - aim295
Visual Presentation for B-rep UoF	ae500 - ae599	aim296 - aim441

NOTE 1 Many of the AE test purposes for the advanced B-rep UoF are also applicable to the elementary B-rep UoF. In such cases a multiple reference is given to the relevant test cases.

4.1 Application element test purposes

AE test purposes are individually identified by the prefix "ae" in the test purpose number. Each test purpose derived from the information requirements shall be interpreted as:

Correctly instantiate in the implementation under test the semantic associated with the unique application concept corresponding to (*insert test purpose here*) in at least one test case within the test suite.

Correctly instantiating an information requirement implies that the semantics of the application element are preserved between the input and the output of a test as well as conformance to the reference path specified in the mapping table of the AP. AE test purposes apply to the input specifications of both preprocessor and postprocessor test cases. AE test purposes are derived from the AP information requirements as follows:

— application objects (4.2 of ISO 10303-204). A test purpose derived from an application object is a simple statement of the object's name. Each application object test purpose is documented in a separate subclause.

— application objects with categorisations (subtypes) (4.2 of ISO 10303-204). Test purposes derived from application objects with categorisations are statements of the application object name as a specific subtype.

— application object attributes (4.2 of ISO 10303- 204). Test purposes derived from application object attributes are statements of the application object name with a specific attribute name.

— application assertions (4.3 of ISO 10303-204). Test purposes derived from application assertions are statements describing the relationship between two application objects. Application assertion test purposes address the directions of relationships as well as the number (cardinality) of relationships.

Each application object test purpose is listed as a separate subclause, with its related application object attribute test purposes and assertion test purposes.

4.1.1 3D_Projection

ae501 3D_projection as camera_model_3d. (see 6.6.1 to 6.6.7)

ae518 3D_projection presenting zero B-rep model objects. (see 6.6.5)

ae519 3D_projection presenting one B-rep model. (see 6.6.1 to 6.6.7)

ae520 3D_projection presenting many B-rep model objects.

4.1.2 Advanced_B-rep

ae2 **advanced_brep_shape_representation**. (see 6.3.1)

4.1.3 Assembly

ae401 Assembly as representation. (see 6.5.4)

ae402 Assembly with user_defined_name. (see 6.5.4)

ae403 Assembly with coordinate_system. (see 6.5.4)

ae412 Assembly containing zero Assembly objects (sub-assemblies). (see 6.5.4)

ae413 Assembly containing one assembly in role of sub-assembly. (see 6.5.4)

ae414 Assembly containing more than one assembly in roles of sub-assembly.

ae415 Assembly as part of one product. (see 6.5.4)

ae416 Assembly as part of more than one product. (see 6.5.4)

ae418 Assembly containing one part. (see 6.5.4)

ae419 Assembly containing many parts with associated transformation objects. (see 6.5.4)

ae428 Assembly with sub-assembly located by transformation.

ae419 Assembly containing many parts with associated transformation objects. (see 6.5.4)

4.1.4 B-rep

ae3 B-rep as an Advanced_B-rep. (see 6.3.1)

ae100 B-rep as an Elementary_B-rep. (see 6.2.1)

ae200 B-rep as a Faceted_B-rep. (see 6.1.1)

ae521 B-rep model associated with zero 3D_projection objects. (see 6.6.5)

ae522 B-rep model associated with one 3D_projection. (see 6.6.1 to 6.6.7)

ae523 B-rep model associated with many 3D_projection objects.

ae73 B-rep consisting of one closed_shell as outer shell. (see 6.3.1, 6.2.1, 6.1.1)

ae56 B-rep consisting of more than one closed_shell, one closed_shell as outer shell. (see 6.3.2, 6.2.2, 6.1.2)

ae60 B-rep with edge with sense true and same edge with sense false. (see 6.3.1, 6.2.1)

ae539 B-rep model presented as zero presentation_appearance objects. (see 6.6.5)

ae601 B-rep model presented as one presentation_appearance objects.

ae539b B-rep model presented as many presentation_appearance objects.

ae74 B-rep in advanced_brep_shape_representation. (see 6.3.1)

ae113 B-rep in elementary_brep_shape_representation. (see 6.2.1)

ae206 B-rep in faceted_brep_shape_representation. (see 6.1.1)

4.1.5 Bounded_curve

ae114 Bounded_curve as Polyline. (see 6.3.4, 6.2.4)

ae5 Bounded_curve as a Twisted_curve. (see 6.3.5)

4.1.6 Circle

ae6 Circle. (see 6.3.1, 6.2.1)

ae7 Circle with location. (see 6.3.1, 6.2.1)

ae8 Circle with radius.(see 6.3.1, 6.2.1)

4.1.7 Closed_shell

ae9 Closed_shell. (see 6.3.1, 6.2.1)

ae10 Closed_shell with one face. (see 6.3.2, 6.2.2)

ae11 Closed_shell with set of more than one faces. (see 6.3.1, 6.2.1, 6.1.1)

4.1.8 Conic

ae12 Conic as Circle. (see 6.3.1, 6.2.1)

ae13 Conic as Ellipse. (see 6.3.1, 6.2.1)

ae14 Conic as Hyperbola. (see 6.3.6, 6.2.5)

ae15 Conic as Parabola. (see 6.3.6, 6.2.5)

4.1.9 Conical_surface

ae16 Conical_surface. (see 6.3.6, 6.2.5)

4.1.10 Curve

ae17 Curve as Bounded_curve. (see 6.3.4, 6.2.4)

ae18 Curve as conic. (see 6.3.1, 6.2.1)

ae19 Curve as line. (see 6.3.3, 6.2.3)

ae58 Curve used to define the geometry of one edge. (see 6.3.1, 6.2.1)

ae59 Curve used to define the geometry of more than one edge. (see 6.3.15)

ae527 Curve represented as zero curve_appearance objects. (see 6.6.5)

ae602 Curve represented as one curve_appearance.

4.1.11 Curve_appearance

ae502 Curve_appearance as curve_style. (see 6.6.2)

ae503 Curve_appearance with colour as colour_rgb. (see 6.6.2)

ae504 Curve_appearance with curve_font. (see 6.6.5)

ae505 Curve_appearance with curve_width. (see 6.6.5)

ae603 Curve_appearance with default colour.

ae604 Curve_appearance with unspecified curve_font.

ae605 Curve_appearance with unspecified curve_width.

ae524 Curve_appearance associated with zero curve objects. (see 6.6.5)

ae525 Curve_appearance associated with one curve.

ae526 Curve_appearance associated with many curve objects. (see 6.6.1 to 6.6.7)

ae528 Curve_appearance associated with zero edge objects. (see 6.6.5)

ae529 Curve_appearance associated with one edge.

ae530 Curve_appearance associated with many edge objects. (see 6.6.1 to 6.6.7)

4.1.12 Cylindrical_surface

ae20 Cylindrical_surface. (see 6.3.1, 6.2.1)

ae21 Cylindrical_surface with axis. (see 6.3.1, 6.2.1)

ae22 Cylindrical_surface with radius. (see 6.3.1, 6.2.1)

4.1.13 Degenerate_toroidal_surface

ae606 Degenerate_toroidal_surface. (see 6.3.12)

4.1.14 Direction

ae23 Direction as a unit vector in 3 dimensional space. (see 6.3.1, 6.2.1)

4.1.15 Edge

ae24 Edge with edge_geometry as curve. (see 6.3.1, 6.2.1)

ae25 Edge with two vertices. (see 6.3.1, 6.2.1)

ae531 Edge represented as zero curve_appearance objects. (see 6.6.5)

ae607 Edge represented as one curve_appearance.

ae61 Edge connected to another edge by vertex. (see 6.3.6, 6.2.5)

ae69 Edge used to define one loop in a B-rep. (see 6.3.1, 6.2.1)

ae70 Edge used to define two loops in a B-rep. (see 6.3.6, 6.2.5)

4.1.16 Elementary_B-rep

ae101 Elementary_B-rep as elementary_brep_shape_representation. (see 6.2.1)

4.1.17 Elementary_surface

ae102 Elementary_surface as Plane. (see 6.2.1, 6.3.1)

ae103 Elementary_surface as Conical_surface. (see 6.2.5, 6.3.6)

ae104 Elementary_surface as Cylindrical_surface. (see 6.2.1, 6.3.1)

ae105 Elementary_surface as Spherical_surface. (see 6.2.1, 6.3.1)

ae106 Elementary_surface as Toroidal_surface. (see 6.2.3, 6.3.3)

4.1.18 Ellipse

ae26 Ellipse. (see 6.3.1, 6.2.1)

ae27 Ellipse with location. (see 6.3.1, 6.2.1)

ae28 Ellipse with major_radius as positive length. (see 6.3.1, 6.2.1)

ae29 Ellipse with minor_radius as positive length. (see 6.3.1, 6.2.1)

4.1.19 Face

ae107 Face as face_surface. (see 6.2.1, 6.1.1)

ae30 Face as advanced_face. (see 6.3.1)

ae31 Face with one outer bound. (see 6.3.1, 6.2.1, 6.1.1)

ae32 Face with set of two or more bounds. (see 6.3.1, 6.2.1, 6.1.3)

ae57 Face used to define more than one closed_shell in different B-reps. (see 6.3.6)

ae62 Face connected to another face by shared edge. (see 6.3.1, 6.2.1)

ae63 Face with one bounding loop. (see 6.3.1, 6.2.1, 6.1.1)

ae64 Face with many bounding loop objects. (see 6.3.1, 6.2.1, 6.1.3)

ae65 Face with face_geometry as surface. (see 6.3.1, 6.2.1)

ae205 Face in faceted_brep with face_geometry as plane. (see 6.1.1)

ae548 Face presented as zero surface_appearance objects. (see 6.6.5)

ae608 Face presented as one surface_appearance.

4.1.20 Faceted_B-rep

ae201 Faceted_brep as manifold_solid_brep (see 6.1.1)

ae202 Faceted_brep with one void shell. (see 6.1.2)

ae203 Faceted_brep with two or more void shells. (see 6.1.3)

4.1.21 Geometric_element

ae33 Geometric_element as part of geometric description of B-rep. (see 6.3.1, 6.2.1, 6.1.1)

ae304 Geometric_element with user defined name. (see 6.4.1)

ae305 Geometric_element with zero name objects. (see 6.4.1)

ae609 Geometric_element as Point. (see 6.1.1, 6.2.1, 6.3.1)

ae610 Geometric_element as Curve. (see 6.1.1, 6.2.1, 6.3.1)

ae611 Geometric_element as Surface. (see 6.1.1, 6.2.1, 6.3.1)

4.1.22 Global_unit

ae404 Geometric context with global_units assigned. (see 6.5.1)

4.1.23 Hyperbola

ae75 Hyperbola with location. (see 6.3.6, 6.2.5)

4.1.24 Light_source

ae612 Light_source with colour. (see 6.6.1 to 6.6.5)

ae613 Light_source with position. (see 6.6.1)

ae614 Light_source with direction. (see 6.6.2)

ae615 Light_source without position. (see 6.6.1 to 6.6.5)

ae616 Light_source without direction. (see 6.6.1 to 6.6.5)

4.1.25 Line

ae19 Line. (see 6.3.3, 6.2.3)

4.1.26 Location

ae34 Location as axis2_placement_3d. (see 6.3.1, 6.2.1, 6.1.1)

ae83 Location as axis2_placement_2d. (see 6.3.13)

NOTE 1 An axis2_pplacement_2d is used only to define the geometry of a pcurve in the parameter space of a surface.

4.1.27 Loop

ae35 Loop as edge_loop. (see 6.3.1, 6.2.1)

ae36 Loop as vertex_loop. (see 6.3.2, 6.2.2)

ae204 Loop as poly_loop. (see 6.1.1)

ae67 edge_loop defined by one edge. (see 6.3.1, 6.2.1)

ae68 edge_loop defined by many edge objects. (see 6.3.6, 6.2.5)

4.1.28 Name

ae301 B-rep model with name. (see 6.4.1)

ae302 B-rep model component with name. (see 6.4.1)

ae303 B-rep model component without name. (see 6.4.1)

4.1.29 Parabola

ae15 Parabola. (see 6.3.6, 6.2.5)

ae37 Parabola with location. (see 6.3.6, 6.2.5)

4.1.30 Part

ae405 Part as B-rep solid model. (see 6.5.1)

ae406 Part with user_defined_name. (see 6.5.1)

ae617 Part in one Assembly.

ae420 Part contained in more than one assemblies. (see 6.5.4)

ae424 Part in zero Product objects.

ae618 Part in one Product.

ae425 Part in many Product objects. (see 6.5.4)

ae426 Part with geometry defined by one B-rep model. (see 6.5.1)

ae427 Part with geometry defined by many B-rep model objects. (see 6.5.1, 6.5.2)

4.1.31 Pcurve

ae76 Curve as Pcurve. (see 6.3.13)

ae77 Pcurve with basis_surface. (see 6.3.13)

ae78 Pcurve with curve_2d. (see 6.3.13)

4.1.32 Plane

ae102 Plane. (see 6.2.1, 6.3.1, 6.1.1)

ae108 Plane with location. (see 6.3.1, 6.2.1, 6.1.1)

4.1.33 Point

ae38 Point as cartesian_point. (see 6.3.1, 6.2.1, 6.1.1)

ae72 Point not used to define geometry of vertex. (see 6.3.1, 6.2.1)

ae535 Point presented as zero Point_appearance objects. (see 6.6.5)

ae619 Point presented as one Point_appearance.

4.1.34 Point_appearance

ae506 Point_appearance as point_style. (see 6.6.2)

ae508 Point_appearance with colour. (see 6.6.2)

ae620 Point_appearance with default colour.

ae509 Point_appearance with marker. (see 6.6.2)

ae621 Point_appearance with default marker.

ae510 Point_appearance with marker_size. (see 6.6.2)

ae511 Point_appearance using default marker_size. (see 6.6.1)

ae532 Point_appearance associated with zero point objects. (see 6.6.5)

ae533 Point_appearance associated with one point.

ae534 Point_appearance associated with many point objects. (see 6.6.1 to 6.6.7)

4.1.35 Polyline

ae5 Polyline. (see 6.3.4, 6.2.6)

4.1.36 Poly_loop

ae204 Poly_loop. (see 6.1.1)

4.1.37 Presentation_appearance

ae536 Presentation_appearance associated with zero B-rep objects. (see 6.6.5)

ae537 Presentation_appearance associated with one B-rep. (see 6.6.1 to 6.6.7)

ae538 Presentation_appearance associated with many B-rep objects.

ae540 Presentation_appearance associated with zero shell objects. (see 6.6.5)

ae541 Presentation_appearance associated with one shell.

ae542 Presentation_appearance associated with many shell objects. (see 6.6.1 to 6.6.7)

4.1.38 Product

ae407 Product as part. (see 6.5.1)

ae408 Product as assembly. (see 6.5.4)

ae409 Product with coordinate_system. (see 6.5.1)

ae410 Product with user_defined_name. (see 6.5.1)

ae411 Product with version_and_id. (see 6.5.1)

ae417 Product containing zero Assembly objects. (see 6.5.1)

ae622 Product containing many Assembly objects.

ae421 Product with one Part. (see 6.5.1)

ae422 Product with zero Part objects.

ae423 Product with many Part objects. (see 6.5.4)

4.1.39 Sculptured_surface

ae39 Sculptured_surface as B_spline_surface. (see 6.3.10)

4.1.40 Screen_image

ae507 Screen_image as presentation_area.

ae516 Screen_image containing one 3D_projection.

ae517 Screen_image containing two or more 3D_projections.

4.1.41 Shape_representation

ae40 Shape_representation. (see 6.3.1, 6.2.1, 6.1.1)

4.1.42 Shell

ae9 Shell as closed_shell. (see 6.3.1, 6.2.1)

ae543 Shell presented as zero presentation_appearance objects. (see 6.6.5)

ae623 Shell presented as one presentation_appearance.

ae544 Shell presented as many presentation_appearance objects. (see 6.6.1 to 6.6.7)

4.1.43 Spherical_surface

ae41 Spherical_surface with centre point. (see 6.3.1, 6.2.1)

ae42 Spherical_surface with radius as positive length. (see 6.3.1, 6.2.1)

4.1.44 Surface

ae109 Surface as Elementary_surface. (see 6.3.1, 6.2.1)

ae43 Surface as Swept_surface. (see 6.3.8)

ae44 Surface as Sculptured_surface. (see 6.3.10)

ae66 Surface used to define the geometry of more than one face. (see 6.3.15)

ae552 Surface presented as zero surface_appearance objects. (see 6.6.5)

ae624 Surface presented as one surface_appearance.

4.1.45 Surface_appearance

ae512 Surface_appearance as surface_style_usage. (see 6.6.1 to 6.6.7)

ae513 Surface_appearance with colour. (see 6.6.1 to 6.6.7)

ae514 Surface_appearance with grid_indicator. (see 6.6.2)

ae515 Surface_appearance with shading_method. (see 6.6.2)

ae625 Surface_appearance with default colour.

ae626 Surface_appearance with unspecified grid_indicator.

ae627 Surface_appearance with unspecified shading_method.

ae545 Surface_appearance associated with zero face objects. (see 6.6.5)

ae546 Surface_appearance associated with one face.

ae547 Surface_appearance associated with many face objects. (see 6.6.1 to 6.6.7)

ae549 Surface_appearance associated with zero surface objects. (see 6.6.5)

ae550 Surface_appearance associated with one surface.

ae551 Surface_appearance associated with many surface objects. (see 6.6.1 to 6.6.7)

4.1.46 Surface_curve

ae79 Curve as Surface_curve. (see 6.3.13)

ae80 Surface_curve with curve_3d. (see 6.3.13)

ae81 Surface_curve with associated geometry as a set of one pcurve. (see 6.3.13)

ae82 Surface_curve with associated geometry as a set of two pcurves. (see 6.3.13)

4.1.47 Surface_of_extrusion

ae45 Surface_of_extrusion. (see 6.3.8)

ae46 Surface_of_extrusion with extruded curve. (see 6.3.8)

ae47 Surface_of_extrusion with extrusion direction. (see 6.3.8)

4.1.48 Surface_of_revolution

ae48 Surface_of_revolution. (see 6.3.8)

ae49 Surface_of_revolution with revolved curve. (see 6.3.8)

ae50 Surface_of_revolution with axis. (see 6.3.8)

4.1.49 Swept_surface

ae45 Swept_surface as Surface_of_extrusion. (see 6.3.8)

ae48 Swept_surface as Surface_of_revolution. (see 6.3.8)

4.1.50 Topological_element

ae628 as Vertex. (see 6.2.1, 6.3.1)

ae629 as Edge. (see 6.2.1, 6.3.1)

ae630 as Loop. (see 6.2.1, 6.3.1)

ae631 as Face. (see 6.2.1, 6.3.1)

ae632 as Shell. (see 6.2.1, 6.3.1)

ae306 Topological_element with user defined name. (see 6.4.1)

ae307 Topological_element with zero name objects. (see 6.4.1)

ae553 Topological_representation_item associated with zero presentation attribute objects. (see 6.6.5)

ae554 Topological_representation_item associated with one presentation attribute.

ae555 Topological_representation_item associated with many presentation attribute objects. (see 6.6.1 to 6.6.7)

4.1.51 Toroidal_surface

ae107 Toroidal_surface. (see 6.2.3, 6.3.3)

ae110 Toroidal_surface with centre point. (see 6.2.3, 6.3.3)

ae111 Toroidal_surface with axis. (see 6.2.3, 6.3.3)

ae112 Toroidal_surface with two radii. (see 6.2.3, 6.3.3)

ae633 as Degenerate_Toroidal_Surface. (see 6.3.12)

4.1.52 Transformation

ae49 Transformation as cartesian_transformation_operator_3d. (see 6.3.14, 6.1.5)

ae50 Transformation with scaling factor. (see 6.3.14, 6.1.5)

4.1.53 Twisted_curve

ae51 Twisted_curve as B_spline_curve. (see 6.3.5)

4.1.54 Unbounded_curve

ae52 Unbounded_curve used as edge_geometry. (see 6.3.6, 6.2.5)

4.1.55 Vertex

ae53 Vertex as vertex_point. (see 6.3.1, 6.2.1)

ae71 Vertex with geometry defined by point. (see 6.3.1, 6.2.1)

4.1.56 Void

ae54 Void as closed_shell. (see 6.3.2, 6.2.2, 6.1.2)

ae55 Void with topological_normal pointing into void. (see 6.3.2, 6.2.2, 6.1.2)

4.2 AIM test purposes

AIM test purposes are identified by the prefix “aim” in the test purpose number. Each test purpose derived from the AIM EXPRESS shall be interpreted as:

Correctly instantiate in the implementation under test the AIM elements associated with the unique AIM entity corresponding to (*insert test purpose here*) in at least one test case within the test suite.

Correctly instantiating an AIM element implies that the semantics of the application element represented by the AIM element are preserved between the input and the output of a test as well as conformance to the reference path specified in the mapping table of the AP. AIM test purposes apply to the input specifications of postprocessor test cases only. AIM test purposes are derived directly from the expanded EXPRESS listing contained in annex A of ISO 10303-204 as follows:

— AIM entities. A test purposes derived from an AIM element is a simple statement of the entity name.

— AIM entity attributes. Test purposes derived from AIM entity attributes are statements of the AIM entity with a given attribute.

Each AIM entity test purpose is grouped with its attribute test purposes.

In the following clauses each test purpose is stated as an entity name, which may require instantiation as a particular subtype, with one or more defined attributes. Each test purpose statement implicitly requires that the entity with the attribute(s) given in the statement will be correctly instantiated in the

implementation under test in at least one test case. A cross reference to the first abstract test case which exercises the test purpose is provided.

Test purposes are grouped into the following categories:

— faceted B-rep (1-6);

— geometry common to elementary and advanced B-reps (7–105);

— elementary B-rep (106);

— advanced B-rep (107–172);

— name preservation (173–198);

— product structure (199–268);

— measures and units (269-295);

— visual presentation (296-441).

4.2.1 faceted_brep

aim1: faceted_brep (no specific test purposes) (see 6.1.1, 6.1.2)

4.2.2 faceted_brep_shape_representation

aim2: faceted_brep_shape_representation (no specific test purposes) (see 6.1.1, 6.1.4)

4.2.3 geometric_representation_item

aim3: geometric_representation_item as **poly_loop** (see 6.1.1)

4.2.4 loop

aim4: loop as **poly_loop** (see 6.1.3)

4.2.5 manifold_solid_brep

aim5: manifold_solid_brep as **faceted_brep** (see 6.1.1, 6.1.3)

aim6: manifold_solid_brep as **faceted_brep AND brep_with_voids** (see 6.1.2)

4.2.6 poly_loop

aim7: poly_loop with **polygon** having at least one element as cartesian_point (see 6.1.3)

4.2.7 axis2_placement_3d

aim8: axis2_placement_3d with **ref_direction** present as direction (see 6.1.1, 6.2.1, 6.3.1)

aim9: axis2_placement_3d with **ref_direction** absent (see 6.1.3, 6.2.4, 6.3.4)

aim10: axis2_placement_3d with **axis** present as direction (see 6.1.1, 6.2.1, 6.3.1)

aim11: axis2_placement_3d with **axis** absent (see 6.1.3, 6.2.4, 6.3.4)

4.2.8 bounded_curve

aim12: bounded_curve as **polyline** (see 6.2.4, 6.3.4)

4.2.9 brep_with_voids

aim13: brep_with_voids with **voids** having at least one element as oriented_closed_shell (see 6.1.2, 6.2.2, 6.3.2)

4.2.10 cartesian_point

aim14: cartesian_point with **coordinates** having at least one element as length_measure (see 6.1.1, 6.2.1, 6.3.1)

4.2.11 cartesian_transformation_operator

aim15: cartesian_transformation_operator with **local_origin** as cartesian_point (see 6.1.5, 6.3.14, 6.5.4)

aim16: cartesian_transformation_operator with **axis2** present as direction (see 6.1.5, 6.3.14)

aim17: cartesian_transformation_operator with **axis2** absent

aim18: cartesian_transformation_operator with **axis1** present as direction (see 6.1.5, 6.3.14)

aim19: cartesian_transformation_operator with **axis1** absent

aim20: cartesian_transformation_operator as **cartesian_transformation_operator_3d** (see 6.1.5, 6.3.14, 6.5.4)

aim21: cartesian_transformation_operator with **scale** present as real (see 6.1.5, 6.3.14, 6.5.4)

aim22: cartesian_transformation_operator with **scale** absent

4.2.12 cartesian_transformation_operator_3d

aim23: cartesian_transformation_operator_3d with **axis3** present as direction (see 6.1.5, 6.3.14)

aim24: cartesian_transformation_operator_3d with **axis3** absent

4.2.13 circle

aim25: circle with **radius** as positive_length_measure (see 6.2.1, 6.3.1)

4.2.14 closed_shell

aim26: closed_shell as **closed_shell** (see 6.1.1, 6.2.1, 6.3.1)

aim27: closed_shell as **oriented_closed_shell** (see 6.1.2, 6.2.2, 6.3.2)

4.2.15 conic

aim28: conic with **position** as axis2_placement_3d (see 6.2.5, 6.3.6)

aim29: conic with **position** as axis2_placement_2d (see 6.3.13)

aim30: conic as **parabola** (see 6.2.5, 6.3.6)

aim31: conic as **hyperbola** (see 6.2.5, 6.3.6)

aim32: conic as **ellipse** (see 6.2.1, 6.3.6)

aim33: conic as **circle** (see 6.2.1, 6.3.1)

4.2.16 conical_surface

aim34: conical_surface with **semi_angle** as plane_angle_measure (see 6.2.5, 6.3.6)

aim35: conical_surface with **radius** as length_measure (see 6.2.5, 6.3.6)

4.2.17 connected_face_set

aim36: connected_face_set with **cfs_faces** having at least one element as face (see 6.1.1, 6.2.2, 6.3.1, 6.3.2)

aim37: connected_face_set as **closed_shell** (see 6.3.1, 6.3.2)

4.2.18 curve

aim38: curve as **bounded_curve** (see 6.2.4, 6.3.4, 6.3.5)

aim39: curve as **conic** (see 6.2.1, 6.3.1)

aim40: curve as **line** (see 6.2.3, 6.3.3)

4.2.19 cylindrical_surface

aim41: cylindrical_surface with **radius** as positive_length_measure (see 6.2.1, 6.3.1)

4.2.20 degenerate_toroidal_surface

aim42: degenerate_toroidal_surface with **select_outer** = true (see 6.3.12)

aim43: degenerate_toroidal_surface with **select_outer** = false (see 6.3.12)

4.2.21 direction

aim44: direction (no specific test purposes) (see 6.2.1, 6.3.1)

4.2.22 edge

aim45: edge with **edge_end** as vertex (see 6.2.1, 6.3.1)

aim46: edge with **edge_start** as vertex (see 6.2.1, 6.3.1)

aim47: edge as **oriented_edge** (see 6.2.1, 6.2.3, 6.3.1, 6.3.3)

aim48: edge as **edge_curve** (see 6.2.1, 6.3.1)

4.2.23 edge_curve

aim49: edge_curve with **same_sense** = true (see 6.2.1, 6.3.1)

aim50: edge_curve with **same_sense** = false (see 6.2.5, 6.3.6)

aim51: edge_curve with **edge_geometry** as curve (see 6.2.1, 6.2.3, 6.2.4, 6.3.3, 6.3.4, 6.3.1, 6.3.5, 6.3.13)

4.2.24 edge_loop

aim52: edge_loop (no specific test purposes) (see 6.3.1)

4.2.25 elementary_surface

aim53: elementary_surface with **position** as axis2_placement_3d (see 6.1.1, 6.1.3, 6.2.1, 6.2.4, 6.3.1, 6.3.4)

aim54: elementary_surface as **toroidal_surface** (see 6.2.3, 6.3.3, 6.3.12)

aim55: elementary_surface as **spherical_surface** (see 6.2.1, 6.3.1)

aim56: elementary_surface as **conical_surface** (see 6.2.5, 6.3.6)

aim57: elementary_surface as **cylindrical_surface** (see 6.2.1, 6.3.1)

aim58: elementary_surface as **plane** (see 6.1.1, 6.1.3, 6.2.1, 6.3.1)

4.2.26 ellipse

aim59: ellipse with **semi_axis_2** as positive_length_measure (see 6.2.1, 6.3.1)

aim60: ellipse with **semi_axis_1** as positive_length_measure (see 6.2.1, 6.3.1)

4.2.27 face

aim61: face with **bounds** having at least one element as face_bound (see 6.1.1, 6.1.3, 6.2.1, 6.2.5, 6.3.1, 6.3.6, 6.3.11)

aim62: face as **face_surface** (see 6.1.1, 6.1.3, 6.2.1, 6.2.5, 6.3.1, 6.3.6, 6.3.11)

4.2.28 face_bound

aim63: face_bound with **orientation** = true (see 6.1.1, 6.1.3, 6.2.1, 6.3.1)

aim64: face_bound with **orientation** = false (see 6.1.3, 6.2.1, 6.3.1)

aim65: face_bound with **bound** as loop (see 6.1.1 6.1.3, 6.2.1, 6.2.5, 6.3.1)

aim66: face_bound as **face_outer_bound** (see 6.1.1, 6.1.3, 6.2.1, 6.2.5, 6.3.1)

4.2.29 face_outer_bound

aim67: face_outer_bound (no specific test purposes) (see 6.2.1, 6.3.1)

4.2.30 face_surface

aim68: face_surface with **same_sense** = true (see 6.1.1, 6.2.1)

aim69: face_surface with **same_sense** = false (see 6.1.3, 6.2.5)

aim70: face_surface with **face_geometry** as surface (see 6.1.1, 6.2.1)

4.2.31 geometric_representation_context

aim71: geometric_representation_context with **coordinate_space_dimension** as dimension_count (see 6.1.1, 6.2.1, 6.3.1, 6.5.1)

4.2.32 hyperbola

aim72: hyperbola with **semi_imag_axis** as positive_length_measure (see 6.2.5, 6.3.6)

aim73: hyperbola with **semi_axis** as positive_length_measure (see 6.2.5, 6.3.6)

4.2.33 line

aim74: line with **dir** as vector (see 6.2.5, 6.3.6)

aim75: line with **pnt** as cartesian_point (see 6.2.5, 6.3.6)

4.2.34 loop

aim76: loop as **edge_loop** (see 6.2.1, 6.3.1)

aim77: loop as **vertex_loop** (see 6.2.2, 6.2.5, 6.3.2)

4.2.35 manifold_solid_brep

aim78: manifold_solid_brep with **outer** as closed_shell (see 6.1.1, 6.1.2, 6.2.1, 6.3.1, 6.3.2)

aim79: manifold_solid_brep as **brep_with_voids** (see 6.2.2, 6.3.2)

4.2.36 oriented_closed_shell

aim80: oriented_closed_shell with **orientation** = true

aim81: oriented_closed_shell with **orientation** = false (see 6.1.2 6.2.2, 6.3.2)

aim82: oriented_closed_shell with **closed_shell_element** as closed_shell (see 6.1.2, 6.2.2, 6.3.2)

4.2.37 oriented_edge

aim83: oriented_edge with **orientation** = true (see 6.2.1, 6.3.1)

aim84: oriented_edge with **orientation** = false (see 6.2.3, 6.3.3)

aim85: oriented_edge with **edge_element** as edge (see 6.2.1, 6.3.1)

4.2.38 parabola

aim86: parabola with **focal_dist** as length_measure (see 6.2.5, 6.3.6)

4.2.39 parametric_representation_context

aim87: parametric_representation_context (no specific test purposes) (see 6.3.13)

4.2.40 path

aim88: path with **edge_list** having at least one element as oriented_edge (see 6.2.1, 6.3.1)

aim89: path as **edge_loop** (see 6.2.1, 6.3.1)

4.2.41 placement

aim90: placement with **location** as cartesian_point (see 6.1.1, 6.2.1)

aim91: placement as **axis2_placement_3d** (see 6.1.1, 6.1.3, 6.2.1, 6.3.1)

4.2.42 plane

aim92: plane (no specific test purposes) (see 6.1.1, 6.2.1, 6.3.1)

4.2.43 point

aim93: point as **cartesian_point** (see 6.2.2, 6.3.2)

4.2.44 polyline

aim94: polyline with **points** having at least one element as cartesian_point (see 6.2.4, 6.3.4)

4.2.45 solid_model

aim95: solid_model as **manifold_solid_brep** (see 6.1.1, 6.2.1, 6.3.1)

4.2.46 spherical_surface

aim96: spherical_surface with **radius** as positive_length_measure (see 6.2.1, 6.3.1)

4.2.47 surface

aim97: surface as **elementary_surface** (see 6.1.1, 6.1.3, 6.2.1, 6.3.1)

4.2.48 toroidal_surface

aim98: toroidal_surface with **minor_radius** as positive_length_measure (see 6.2.3, 6.3.3)

aim99: toroidal_surface with **major_radius** as positive_length_measure (see 6.2.3, 6.3.3)

aim100: toroidal_surface as **degenerate_toroidal_surface** (see 6.3.12)

4.2.49 vector

aim101: vector with **magnitude** as length_measure (see 6.2.5, 6.3.6)

aim102: vector with **orientation** as direction (see 6.2.5, 6.3.6)

4.2.50 vertex

aim103: vertex as **vertex_point** (see 6.2.1, 6.3.2, 6.3.1)

4.2.51 vertex_loop

aim104: vertex_loop with **loop_vertex** as vertex (see 6.2.2, 6.3.2)

4.2.52 vertex_point

aim105: vertex_point with **vertex_geometry** as point (see 6.2.2, 6.3.2)

4.2.53 elementary_brep_shape_representation

aim106: elementary_brep_shape_representation (no specific test purposes) (see 6.2.1)

4.2.54 advanced_brep_shape_representation

aim107: advanced_brep_shape_representation (no specific test purposes) (see 6.3.1)

4.2.55 advanced_face

aim108: advanced_face (no specific test purposes) (see 6.3.1)

4.2.56 axis1_placement

aim109: axis1_placement with **axis** present as direction (see 6.3.8)

aim110: axis1_placement with **axis** absent

4.2.57 axis2_placement_2d

aim111: axis2_placement_2d with **ref_direction** present as direction (see 6.3.13)

aim112: axis2_placement_2d with **ref_direction** absent

4.2.58 b_spline_curve

aim113: b_spline_curve with **self_intersect** = false (see 6.3.5)

aim114: b_spline_curve with **self_intersect** = unknown (see 6.3.9)

aim115: b_spline_curve with **closed_curve** = true (see 6.3.5)

aim116: b_spline_curve with **closed_curve** = false (see 6.3.10)

aim117: b_spline_curve with **closed_curve** = unknown (see 6.3.9)

aim118: b_spline_curve with **curve_form** = unspecified (see 6.3.5)

aim119: b_spline_curve with **control_points_list** having at least one element as cartesian_point (see 6.3.5)

aim120: b_spline_curve as **rational_b_spline_curve** (see 6.3.10)

aim121: b_spline_curve as **bezier_curve** (see 6.3.5, 6.3.10)

aim122: b_spline_curve as **b_spline_curve_with_knots** (see 6.3.11)

4.2.59 b_spline_curve_with_knots

aim123: b_spline_curve_with_knots with **knot_spec** = piecewise_bezier_knots (see 6.3.11)

aim124: b_spline_curve_with_knots with **knot_spec** = quasi_uniform_knots

aim125: b_spline_curve_with_knots with **knot_spec** = unspecified

aim126: b_spline_curve_with_knots with **knot_spec** = uniform_knots

aim127: b_spline_curve_with_knots with **knots** having at least one element as parameter_value (see 6.3.11)

4.2.60 b_spline_surface

aim128: b_spline_surface with **self_intersect** = false (see 6.3.10)

aim129: b_spline_surface with **self_intersect** = unknown (see 6.3.11)

aim130: b_spline_surface with **v_closed** = true (see 6.3.11)

aim131: b_spline_surface with **v_closed** = false (see 6.3.10)

aim132: b_spline_surface with **v_closed** = unknown

aim133: b_spline_surface with **u_closed** = true (see 6.3.11)

aim134: b_spline_surface with **u_closed** = false (see 6.3.10)

aim135: b_spline_surface with **u_closed** = unknown

aim136: b_spline_surface with **surface_form** = unspecified (see 6.3.10)

aim137: b_spline_surface with **control_points_list** having at least one element as cartesian_point

aim138: b_spline_surface as **rational_b_spline_surface** (see 6.3.10)

aim139: b_spline_surface as **bezier_surface** (see 6.3.10)

aim140: b_spline_surface as **b_spline_surface_with_knots** (see 6.3.11)

4.2.61 b_spline_surface_with_knots

aim141: b_spline_surface_with_knots with **knot_spec** = piecewise_bezier_knots

aim142: b_spline_surface_with_knots with **knot_spec** = quasi_uniform_knots

aim143: b_spline_surface_with_knots with **knot_spec** = unspecified (see 6.3.11)

aim144: b_spline_surface_with_knots with **knot_spec** = uniform_knots

aim145: b_spline_surface_with_knots with **v_knots** having at least one element as parameter_value

aim146: b_spline_surface_with_knots with **u_knots** having at least one element as parameter_value (see 6.3.11)

4.2.62 bezier_curve

aim147: bezier_curve (no specific test purposes) (see 6.3.9)

4.2.63 bezier_surface

aim148: bezier_surface (no specific test purposes) (see 6.3.9)

4.2.64 bounded_curve

aim149: bounded_curve as **b_spline_curve** (see 6.3.5)

4.2.65 bounded_surface

aim150: bounded_surface as **b_spline_surface** (see 6.3.10)

4.2.66 curve

aim151: curve as **surface_curve** (see 6.3.13)

aim152: curve as **pcurve** (see 6.3.13)

4.2.67 definitional_representation

aim153: definitional_representation (no specific test purposes) (see 6.3.13)

4.2.68 face_surface

aim154: face_surface as **advanced_face** (see 6.3.1, 6.3.6, 6.3.11)

4.2.69 pcurve

aim155: pcurve with **reference_to_curve** as definitional_representation (see 6.3.13)

aim156: pcurve with **basis_surface** as surface (see 6.3.13)

4.2.70 placement

aim157: placement as **axis2_placement_2d** (see 6.3.13)

aim158: placement as **axis1_placement** (see 6.3.8)

4.2.71 rational_b_spline_curve

aim159: rational_b_spline_curve (no specific test purposes) (see 6.3.10)

4.2.72 rational_b_spline_surface

aim160: rational_b_spline_surface (no specific test purposes) (see 6.3.10)

4.2.73 surface

aim161: surface as **bounded_surface** (see 6.3.10)

aim162: surface as **swept_surface** (see 6.3.8)

4.2.74 surface_curve

aim163: surface_curve with **master_representation** = pcurve_s2

aim164: surface_curve with **master_representation** = pcurve_s1 (see 6.3.13)

aim165: surface_curve with **master_representation** = curve_3d (see 6.3.13)

aim166: surface_curve with **associated_geometry** having at least one element as pcurve (see 6.3.13)

aim167: surface_curve with **curve_3d** as curve (see 6.3.13)

4.2.75 surface_of_linear_extrusion

aim168: surface_of_linear_extrusion with **extrusion_axis** as vector (see 6.3.8)

4.2.76 surface_of_revolution

aim169: surface_of_revolution with **axis_position** as axis1_placement (see 6.3.8)

4.2.77 swept_surface

aim170: swept_surface with **swept_curve** as curve (see 6.3.8)

aim171: swept_surface as **surface_of_revolution** (see 6.3.8)

aim172: swept_surface as **surface_of_linear_extrusion** (see 6.3.8)

4.2.78 geometric_representation_item

aim173: geometric_representation_item as **planar_extent**

aim174: geometric_representation_item as **camera_model** (see 6.6.1)

aim175: geometric_representation_item as **light_source** (see 6.6.1)

aim176: geometric_representation_item as **solid_model** (see 6.1.1, 6.2.1, 6.3.1)

aim177: geometric_representation_item as **vertex_point** (see 6.2.1, 6.3.1)

aim178: geometric_representation_item as **face_surface** (see 6.1.1, 6.2.1, 6.3.1)

aim179: geometric_representation_item as **edge_curve** (see 6.2.1, 6.3.1)

aim180: geometric_representation_item as **surface** (see 6.2.1, 6.3.1)

aim181: geometric_representation_item as **curve** (see 6.2.1, 6.3.1)

aim182: geometric_representation_item as **cartesian_transformation_operator** (see 6.3.14, 6.5.4)

aim183: geometric_representation_item as **placement** (see 6.2.1, 6.3.1)

aim184: geometric_representation_item as **vector** (see 6.2.5, 6.3.6)

aim185: geometric_representation_item as **direction** (see 6.1.1)

aim186: geometric_representation_item as **point** (see 6.2.1)

4.2.79 representation_item

aim187: representation_item with **name** as label (see 6.4.1)

aim188: representation_item as **topological_representation_item** (see 6.4.1)

aim189: representation_item as **geometric_representation_item** (see 6.3.14, 6.4.1, 6.5.4)

aim190: representation_item as **styled_item**

aim191: representation_item as **mapped_item** (see 6.2.6, 6.3.14)

4.2.80 topological_representation_item

aim192: topological_representation_item as **path** (see 6.2.1, 6.3.1)

aim193: topological_representation_item as **loop** (see 6.1.1, 6.2.1, 6.3.1)

aim194: topological_representation_item as **connected_face_set** (see 6.1.1, 6.2.1, 6.3.1)

aim195: topological_representation_item as **face** (see 6.1.1, 6.2.1, 6.3.1)

aim196: topological_representation_item as **face_bound** (see 6.1.1, 6.2.1, 6.3.1)

aim197: topological_representation_item as **edge** (see 6.2.1, 6.3.1)

aim198: topological_representation_item as **vertex** (see 6.2.1, 6.3.1)

4.2.81 application_context

aim199: application_context with **application** as text (see 6.5.1)

4.2.82 application_context_element

aim200: application_context_element with **frame_of_reference** as application_context (see 6.5.1)

aim201: application_context_element with **name** as label (see 6.5.1)

aim202: application_context_element as **product_definition_context** (see 6.5.1)

aim203: application_context_element as **product_context** (see 6.5.1)

4.2.83 application_protocol_definition

aim204: application_protocol_definition with **application** as application_context (see 6.5.1)

aim205: application_protocol_definition with **application_protocol_year** as year_number (see 6.5.1)

aim206: application_protocol_definition with **application_interpreted_model_schema_name** as label (see 6.5.1) [2)]

aim207: application_protocol_definition with **status** as label (see 6.5.1)

4.2.84 assembly_component_usage

aim208: assembly_component_usage with **reference_designator** present as identifier (see 6.5.4)

[2)] = 'part_204_brep_product_schema'

aim209: assembly_component_usage with **reference_designator** absent (see 6.5.4)

4.2.85 design_context

aim210: design_context (no specific test purposes) (see 6.5.1)

4.2.86 functionally_defined_transformation

aim211: functionally_defined_transformation with **description** as text (see 6.1.5, 6.3.14)

aim212: functionally_defined_transformation with **name** as label (see 6.1.5, 6.3.14)

aim213: functionally_defined_transformation as **cartesian_transformation_operator** (see 6.1.5, 6.3.14)

4.2.87 mapped_item

aim214: mapped_item with **mapping_target** as representation_item (see 6.2.6, 6.3.14, 6.5.4)

aim215: mapped_item with **mapping_source** as representation_map (see 6.2.6, 6.3.14)

aim216: mapped_item as **camera_image**

4.2.88 mechanical_context

aim217: mechanical_context (no specific test purposes) (see 6.5.1)

4.2.89 product

aim218: product with **frame_of_reference** having at least one element as product_context (see 6.5.1)

aim219: product with **description** as text (see 6.5.4)

aim220: product with **name** as label (see 6.5.4)

aim221: product with **id** as identifier (see 6.5.4)

4.2.90 product_context

aim222: product_context with **discipline_type** as label (see 6.5.1)

aim223: product_context as **mechanical_context** (see 6.5.1)

4.2.91 product_definition

aim224: product_definition with **frame_of_reference** as product_definition_context (see 6.5.1)

aim225: product_definition with **formation** as product_definition_formation (see 6.5.1)

aim226: product_definition with **description** as text (see 6.5.1)

aim227: product_definition with **id** as identifier (see 6.5.1)

4.2.92 product_definition_context

aim228: product_definition_context with **life_cycle_stage** as label (see 6.5.1)

aim229: product_definition_context as **design_context** (see 6.5.1)

4.2.93 product_definition_formation

aim230: product_definition_formation with **of_product** as product (see 6.5.1)

aim231: product_definition_formation with **description** as text (see 6.5.1)

aim232: product_definition_formation with **id** as identifier (see 6.5.1)

4.2.94 product_definition_relationship

aim233: product_definition_relationship with **related_product_definition** as product_definition (see 6.5.4)

aim234: product_definition_relationship with **relating_product_definition** as product_definition (see 6.5.4)

aim235: product_definition_relationship with **description** as text (see 6.5.4)

aim236: product_definition_relationship with **name** as label (see 6.5.4)

aim237: product_definition_relationship with **id** as identifier (see 6.5.4)

aim238: product_definition_relationship as **product_definition_usage** (see 6.5.4)

4.2.95 product_definition_shape

aim239: product_definition_shape (no specific test purposes) (see 6.5.2)

4.2.96 product_definition_usage

aim240: product_definition_usage as **assembly_component_usage** (see 6.5.4)

4.2.97 property_definition

aim241: property_definition with **definition** as product_definition (see 6.5.1, 6.5.2, 6.5.3)

aim242: property_definition with **definition** as product_definition_relationship (see 6.5.1, 6.5.2, 6.5.3)

aim243: property_definition with **description** as text (see 6.5.1, 6.5.2, 6.5.3)

aim244: property_definition with **name** as label (see 6.5.1, 6.5.2, 6.5.3)

aim245: property_definition as **product_definition_shape** (see 6.5.1, 6.5.2, 6.5.3)

4.2.98 property_definition_representation

aim246: property_definition_representation with **used_representation** as representation (see 6.5.1, 6.5.2, 6.5.3)

aim247: property_definition_representation with **definition** as property_definition (see 6.5.1, 6.5.2, 6.5.3)

aim248: property_definition_representation as **shape_definition_representation** (see 6.5.1, 6.5.2, 6.5.3)

4.2.99 representation

aim249: representation with **context_of_items** as representation_context (see 6.5.1, 6.5.2, 6.5.3)

aim250: representation with **items** having at least one element as representation_item (see 6.5.1, 6.5.2, 6.5.3)

aim251: representation with **name** as label (see 6.5.1, 6.5.2, 6.5.3)

aim252: representation as **shape_representation** (see 6.5.1, 6.5.2, 6.5.3)

aim253: representation as **presentation_representation**

aim254: representation as **mechanical_design_shaded_presentation_representation**

aim255: representation as **mechanical_design_geometric_presentation_representation**

aim256: representation as **definitional_representation** (see 6.3.13)

4.2.100 representation_context

aim257: representation_context with **context_type** as text (see 6.5.1)

aim258: representation_context with **context_identifier** as identifier (see 6.5.1)

aim259: representation_context as **parametric_representation_context** (see 6.3.13)

aim260: representation_context as **global_unit_assigned_context** (see 6.1.1, 6.2.1, 6.3.1)

aim261: representation_context as **geometric_representation_context** (see 6.1.1, 6.2.1, 6.3.1)

4.2.101 representation_map

aim262: representation_map with **mapped_representation** as representation (see 6.1.4, 6.2.6, 6.3.14)

aim263: representation_map with **mapping_origin** as representation_item (see 6.1.4, 6.2.6, 6.3.14)

aim264: representation_map as **camera_usage**

4.2.102 shape_definition_representation

aim265: shape_definition_representation (no specific test purposes) (see 6.5.1, 6.5.2, 6.5.3)

4.2.103 shape_representation

aim266: shape_representation as **faceted_brep_shape_representation** (see 6.1.1, 6.5.3)

aim267: shape_representation as **elementary_brep_shape_representation** (see 6.2.1, 6.5.2)

aim268: shape_representation as **advanced_brep_shape_representation** (see 6.3.1, 6.5.1)

4.2.104 conversion_based_unit

aim269: conversion_based_unit with **conversion_factor** as measure_with_unit (see 6.2.5)

aim270: conversion_based_unit with **name** as label (see 6.2.5)

4.2.105 derived_unit

aim271: derived_unit with **elements** having at least one element as derived_unit_element

4.2.106 derived_unit_element

aim272: derived_unit_element with **unit** as named_unit

4.2.107 dimensional_exponents

aim273: dimensional_exponents (no specific test purposes) (see 6.1.1, 6.2.1, 6.3.1)

4.2.108 global_unit_assigned_context

aim274: global_unit_assigned_context with **units** having at least one element as derived_unit

aim275: global_unit_assigned_context with **units** having at least one element as named_unit (see 6.1.1, 6.2.1, 6.3.1)

4.2.109 length_measure_with_unit

aim276: length_measure_with_unit (no specific test purposes)

4.2.110 length_unit

aim277: length_unit (no specific test purposes) (see 6.1.1, 6.2.1, 6.3.1)

4.2.111 measure_with_unit

aim278: measure_with_unit with **unit_component** as derived_unit

aim279: measure_with_unit with **unit_component** as named_unit (see 6.2.5)

aim280: measure_with_unit with **value_component** as positive_length_measure

aim281: measure_with_unit with **value_component** as parameter_value

aim282: measure_with_unit with **value_component** as plane_angle_measure (see 6.2.5)

aim283: measure_with_unit with **value_component** as length_measure

aim284: measure_with_unit as **plane_angle_measure_with_unit** (see 6.2.5)

aim285: measure_with_unit as **length_measure_with_unit**

4.2.112 named_unit

aim286: named_unit with **dimensions** as dimensional_exponents (see 6.1.1, 6.2.1, 6.3.1)

aim287: named_unit as **plane_angle_unit** (see 6.2.5)

aim288: named_unit as **length_unit** (see 6.1.1)

aim289: named_unit as **conversion_based_unit** (see 6.2.5)

aim290: named_unit as **si_unit** (see 6.1.1)

4.2.113 plane_angle_measure_with_unit

aim291: plane_angle_measure_with_unit (no specific test purposes) (see 6.2.5)

4.2.114 plane_angle_unit

aim292: plane_angle_unit (no specific test purposes) (see 6.2.5)

4.2.115 si_unit

aim293: si_unit with **name** = radian (see 6.1.1)

aim294: si_unit with **name** = metre (see 6.1.1)

aim295: si_unit with **prefix** present as milli (see 6.1.1)

4.2.116 background_colour

aim296: background_colour with **presentation** as presentation_view

aim297: background_colour with **presentation** as presentation_area

4.2.117 camera_image

aim298: camera_image as **camera_image_3d_with_scale**

4.2.118 camera_image_3d_with_scale

aim299: camera_image_3d_with_scale (no specific test purposes)

4.2.119 camera_model

aim300: camera_model as **camera_model_d3** (see 6.6.1)

4.2.120 camera_model_d3

aim301: camera_model_d3 with **perspective_of_volume** as view_volume (see 6.6.1)

aim302: camera_model_d3 with **view_reference_system** as axis2_placement_3d (see 6.6.1)

aim303: camera_model_d3 as **camera_model_with_light_sources** (see 6.6.1)

aim304: camera_model_d3 as **camera_model_d3_with_hlhsr** (see 6.6.1)

4.2.121 camera_model_d3_with_hlhsr

aim305: camera_model_d3_with_hlhsr with **hidden_line_surface_removal** = true (see 6.6.1)

aim306: camera_model_d3_with_hlhsr with **hidden_line_surface_removal** = false (see 6.6.5)

4.2.122 camera_model_with_light_sources

aim307: camera_model_with_light_sources with **sources** having at least one element as light_source (see 6.6.1)

4.2.123 camera_usage

aim308: camera_usage (no specific test purposes)

4.2.124 colour

aim309: colour with **name** as label (see 6.6.1)

aim310: colour as **pre_defined_colour**

aim311: colour as **colour_specification** (see 6.6.1)

4.2.125 colour_rgb

aim312: colour_rgb (no specific test purposes) (see 6.6.1)

4.2.126 colour_specification

aim313: colour_specification as **colour_rgb** (see 6.6.1)

4.2.127 curve_style

aim314: curve_style with **curve_colour** as colour (see 6.6.2)

aim315: curve_style with **curve_width** as measure_with_unit

aim316: curve_style with **curve_width** as positive_length_measure (see 6.6.2)

aim317: curve_style with **curve_font** as curve_style_font_and_scaling

aim318: curve_style with **curve_font** as curve_style_font (see 6.6.5)

aim319: curve_style with **curve_font** as pre_defined_curve_font (see 6.6.2)

aim320: curve_style with **name** as label (see 6.6.2)

4.2.128 curve_style_font

aim321: curve_style_font with **pattern_list** having at least one element as curve_style_font_pattern (see 6.6.5)

aim322: curve_style_font with **name** as label (see 6.6.5)

4.2.129 curve_style_font_and_scaling

aim323: curve_style_font_and_scaling with **curve_font** as pre_defined_curve_font (see 6.6.2)

aim324: curve_style_font_and_scaling with **curve_font** as curve_style_font (see 6.6.5)

aim325: curve_style_font_and_scaling with **name** as label (see 6.6.2)

4.2.130 curve_style_font_pattern

aim326: curve_style_font_pattern with **invisible_segment_length** as positive_length_measure (see 6.6.5)

aim327: curve_style_font_pattern with **visible_segment_length** as positive_length_measure (see 6.6.5)

4.2.131 curve_style_rendering

aim328: curve_style_rendering with **rendering_properties** as surface_rendering_properties (see 6.6.1)

aim329: curve_style_rendering with **rendering_method** = linear_colour (see 6.6.2)

aim330: curve_style_rendering with **rendering_method** = constant_colour (see 6.6.1)

4.2.132 draughting_pre_defined_colour

aim331: draughting_pre_defined_colour (no specific test purposes)

4.2.133 draughting_pre_defined_curve_font

aim332: draughting_pre_defined_curve_font (no specific test purposes)

4.2.134 fill_area_style

aim333: fill_area_style with **fill_styles** having at least one element as fill_area_style_colour (see 6.6.2)

aim334: fill_area_style with **name** as label (see 6.6.2)

4.2.135 fill_area_style_colour

aim335: fill_area_style_colour with **fill_colour** as colour (see 6.6.2)

aim336: fill_area_style_colour with **name** as label (see 6.6.2)

4.2.136 light_source

aim337: light_source with **light_colour** as colour (see 6.6.1)

aim338: light_source as **light_source_spot** (see 6.6.3)

aim339: light_source as **light_source_positional** (see 6.6.1)

aim340: light_source as **light_source_directional** (see 6.6.2)

aim341: light_source as **light_source_ambient** (see 6.6.1)

4.2.137 light_source_ambient

aim342: light_source_ambient (no specific test purposes) (see 6.6.1)

4.2.138 light_source_directional

aim343: light_source_directional with **orientation** as direction (see 6.6.2)

4.2.139 light_source_positional

aim344: light_source_positional with **position** as cartesian_point (see 6.6.1)

4.2.140 light_source_spot

aim345: light_source_spot with **spread_angle** as positive_plane_angle_measure (see 6.6.3)

aim346: light_source_spot with **orientation** as direction (see 6.6.3)

aim347: light_source_spot with **position** as cartesian_point (see 6.6.3)

4.2.141 mechanical_design_geometric_presentation_area

aim348: mechanical_design_geometric_presentation_area (no specific test purposes)

4.2.142 mechanical_design_geometric_presentation_representation

aim349: mechanical_design_geometric_presentation_representation (no specific test purposes)

4.2.143 mechanical_design_shaded_presentation_area

aim350: mechanical_design_shaded_presentation_area (no specific test purposes)

4.2.144 mechanical_design_shaded_presentation_representation

aim351: mechanical_design_shaded_presentation_representation (no specific test purposes)

4.2.145 over_riding_styled_item

aim352: over_riding_styled_item with **over_ridden_style** as styled_item

4.2.146 planar_box

aim353: planar_box with **placement** as axis2_placement_3d

aim354: planar_box with **placement** as axis2_placement_2d

4.2.147 planar_extent

aim355: planar_extent with **size_in_y** as length_measure

aim356: planar_extent with **size_in_x** as length_measure

aim357: planar_extent as **planar_box**

4.2.148 point_style

aim358: point_style with **marker_colour** as colour (see 6.6.2)

aim359: point_style with **marker_size** as measure_with_unit

aim360: point_style with **marker_size** as positive_length_measure (see 6.6.2)

aim361: point_style with **marker** = dot (see 6.6.2)

aim362: point_style with **marker** = x

aim363: point_style with **marker** = plus

aim364: point_style with **marker** = asterisk

aim365: point_style with **marker** = ring

aim366: point_style with **marker** = square

aim367: point_style with **marker** = triangle

aim368: point_style with **name** as label (see 6.6.2)

4.2.149 pre_defined_colour

aim369: pre_defined_colour as **draughting_pre_defined_colour**

4.2.150 pre_defined_curve_font

aim370: pre_defined_curve_font as **draughting_pre_defined_curve_font**

4.2.151 pre_defined_item

aim371: pre_defined_item with **name** as label

aim372: pre_defined_item as **pre_defined_curve_font**

aim373: pre_defined_item as **pre_defined_colour**

4.2.152 presentation_area

aim374: presentation_area as **mechanical_design_shaded_presentation_area**

aim375: presentation_area as **mechanical_design_geometric_presentation_area**

4.2.153 presentation_representation

aim376: presentation_representation as **presentation_view**

aim377: presentation_representation as **presentation_area**

4.2.154 presentation_size

aim378: presentation_size with **size** as planar_box

aim379: presentation_size with **unit** as presentation_area

aim380: presentation_size with **unit** as presentation_view

4.2.155 presentation_style_assignment

aim381: presentation_style_assignment with **styles** having at least one element as surface_style_usage (see 6.6.2)

aim382: presentation_style_assignment with **styles** having at least one element as curve_style (see 6.6.2)

aim383: presentation_style_assignment with **styles** having at least one element as point_style (see 6.6.2)

aim384: presentation_style_assignment as **presentation_style_by_context**

4.2.156 presentation_style_by_context

aim385: presentation_style_by_context with **style_context** as representation_item (see 6.6.5)

aim386: presentation_style_by_context with **style_context** as representation (see 6.6.2)

4.2.157 presentation_view

aim387: presentation_view (no specific test purposes)

4.2.158 styled_item

aim388: styled_item with **item** as representation_item (see 6.6.2)

aim389: styled_item with **styles** having at least one element as presentation_style_assignment (see 6.6.2)

aim390: styled_item as **over_riding_styled_item**

4.2.159 surface_rendering_properties

aim391: surface_rendering_properties with **rendered_colour** as colour (see 6.6.1)

4.2.160 surface_side_style

aim392: surface_side_style with **styles** having at least one element as surface_style_rendering (see 6.6.4)

aim393: surface_side_style with **styles** having at least one element as surface_style_parameter_line (see 6.6.2)

aim394: surface_side_style with **styles** having at least one element as surface_style_control_grid (see 6.6.2)

aim395: surface_side_style with **styles** having at least one element as surface_style_segmentation_-curve (see 6.6.2)

aim396: surface_side_style with **styles** having at least one element as surface_style_silhouette (see 6.6.2)

aim397: surface_side_style with **styles** having at least one element as surface_style_boundary (see 6.6.2)

aim398: surface_side_style with **styles** having at least one element as surface_style_fill_area (see 6.6.2)

aim399: surface_side_style with **name** as label (see 6.6.2)

4.2.161 surface_style_boundary

aim400: surface_style_boundary with **style_of_boundary** as curve_style_rendering

aim401: surface_style_boundary with **style_of_boundary** as curve_style (see 6.6.2)

4.2.162 surface_style_control_grid

aim402: surface_style_control_grid with **style_of_control_grid** as curve_style_rendering

aim403: surface_style_control_grid with **style_of_control_grid** as curve_style (see 6.6.2)

4.2.163 surface_style_fill_area

aim404: surface_style_fill_area with **fill_area** as fill_area_style (see 6.6.2)

4.2.164 surface_style_parameter_line

aim405: surface_style_parameter_line with **direction_counts** having at least one element as v_-direction_count (see 6.6.2)

aim406: surface_style_parameter_line with **direction_counts** having at least one element as u_-direction_count (see 6.6.2)

aim407: surface_style_parameter_line with **style_of_parameter_lines** as curve_style_rendering

aim408: surface_style_parameter_line with **style_of_parameter_lines** as curve_style (see 6.6.2)

4.2.165 surface_style_reflectance_ambient

aim409: surface_style_reflectance_ambient as **surface_style_reflectance_ambient_diffuse** (see 6.6.3)

4.2.166 surface_style_reflectance_ambient_diffuse

aim410: surface_style_reflectance_ambient_diffuse as **surface_style_reflectance_ambient_diffuse_-specular** (see 6.6.4)

4.2.167 surface_style_reflectance_ambient_diffuse_specular

aim411: surface_style_reflectance_ambient_diffuse_specular with **specular_colour** as colour (see 6.6.4)

4.2.168 surface_style_rendering

aim412: surface_style_rendering with **surface_colour** as colour (see 6.6.1)

aim413: surface_style_rendering with **rendering_method** = normal_shading (see 6.6.4)

aim414: surface_style_rendering with **rendering_method** = dot_shading (see 6.6.3)

aim415: surface_style_rendering with **rendering_method** = colour_shading (see 6.6.2)

aim416: surface_style_rendering with **rendering_method** = constant_shading (see 6.6.1)

aim417: surface_style_rendering as **surface_style_rendering_with_properties** (see 6.6.1)

4.2.169 surface_style_rendering_with_properties

aim418: surface_style_rendering_with_properties with **properties** having at least one element as surface_style_transparent (see 6.6.1)

aim419: surface_style_rendering_with_properties with **properties** having at least one element as surface_style_reflectance_ambient (see 6.6.1)

4.2.170 surface_style_segmentation_curve

aim420: surface_style_segmentation_curve with **style_of_segmentation_curve** as curve_style_rendering

aim421: surface_style_segmentation_curve with **style_of_segmentation_curve** as curve_style (see 6.6.2)

4.2.171 surface_style_silhouette

aim422: surface_style_silhouette with **style_of_silhouette** as curve_style_rendering

aim423: surface_style_silhouette with **style_of_silhouette** as curve_style (see 6.6.2)

4.2.172 surface_style_transparent

aim424: surface_style_transparent (no specific test purposes)

4.2.173 surface_style_usage

aim425: surface_style_usage with **style** as surface_side_style (see 6.6.2)

aim426: surface_style_usage with **side** = both (see 6.6.2)

aim427: surface_style_usage with **side** = negative

aim428: surface_style_usage with **side** = positive

4.2.174 view_volume

aim429: view_volume with **view_window** as planar_box (see 6.6.1)

aim430: view_volume with **view_volume_sides_clipping** = true (see 6.6.1)

aim431: view_volume with **view_volume_sides_clipping** = false (see 6.6.5)

aim432: view_volume with **back_plane_clipping** = true (see 6.6.1)

aim433: view_volume with **back_plane_clipping** = false (see 6.6.5)

aim434: view_volume with **back_plane_distance** as length_measure (see 6.6.1)

aim435: view_volume with **front_plane_clipping** = true (see 6.6.1)

aim436: view_volume with **front_plane_clipping** = false (see 6.6.5)

aim437: view_volume with **front_plane_distance** as length_measure (see 6.6.1)

aim438: view_volume with **view_plane_distance** as length_measure (see 6.6.1)

aim439: view_volume with **projection_point** as cartesian_point (see 6.6.1)

aim440: view_volume with **projection_type** = parallel (see 6.6.1)

aim441: view_volume with **projection_type** = central (see 6.6.5)

4.3 Domain test purposes

4.3.1 Advanced_elementary_or_faceted

other1: any instance of shape_representation in the model shall be either an **advanced_brep_shape_representation**, an **elementary_brep_shape_representation**, or a **faceted_brep_shape_representation**; no other subtypes shall occur. (see 6.3.1, 6.2.1, 6.1.1)

other2: Polyline and b_spline curve shall be the only permissible subtypes of bounded curve used in defining an advanced B-rep.

other3: Polyline shall be the only permissible subtype of bounded curve used in defining an elementary B-rep.

other4: b_spline_surface shall be the only permissible subtype of bounded surface used in defining an advanced B-rep.

other5: **advanced_brep_shape_representation** with context as **geometric_context** with **items** as **manifold_solid_brep**. (see 6.3.1.)

other6: **advanced_brep_shape_representation** with context as **geometric_context** with **items** as **mapped_item**; (see 6.3.7.)

other7: **advanced_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 6.3.7.)

other8: elementary_brep_shape_representation with context as **geometric_context** with **items** as **manifold_solid_brep**. (see 6.2.1.)

other9: elementary_brep_shape_representation with context as **geometric_context** with **items** as **mapped_item**; (see 6.2.6.)

other10: 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 6.2.6.)

other11: faceted_brep_shape_representation with context as **geometric_context** with **items** as **faceted_brep**. (see 6.1.1.)

other12: faceted_brep_shape_representation with context as **geometric_context** with **items** as **mapped_item**. (see 6.1.4.)

other13: faceted_brep_shape_representation with context as **geometric_context** with **items** as two or more items as **faceted_brep**, or **mapped_item**, or **axis2_placement_3d**, including at least one **axis2_placement_3d**. (see 6.1.4.)

other14: mapped_item with **mapping_target** as **cartesian_transformation_operator_3d**. (see 6.3.14, 6.1.5.)

other15: assembly_component_usage with assembly as **SELF\product_definition_relationship.relating_product_definition** as **product_definition** related to an instance of **shape_representation** by **product_definition_shape** and **shape_definition_representation**. (see 6.5.4.)

other16: assembly_component_usage with component as **SELF\product_definition_relationship.related_product_definition** as **product_definition** related to an instance of **shape_representation** by **product_definition_shape** and **shape_definition_representation**. (see 6.5.4.)

5 General test purposes and verdict criteria

General test purposes are statements of requirements that apply to all abstract test cases, all preprocessor abstract test cases, or all postprocessor abstract test cases. General verdict criteria are the means for evaluating whether the general test purposes are met. General verdict criteria shall be evaluated as a part of every executable test case to which they apply.

5.1 General test purposes

The following are the general test purposes for this part of ISO 10303:

g1: The output of an IUT shall preserve all the semantics defined by the input model.

g2: The output of a preprocessor shall conform to the implementation method the IUT claims conformance to.

NOTE 1 Any exchange file produced by a preprocessor shall be satisfy the requirements of schema conformance, as defined in ISO 10303-21.

g3: The instances in the output of a preprocessor shall be encoded according to the AIM EXPRESS long form as defined in Annex A of ISO 10303-204.

g4: The output of a preprocessor shall encode the input model according to the mapping tables presented in clause 5 of ISO 10303-204.

g5: A postprocessor shall accept input data which is encoded according to the implementation method the IUT claims conformance to.

g6: No redundancy in geometric models: all geometric constructs defined shall be used in the construction of a surface model.

g7: Complete topological information.

NOTE 2 For the topological information to be complete each shell of a B-rep model shall be defined by faces of type face_surface, each edge shall be of type edge_curve and trimmed by vertex_points.

g8: No trimmed geometry.

NOTE 3 All trimming shall use topological constructs, trimmed_curve and bounded_surface entities, other than b_spline_surface, shall not be used to define edges or faces.

g9: Simple and direct geometric representation. For any given geometry the representation shall be as simple and direct as possible.

In particular:

All lines, conics and elementary surfaces shall be represented as such; B-spline curves and surfaces shall not be used for their representation.

Swept surfaces shall not be used to represent elementary geometry such as a spherical surface.

5.2 General verdict criteria for all abstract test cases

The following verdict criteria apply to all abstract test cases contained in this part of ISO 10303:

gvc1: The semantics of the input model are preserved in the output of the IUT.

gvc2: The model shall conform to the semantics of the part_204_brep_product_schema defined in ISO 10303-204. This includes the verification of all relevant formal propositions.

gvc3: For all instantiated entities the informal propositions shall be satisfied. In particular this applies to the informal propositions relating to manifold_solid_brep and closed_shell defined in ISO 10303-42.

gvc4: All geometric elements of the model shall be bounded by topology: explicit geometric bounding shall not be used.

5.3 General verdict criteria for preprocessor abstract test cases

The following verdict criteria apply to all preprocessor abstract test cases contained in this part of ISO 10303:

gvc5: The output of a preprocessor conforms to the implementation method the IUT claims conformance to.

gvc6: The instances in the output of a preprocessor are encoded according to the AIM EXPRESS long form as defined in Annex A of ISO 10303-204.

gvc7: The output of a preprocessor encodes the input model according to the mapping tables presented in clause 5 of ISO 10303-204.

gvc8: The model shall not contain any instance of a geometric_representation_item which is not used, directly or indirectly, as part of the definition of shape_representation.

gvc9: The model shall not contain any instance of a topological_representation_item which is not used, directly or indirectly, as part of the definition of shape_representation.

gvc10: The model shall not contain any instance of simple geometry which does not use the most direct form of representation.

5.4 General verdict criteria for postprocessor abstract test cases

The following verdict criteria apply to all postprocessor abstract test cases contained in this part of ISO 10303:

gvc11: A postprocessor accepts input data which is encoded according to the implementation method the IUT claims conformance to.

gvc12: For each geometric entity occurring in the model the semantics shall be preserved by the postprocessor as well as preserving the geometric shape.

EXAMPLE 1 A conic curve instance shall not be converted to a B-spline curve, a surface of revolution shall not be converted to a surface of any other type.

6 Abstract test cases

This clause specifies the abstract test cases for this part of ISO 10303. Each abstract test case addresses one or more test purposes from clause 4. All the test purposes addressed by the test case are referenced explicitly, in the other and AIM test purposes covered sections, or indirectly, through the verdicted rows of the preprocessor input specification table.

The abstract test cases are organised according to the unit of functionality in ISO 10303-204 to which they relate. The title of each test case includes an alphabetic code which identifies the UoF and a reference number. This alphanumeric identifier is unique within this part of ISO 10303. The contents of an abstract test case are defined in clause 7 of ISO 10303-33.

Each abstract test case has two subclauses, one for preprocessor test information and one for all the post-processor test information. The preprocessor and postprocessor input specifications are mirror images of each other, i.e. they represent the same semantic information. The preprocessor input model is presented in the form of a table with five columns:

— The Id column is used to reference application objects for assertions and categorisations. It uses the same identifier as the test purpose associated with the application element in that row of the table.

  NOTE 1 In this column the application elements have an identifier which is generally of the form '@nnn.xxxx'. The numerical part of this (nnn) identifies the application element test purpose (aennn) to which it relates, alphanumeric component (xxxx), when present, provides the corresponding EXPRESS-I identifier.

— The V column specifies whether the element in that row of the table is verdicted in this test case. A blank indicates that it is not verdicted. A '*' indicates that it is verdicted using a derived verdict criteria. A number references a specific verdict criteria defined at the end of the table.

— The application Elements and Categorisations column identifies the particular application element or categorisation that is being defined by the table. For assertions, the role is specified in parenthesis.

— The value column specifies a specific value for the application element. For application objects and attributes the value column defines the semantic value for that element's instance in the input model. A #<number> in the column is a reference to an entity instance name in the postprocessor input specification where the mirror value is specified. For assertions, this column holds a link to the related application object. For categorisations, the Value column identifies the subtype application object. A 'not_present' indicates that the application element or categorisation is not present in the input model.

— The Req column specifies whether the value in the Value column is mandatory (M), suggested (S), or constrained (Cn). A suggested value may be changed by a test realiser. A mandatory value may not be changed due to rules in EXPRESS, the mapping table, or the requirements of the test purpose being verdicted. A constrained value may be modified according to the specific constraints specified at the end of the table.

The postprocessor input specifications are defined using ISO 10303-12 (the EXPRESS-I language. The values in the postprocessor specification are suggested unless declared mandatory or constrained by the preprocessor input table.

NOTE 2 Generally a C denotes a value which may be changed but only if at the same time other related values are changed to maintain the geometric integrity of the model being defined. Such changes are usually possible by varying the input parameters to the contexts used in the EXPRESS-I specification of the post-processor test case.

The abstract test case specifies all the verdict criteria which are used to assign a verdict during testing. Special verdict criteria for preprocessor and postprocessor testing are defined explicitly in each abstract test case subclause. The relevant derived verdict criteria for preprocessor testing are identified in the V column of the preprocessor input table.

6.1 Abstract test cases for faceted B-rep AIC

The post-processor abstract test cases in this clause are fully documented in EXPRESS-IȦ 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.

6.1.1 Test case fb1

Test case summary:

Test case fb1 is the most basic test case consisting of a single solid tetrahedron with one vertex at the origin of the coordinate system. All geometry is explicitly defined with no defaults and no sense reversals required. The definition of the outer shell is provided by the **tetrashell_instance** context using the original parameters.

Other test purpose coverage:

other1: any instance of shape_representation in the model shall be either an **advanced_brep_shape_representation**, an **elementary_brep_shape_representation**, or a **faceted_brep_shape_representation**; no other subtypes shall occur.

other11: **faceted_brep_shape_representation** with context as **geometric_context** with **items** as **faceted_brep**.

6.1.1.1 Preprocessor

Input specification:

see table 2

NOTE 1 See Test Case ps3 for details of product and part.

Constraints on values:

Table 2 – Preprocessor details: test case fb1

Id	V	Application Element	Value	Req
@200	*	B-rep	#tetrahedron	C2
@73	*	B-rep to closed_shell (as outer)	@200 to @9	M
@9	*	closed_shell	#tetrashell	C2
@107	*	face	#fs1	C1
@63	aim67	face to loop	@107 to @204	M
@205	*	face to surface	@107 to @109	M
@201	*	faceted_brep	#tetrahedron	M
@38.1	*	point	#origin	C1
@38.2	*	point	#p_x [100, 0, 0]	C1
@34	*	location	#a1	C1
@204	*	loop	#loop_x	C1
@102	*	plane	plane	C1
@204	*	poly_loop	#loop_y	C1
@9	*	shell	#tetrashell	M
@109	*	elementary_surface	#p1	C1
@1	*	shape_representation	#fbsr1	M

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The the outer shell of the faceted B-rep shall be the closed shell defined here.

Specific verdict criteria:

aim1: Model shall contain an example of **representation** as **shape_representation** as **faceted_brep_shape_representation**. **representation** shall contain a single **representation_item**; all WRs on **faceted_brep_shape_representation** shall be satisfied;

other11: Model shall contain an example of**faceted_brep_shape_representation** with context as **geometric_context** with **items** as **faceted_brep**.

aim5: Model shall contain an example of **faceted_brep** with **outer** (voids absent) as **closed_shell**. (NOT **oriented_closed_shell** subtype.)

aim26: Model shall contain an example of **closed_shell** with **cfs_faces** as a SET of more than one **face_surface**.

aim68: Model shall contain an example of **face_surface** with **same_sense** = TRUE. **same_sense** attribute of each face surface in model shall be TRUE.

aim10: Model shall contain an example of **surface** (as **elementary_surface**) as **plane** with **position** as **axis2_placement_3d** with **axis** present. Each **axis2_placement_3d** in model shall have **axis** attribute.

aim8: Model shall contain an example of **surface** (as **elementary_surface**) as **plane** with **position** as **axis2_placement_3d** with **ref_direction** present. Each **axis2_placement_3d** shall have **ref_direction** attribute.

Extra details:

Create a **faceted_brep_shape_representation** consisting of a single **faceted_brep**. The **faceted_brep** should be in the form of a solid tetrahedron with one vertex at the origin and the adjacent edges along the coordinate axes. All faces shall be defined by **poly_loop**s and shall be of type **face_surface** with the surface geometry defined by a plane. A suitable set of dimensions is defined in the EXPRESS-I specification below, and in the context defined in C.2.1.

6.1.1.2 Postprocessor

AIM test purpose coverage:

aim1, aim2, aim3, aim4, aim5, aim7, aim8, aim10 aim14, aim26, aim36, aim53, aim58, aim61, aim62, aim63, aim65, aim66, aim67, aim68, aim70, aim71, aim78, aim91, aim92, aim95, aim97, aim176, aim178, aim193, aim194, aim195, aim196, aim260, aim261, aim266, aim273, aim275, aim276, aim277, aim286, aim290, aim294, aim295.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

NOTE 2 The **tetrashell_instance** context is used, in its simplest form with default values, to define the faces of the B-rep.

```
*)
TEST_CASE example_fbrep_1; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep3 : shape_definition_representation ;
    shell_object : closed_shell ;
    tetrahedron  : faceted_brep ;
    fbsr1 : faceted_brep_shape_representation ;
    its_units : named_unit ;
```

```
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
  END_LOCAL;

    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;

    CALL tetrashell_instance ; -- uses default values, so no WITH
      IMPORT (shell_object := @tetrashell; ) ;
    END_CALL;

      its_units := named_unit(len_exp) || length_unit() ||
                              si_unit (’milli’, ’metre’) ;

      its_context := geometric_representation_context
                     (’context_1’, ’context_for_tetrahedron’, 3) ||
                     global_unit_assigned_context ( [its_units] ) ;

      tetrahedron := faceted_brep (’tetrahedron’, shell_object) ;

      fbsr1 := faceted_brep_shape_representation
                 (’fbsr1’,  [tetrahedron], its_context );
      shape1_def_rep3 := shape_definition_representation
                       (shape_1_def, fbsr1 );


  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim1: All WRs on **faceted_brep_shape_representation** shall be verified.

other11: model created shall contain no vertices,
model shall contain no edges,
model shall contain no curves,
length units correctly interpreted,
global_unit_assigned_context correctly interpreted.

aim36: cfs_faces attribute shall be a SET of more than one face_surface these faces shall intersect on lines joining points.

aim70: plane defining **face_geometry** for each **face_surface** shall pass through all points defining bounding **poly_loop**.

aim92: Normal to each **plane** shall point out of solid.

aim8: Set **P** of 3 axes for each **axis2_placement_3d** shall be correctly derived.

aim266: Postprocessor shall correctly interpret **shape_representation** as **faceted_brep_shape_-representation**.

6.1.2 Test case fb2

Test case summary:

Test case fb2 is designed to test the definition of a faceted B-rep containing one or more voids. The **tetrashell_instance** context is used with different parameters to define the outer shell and the void shells. The result is a hollow tetrahedral solid with void(s) of a similar shape.

NOTE 1 If required this test can easily be modified to test geometric precision by varying the parameters to define voids which 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.

6.1.2.1 Preprocessor

Input specification:

See table 3

NOTE 1 See Test Case ps3 for details of product and part.

NOTE 2 See Test Case fb1 for details of tetrashell as outer shell.

Constraints on values:

C1: The **outer** attribute of the first B-rep shall be the identifier of the outer shell, the **voids** attribute shall be the identifier of the first void shell.

C2: The **items** attribute of the **faceted_brep_shape_representation** entity shall consist only of the identifier of the B-rep with voids. The voids attribute shall contain the identifiers of the two void shells.

Specific verdict criteria:

ae202: Model shall contain an example of Faceted_brep with one void shell. **items** attribute of **representation** fbsr1 shall be **faceted_brep** and **brep_with_voids** subtype.
outer attribute shall be **closed_shell**, **voids** shall be a SET of one **oriented_closed_shell**, void shell shall not intersect outer shell.

Table 3 – Preprocessor details: test case fb2

Id	V	Application Element	Value	Req
@202	*	B-rep	#tetra_with_void	M
@56.1	*	B-rep to closed_shell	@202 to [@11, @11.1]	C1
@203	*	B-rep	faceted_brep AND brep_with_voids	M
@56.2	*	B-rep to closed_shell	@203 to [@11, @11.1, @11.2]	C2
@9	*	shell	closed_shell #void1	C1
@11	*	closed_shell	#tetrashell	C1
@54	aim81	void as closed_shell	#void1	C1
@55	aim82	void as closed_shell	#void2	C1
@206	*	shape_representation	#fbsr2	M
	*	B-rep as items	@203	C2

aim6: Model shall contain an example of **faceted_brep** and **brep_with_voids** subtype with outer as **closed_shell** and voids as a SET of one **oriented_closed_shell**. (voids present)

aim13: Model shall contain an example of **faceted_brep** and **brep_with_voids** subtype with outer as **closed_shell** and voids as a SET of more than one **oriented_closed_shell**. (voids present) Void shells shall not intersect each other, or outer.

aim81: Model shall contain an example of **oriented_closed_shell** with **orientation** = FALSE.

Extra details:

This test requires the creation of a **faceted_brep_shape_representation** consisting of a single **faceted_brep**. The **faceted_brep** should be in the form of a hollow tetrahedron with one vertex at the origin and the adjacent edges along the coordinate axes. Two separate instances shall be created, one with a single inner void, the other with 2 voids. Each void shell shall be of a similar shape to the outer shell and located inside the body. The voids shall not intersect each other or the outer shell. A suitable set of dimensions is defined in the EXPRESS-I specification below, and in the context defined in C.2.1.

6.1.2.2 Postprocessor

AIM test purpose coverage:

aim6, aim13, aim81, aim82

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

NOTE 2 The **tetrashell_instance** context is used, in its simplest form with default values, to define the outer shell of the B-rep; it is re-used, with different parameters, to define the void shells.

```
*)
TEST_CASE example_fbrep_2; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    prod2_name : STRING := 'prod2_name';
    shape_1_def, shape_2_def : product_definition_shape ;
    shape1_def_rep, shape2_def_rep : shape_definition_representation ;
    shell_object, hollow1, hollow2 : closed_shell ;
    void1, void2 : oriented_closed_shell;
    tetra_with_void : manifold_solid_brep ;
    tetra_with_voids : manifold_solid_brep ;
    fbsr1, fbsr2 : faceted_brep_shape_representation ;
    its_context : representation_context ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_units : named_unit ;
  END_LOCAL;

    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;

    CALL tetrashell_instance ; -- uses default values, so no WITH
      IMPORT (shell_object := @tetrashell; );
    END_CALL;

    CALL tetrashell_instance ;
(* parameters re-set for dimensions (large void) *)
      IMPORT (hollow1 := @tetrashell; );
      WITH (orc := 20; lx := 50; ly := 50; lz := 50;);
    END_CALL;

    CALL tetrashell_instance ;
  (* parameters re-set for dimensions (small void) *)
      IMPORT (hollow2 := @tetrashell; );
      WITH (orc := 5; lx := 20; ly := 20; lz := 20;);
    END_CALL;

    CALL basic_product_structure ;  -- parameters for second product.
```

```
      IMPORT (shape_2_def := @prod_def_shape; );
      WITH   (prod_name := @prod2_name;
              pdef_desc := 'test product definition 2';
              propd_desc := 'shape of test product 2';
              prod_name :=  'second test product';
              prod_id   := 'P02' ;
              pdf_id    := 'PDF02' ; );
    END_CALL;

    void1 := oriented_closed_shell ('void1', hollow1, FALSE) ;
    void2 := oriented_closed_shell ('void2', hollow2, FALSE) ;

  tetra_with_void := faceted_brep ('tetra_with_void', shell_object) ||
                       brep_with_voids ([void1]) ;

 tetra_with_voids := faceted_brep ('tetra_with_voids', shell_object) ||
                         brep_with_voids ([void1, void2]) ;

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

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


    fbsr1 := faceted_brep_shape_representation
             ( 'fbsr1', [tetra_with_void], its_context ) ;

    fbsr2 := faceted_brep_shape_representation
             ('fbsr2',  [tetra_with_voids], its_context ) ;

   shape1_def_rep := shape_definition_representation
                        (shape_1_def, fbsr1 );
  shape2_def_rep := shape_definition_representation
                        (shape_2_def, fbsr2 );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim6: Complex subtype faceted_brep and brep_with_voids shall be correctly interpreted.

aim13: faceted_brep with more than one void correctly interpreted, void shells shall not intersect each other, or outer.

aim81: Normal to each void shell shall point into void.

6.1.3 Test case fb3

Test case summary:

Test case fb3 is designed to test the abilities to define faces with inner loops and also to test the use of defaults and sense reversals. The test object is in the form of a rectangular block with a triangular through hole and a triangular depression in the top face. All geometry and topology is defined in the text case.

6.1.3.1 Preprocessor

Input specification:

See table 4.

NOTE 1 See Test Case ps3 for details of product and part.

Table 4 – Preprocessor details: test case fb3

Id	V	Application Element	Value	Req
@201 @73	* 	B-rep B-rep to shell (as outer)	#block @201 to @9	M
@107 @32	* aim63 * *	face as face_surface face to bounds face to surface face to loop	#fsp3 @107 to [#top,#t1bd, #t2bd] @107 to #pap3 #fsp3 to #loopt1 via #t1bd	M C1 M M
@31	aim64	face	#fsq4	M
@34.1	aim9	location as axis2_placement_3d	#aq4	M
@34.2	aim11	location as axis2_placement_3d	#ar3	M
@204	aim4	loop	#loopt1	C1
@9		shell	#blockshell	C1
@1	*	shape_representation	#fbsr	M
@102 @108	 *	plane plane.location	#psq4 @34.1	C1 C1

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Specific verdict criteria:

aim7: Model shall contain an example of **face** as **face_surface** with **bounds** as SET of at least two **face_bound**s (including one **face_outer_bound**) with **bound** as **poly_loop** and **orientation** = FALSE.

aim9: Model shall contain an example of **surface** (as **elementary_surface**) as **plane** with **position** as **axis2_placement_3d** with **ref_direction** absent.

aim11: Model shall contain an example of **surface** (as **elementary_surface**) as **plane** with **position** as **axis2_placement_3d** with **axis** absent.

aim61: Model shall contain an example of **face** as **face_surface** with **bounds** as SET of at least two **face_bound**s (including one **face_outer_bound**) with **bound** as **poly_loop** and **orientation** = TRUE.

aim64: Model shall contain an example of **face** as **face_surface** with **bounds** as SET of one **face_bound** as **face_outer_bound** with **bound** as **poly_loop** and **orientation** = FALSE.

aim67: Model shall contain an example of **face** as **face_surface** with **bounds** as SET of one **face_bound** as **face_outer_bound** with **orientation** = TRUE.

Extra details:

Create a **faceted_brep_shape_representation** consisting of a single **faceted_brep**. The **faceted_brep** should be in the form of a rectangular block with edges parallel to the coordinate axes. The block should have a triangular through hole and a triangular depression in one face. Some loops shall be created with orientation false. For some faces the sense of the **face_geometry** shall be false. Default values shall be used for some attributes of **axis2_placement_3d** when creating an exchange model. Some geometric details are described in table 4 and a full specification is provided in EXPRESS-I.

6.1.3.2 Postprocessor

AIM test purpose coverage:

aim7, aim9, aim11, aim61, aim64, aim67, aim69

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

```
*)
TEST_CASE example_fbrep_3; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'block_with_hole' ;
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation ;
    origin, px, py, pxy, pz, pxz, pyz, pzyz   : cartesian_point;
    q, qx, qy, qz, qxz, qyz                   : cartesian_point;
    r, rx, ry, rz, rxz, ryz                   : cartesian_point;
    neg_x, neg_y, neg_z, slope, pos_x, pos_y  : direction;
    loopb, loopt, loopf, loopbk, loopl, loopr : poly_loop;
    loopbi, loopmid, loopt1, loopt2, loopqf   : poly_loop;
    loopql, loopqs, looprb, looprr, looprs    : poly_loop;
    a1, a2, a3, ap3, ar3, ap1, ap2, ar1, ar2  : axis2_placement_3d;
    ar4, aq1, aq2, aq4                        : axis2_placement_3d;
    pa1, pa2, pa3, pap1, pap2, pap3, paq1     : plane;
    paq2, paq4, par1, par2, par3, par4        : plane;
    bottom, top, front, back, left, right     : face_outer_bound;
    qfbd, qlbd, qsbd, rbbd, rrbd, rsbd, midbd : face_outer_bound;
    bibd, t1bd, t2bd                          : face_bound;
    fs1, fs2, fs3, fsp1, fsp2, fsp3, fsq1     : face_surface;
    fsq2, fsq4, fsr1, fsr2, fsr3, fsr4        : face_surface;
    blockshell                                : closed_shell ;
    block                                     : faceted_brep ;
    fbsr                        : faceted_brep_shape_representation ;
    its_context : representation_context ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_units : named_unit ;
  END_LOCAL;

    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;

(* Cartesian_points on boundary: *)
origin := cartesian_point ('origin', [0, 0, 0]) ;
px := cartesian_point ('px', [50, 0, 0]) ;
py := cartesian_point ('py', [0, 50, 0]) ;
pxy := cartesian_point ('pxy', [50, 50, 0]) ;
pz := cartesian_point ('pz', [0, 0, 100]) ;
pxz := cartesian_point ('pxz', [50, 0, 100]) ;
```

```
  pyz := cartesian_point (’pyz’, [0, 50, 100]) ;
  pxyz := cartesian_point (’pxyz’, [0, 0, 100]) ;

  q   := cartesian_point (’q’, [10,10, 0]) ;
  qx := cartesian_point (’qx’, [25,10, 0]) ;
  qy := cartesian_point (’qy’, [10, 25, 0]) ;
  qz := cartesian_point (’qz’, [10, 10, 100]) ;
  qxz := cartesian_point (’qxz’, [25, 10, 100]) ;
  qyz := cartesian_point (’qyz’, [10, 25, 100]) ;

  r   := cartesian_point (’r’, [45, 45, 50]) ;
  rx := cartesian_point (’rx’, [35, 45, 50]) ;
  ry := cartesian_point (’ry’, [45, 35, 50]) ;
  rz := cartesian_point (’rz’, [45, 45, 100]) ;
  rxz := cartesian_point (’rxz’, [35, 45, 100]) ;
  ryz := cartesian_point (’ryz’, [45, 35, 100]) ;

  neg_x := direction (’neg_x’, [-1, 0, 0]) ;
  neg_y := direction (’neg_y’, [0, -1, 0]) ;
  neg_z := direction (’neg_z’, [0, 0, -1]) ;
  slope := direction (’slope’, [1, 1, 0 ]) ;
  pos_x := direction (’pos_x’, [1,  0, 0]) ;
  pos_y := direction (’pos_y’,[0, 1, 0 ]) ;

(*  outer loops:      *)
  loopb := poly_loop (’loopb’, [origin, px, pxy, py]) ;
  loopt := poly_loop (’loopt’, [origin, pxz, pxyz, pyz]) ;
  loopf := poly_loop (’loopf’, [origin, px, pxz, pz]) ;
  loopbk := poly_loop (’loopbk’, [py, pxy, pxyz, pyz]) ;
  loopl  := poly_loop (’loopl ’, [origin, pz, pyz, py]) ;
  loopr  := poly_loop (’loopr ’, [px, pxz, pxyz, pxy]) ;

(* inner loops (triangular):    *)
  loopbi  := poly_loop (’loopbi ’, [q, qx, qy]) ;
  loopmid := poly_loop (’loopmid’, [r, rx, ry]) ;
  loopt1  := poly_loop (’loopt1’,  [qz, qxz, qyz]) ;
  loopt2  := poly_loop (’loopt2’,  [rz, rxz, ryz]) ;

(*   inside loops (rectangular):     *)
  loopqf  := poly_loop (’loopqf’, [q, qx, qxz, qz]) ;
  loopql  := poly_loop (’loopql’, [q, qz, qyz, qy]) ;
  loopqs  := poly_loop (’loopqs’, [qx, qxz, qyz, qy]) ;
  looprb  := poly_loop (’looprb’, [r, rx, rxz, rz]) ;
  looprr  := poly_loop (’looprr’, [r, rz, ryz, ry]) ;
  looprs  := poly_loop (’looprs’, [rx, rxz, ryz, ry]) ;
```

```
(* axis_placements {number determines if axis parallel to x, y or z} *)
   a1 := axis2_placement_3d ('a1', origin, neg_x, neg_y) ;
   a2 := axis2_placement_3d ('a2', origin, neg_y, neg_x) ;
   a3 := axis2_placement_3d ('a3', origin, neg_z, ?) ;
   ap3 := axis2_placement_3d ('ap3', pz,  ?, ?) ;
   ar3 := axis2_placement_3d ('ar3', r,  ?,  ?) ;
   ap1 := axis2_placement_3d ('ap1', px, pos_x, ?) ;
   ap2 := axis2_placement_3d ('ap2', py, pos_y, ?) ;
   ar1 := axis2_placement_3d ('ar1', r, pos_x, ?) ;
   ar2 := axis2_placement_3d ('ar2', r, pos_y, ?) ;
   ar4 := axis2_placement_3d ('ar4', r, slope, ?) ;
   aq1 := axis2_placement_3d ('aq1', q, pos_x, ?) ;
   aq2 := axis2_placement_3d ('aq2', q, pos_y, ?) ;
   aq4 := axis2_placement_3d ('aq4', r, slope, ?) ;

(* plane defined for each axis placement: *)
  pa1 := plane ('pa1', a1) ;
  pa2 := plane ('pa2', a2) ;
  pa3 := plane ('pa3', a3) ;
  pap1 := plane ('pap1', ap1) ;
  pap2 := plane ('pap2', ap2) ;
  pap3 := plane ('pap3', ap3) ;
  paq1 := plane ('paq1', aq1) ;
  paq2 := plane ('paq2', aq2) ;
  paq4 := plane ('paq4', aq4) ;
  par1 := plane ('par1', ar1) ;
  par2 := plane ('par2', ar2) ;
  par3 := plane ('par3', ar3) ;
  par4 := plane ('par4', ar4) ;

(*   block outer bounds  *)
  bottom := face_outer_bound ('bottom', loopb,  FALSE) ;
  top    := face_outer_bound ('top', loopt,  TRUE) ;
  front  := face_outer_bound ('front', loopf,  TRUE) ;
  back   := face_outer_bound ('back', loopbk,  FALSE) ;
  left   := face_outer_bound ('left', loopl,  FALSE) ;
  right  := face_outer_bound ('right', loopr,  TRUE) ;

(*   inner face outer bounds  *)
  qfbd  := face_outer_bound ('qfbd', loopqf,  FALSE) ;
  qlbd   := face_outer_bound ('qlbd', loopql,  TRUE) ;
  qsbd  := face_outer_bound ('qsbd', loopqs,  FALSE) ;
  rbbd   := face_outer_bound ('rbbd', looprb,  FALSE) ;
  rrbd  := face_outer_bound ('rrbd', looprr,  TRUE) ;
  rsbd   := face_outer_bound ('rsbd', looprs,  TRUE) ;
  midbd := face_outer_bound ('midbd', loopmid,  TRUE) ;
```

```
(*   inner bounds    *)
  bibd  := face_bound ('bibd', loopbd,  TRUE) ;
  t1bd  := face_bound ('t1bd', loopt1,  FALSE) ;
  t2bd  := face_bound ('t2bd', loopt2,  FALSE) ;

(*  outer faces of block *)
  fs1 := face_surface ('fs1', [left], pa1,  TRUE) ;
  fs2 := face_surface ('fs2', [front],  pa2,  TRUE) ;
  fs3 := face_surface ('fs3', [bottom, bibd],  pa3,  TRUE) ;
  fsp1 := face_surface ('fsp1', [right], pap1, TRUE) ;
  fsp2 := face_surface ('fsp2', [back], pap2, TRUE) ;
  fsp3 := face_surface ('fsp3', [top, t1bd, t2bd], pap3, TRUE) ;

(*   inner faces
      hole:    *)
  fsq1 := face_surface ('fsq1', [qlbd], paq1, TRUE) ;
  fsq2 := face_surface ('fsq2', [qfbd], paq2,  TRUE) ;
  fsq4 := face_surface ('fsq4', [qsbd], paq4,  FALSE) ;
(*  depression:  *)
  fsr1 := face_surface ('fsr1', [rrbd], par1,  FALSE) ;
  fsr2 := face_surface ('fsr2', [rbbd], par2,  FALSE) ;
  fsr3 := face_surface ('fsr3', [midbd], par3,  TRUE) ;
  fsr4 := face_surface ('fsr4', [rsbd], par4,  TRUE) ;

blockshell := closed_shell ('blockshell', [fs1, fs2, fs3, fsp1, fsp2,
                     fsp3,fsq1, fsq2, fsq4, fsr1, fsr2, fsr3, fsr4]);

block := faceted_brep ('block', blockshell) ;


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


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


      fbsr := faceted_brep_shape_representation
       ('fbsr',  [block], its_context ) ;

   shape1_def_rep := shape_definition_representation
                        (shape_1_def, fbsr );
```

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

Specific verdict criteria:

aim9: ref_direction shall be correctly defaulted when processing **axis2_placement_3d**.

aim11: axis shall be correctly defaulted when processing **axis2_placement_3d**.

aim61: faces with inner bound shall be correctly processed, bounds shall be checked for intersections.

aim64: face_bound with orientation FALSE shall be correctly processed, final orientation shall always be correct with face to left.

aim69: face_surface with orientation FALSE shall be correctly processed, the **closed_shell** defined shall have all face normals pointing out of solid material.

6.1.4 Test case fb4

Test case summary:

Test case fb4 is designed to test the use of **mapped_item**s in the creation of a simple assembly of faceted 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 **tetrashell_instance** context to define the geometry and topology.

Other test purpose coverage:

other12: **faceted_brep_shape_representation** with context as **geometric_context** with **items** as **mapped_item**.

other13: **faceted_brep_shape_representation** with context as **geometric_context** with **items** as two or more items as **faceted_brep**, or **mapped_item**, or **axis2_placement_3d**, including at least one **axis2_placement_3d**.

6.1.4.1 Preprocessor

Input specification:

See table 5

NOTE 1 See Test Case ps3 for details of product and part.

NOTE 2 See Test Case fb1 for details of tetrashell as outer shell.

Table 5 – Preprocessor details: test case fb4

Id	V	Application Element	Value	Req
@201 @73		faceted_brep B-rep to closed_shell	#tetrahedron #tetrahedron to #tetrashell	C2
@200 @34.1 @34.2 @9	aim103 * * 	rotated B-rep axis2_placement_3d axis2_placement_3d shell	mapped_item = #tetrarot1 #newaxes #oldaxes #tetrashell	M M C1 C1
@200.1		shape_representation representation.items	#fbsr [#tetrahedron]	C2
@200.2	other12 *	shape_representation items	#fbsr1 [#tetrarot1]	M C3
@200.3	other13 *	shape_representation represention.items	#fbsrass [#tetrahedron, #tetrarot2, #oldaxes]	M C

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The **items** attribute of the **factored_brep_shape_representation** entity shall consist only of the identifier of the original B-rep.

C3: The **items** attribute of the second **factored_brep_shape_representation** entity shall consist only of the identifier of the rotated B-rep.

Specific verdict criteria:

other12: Model shall contain a **faceted_brep_shape_representation** with context as **geometric_context** with **items** as **mapped_item**.

other13: Model shall contain a **faceted_brep_shape_representation** with context as **geometric_context** with **items** as two or more items as **faceted_brep**, or **mapped_item**, or **axis2_placement_3d**, including at least one **axis2_placement_3d**.

Extra details:

This test requires the creation of a **faceted_brep_shape_representation** consisting of a single **faceted_brep**. The **faceted_brep** should be in the form of a solid tetrahedron with one vertex at the origin and the adjacent edges along the coordinate axes. This representation is then used in conjunction with the **mapped_item** entity to create, in the same **representation_context**, a representation consisting of a rotated copy of the original representation. In a separate **representation_context** a representation is cre-

ated consisting of the original **faceted_brep** and a mapped copy of the original representation. See the EXPRESS-I specification below and the context in C.2.1 for dimensional details.

6.1.4.2 Postprocessor

AIM test purpose coverage:

aim214, aim215, aim262, aim263, aim266

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

NOTE 2 Test case of mapped_item and 'assembly' using simple solid tetrahedron.

NOTE 3 Tetrahedron shell is created by using **tetrashell_instance** context with default parameters.

NOTE 4 **fbsr1** should be a rotated copy of **fbsr**.

NOTE 5 **fbsrass** should be equivalent to 2 copies of **fbsr** 'glued' together.

```
  *)
TEST_CASE example_fbrep_4; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    prod2_name : STRING := 'prod2_name';
    shape_1_def, shape_2_def : product_definition_shape ;
    shape1_def_rep, shape2_def_rep : shape_definition_representation ;
    origin : cartesian_point ;
    pos_z, neg_y  : direction ;
    refaxes, oldaxes, newaxes : axis_placement_3d;
    shell_object  : closed_shell ;
    tetrahedron : faceted_brep ;
    fbsr, fbsr1, fbsrass : faceted_brep_shape_representation ;
    grc1, grc2 : representation_context ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_units : named_unit ;
    tetrarot1, tetrarot2  : mapped_item ;
    mapping1, mapping2   : representation_map ;
  END_LOCAL;
```

```
    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;

    CALL basic_product_structure ;  -- parameters for second product.
      IMPORT (shape_2_def := @prod_def_shape; );
      WITH   (prod_name := @prod2_name;
              pdef_desc := 'test product definition 2';
              propd_desc := 'shape of test product 2';
              prod_name :=  'second test product';
              prod_id   := 'P02' ;
              pdf_id    := 'PDF02' ; );
    END_CALL;


   CALL tetrashell_instance ; -- uses default values, so no WITH
      IMPORT (shell_object := @tetrashell;
              origin := @origin;
              neg_y := @ neg_y; refaxes := @a1;);
    END_CALL;


tetrahedron := faceted_brep ('tetrahedron', shell_object) ;
its_units := named_unit(len_exp) || length_unit() ||
                              si_unit ('milli', 'metre') ;

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

grc2 := geometric_representation_context ('grc2',
      'context for rotated tetrahedron', 3) ||
      global_unit_assigned_context ( [its_units] ) ;

fbsr := faceted_brep_shape_representation('fbsr', [tetrahedron], grc1);

(*  Define axis_placements for use in mapping *)
oldaxes := axis2_placement_3d ('oldaxes', origin, ?,  ?) ;

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

newaxes := axis2_placement_3d ('newaxes', origin, pos_z, neg_y) ;

mapping1 := representation_map (refaxes, fbsr );
tetrarot1 := mapped_item ('tetrarot1', mapping1, newaxes );
```

```
(* Define representation using tetrarot1 only  *)
fbsr1 := faceted_brep_shape_representation('fbsr1', [tetrarot1], grc1);

(* Define representation which is an assembly of tetrahedron
                                                + mapped copy.*)

mapping2 := representation_map (oldaxes, fbsr );
tetrarot2 := mapped_item ('tetrarot2', mapping2, newaxes );

fbsrass := faceted_brep_shape_representation
        ('fbsrass', [tetrahedron, tetrarot2, oldaxes], grc2 ) ;

shape1_def_rep := shape_definition_representation
                       (shape_1_def, fbsr1 );
shape2_def_rep := shape_definition_representation
                       (shape_2_def, fbsrass );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

other12: After processing the solid tetrahedron defined by **mapped_item** shall be correctly defined and positioned.

other13: Two separate geometric representation contexts are created, fbsr and fbsr1 are spatially related and should touch at one point only, fbsr1 and fbsrass are not spatially related, the shells in fbsrass should not intersect but should coincide over a common face which is implicitly shared.

6.1.5 Test case fb5

Test case summary:

Test case fb5 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 **tetrashell_instance** context to define the geometry and topology.

Other test purpose coverage:

other12: **faceted_brep_shape_representation** with context as **geometric_context** with **items** as **mapped_-item**.

other14: **mapped_item** with **mapping_target** as **cartesian_transformation_operator_3d**.

6.1.5.1 Preprocessor

Input specification:

See table 6.

NOTE 1 See Test Case ps3 for details of product and part.

NOTE 2 See Test Case fb1 for details of tetrashell as outer shell.

Table 6 – Preprocessor details: test case fb5

Id	V	Application Element	Value	Req
@201		faceted_brep	#tetrahedron	C1
@200 @34	other14	B-rep axis2_placement_3d	mapped_item = #tetratrans #oldaxes	M C3
@9		shell	#tetrashell	C
@200.1		shape_representation representation.items	#fbsr #tetrahedron	C1 C1
@200.2	*	shape_representation representation.items	#fbsrass [#tetrahedron, #tetratrans]	M
@49 @50	* aim21	transformation transformation.scaling_factor	#transform 0.75	M C3

Constraints on values:

C1: The **items** attribute of the first **faceted_brep_shape_representation** entity shall consist only of the identifier of the B-rep defined by the tetrahedron.

C2: The transformed B-rep shall reference the first B-rep as source.

C3: The transformation is defined using the original reference axes and the transformation operator.

Specific verdict criteria:

aim21: Model shall contain an example of **cartesian_transformation_operator** as **cartesian_-transformation_operator_3d** with scale as REAL not equal to 1.0.

Extra details:

This test requires the creation of a **faceted_brep_shape_representation** consisting of a single **faceted_-brep**. The **faceted_brep** should be in the form of a solid tetrahedron with one vertex at the origin and the adjacent edges along the coordinate axes. 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 and scaled (not by 1.0) copy of the original representation

and the original **faceted_brep**. See the EXPRESS-I specification below and the context in C.2.1 for dimensional details.

6.1.5.2 Postprocessor

AIM test purpose coverage:

aim15, aim16, aim18, aim20, aim21, aim23, aim211, aim212, aim213.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

NOTE 2 Test case of **mapped_item** and 'assembly' using simple solid tetrahedron.

NOTE 3 Tetrahedron shell is created by using **tetrashell_instance** context with default parameters.

NOTE 4 **tetratrans** should be a scaled copy of **fbsr** after reflection in OZX plane.

NOTE 5 **fbsrass** should be equivalent to two tetrahedra 'glued' together.

```
  *)

TEST_CASE example_fbrep_5; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation ;
    origin : cartesian_point ;
    pos_z, neg_y, pos_x  : direction ;
    oldaxes : axis_placement_3d;
    transform : cartesian_transformation_operator_3d;
    shell_object  : closed_shell ;
    tetrahedron : faceted_brep ;
    fbsr, fbsrass : faceted_brep_shape_representation ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
     its_units : named_unit ;
    grc1, grc2 : representation_context ;
    tetratrans  : mapped_item ;
    mapping1  : representation_map ;
  END_LOCAL;
```

```
    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;

    CALL tetrashell_instance ; -- uses default values, so no WITH
      IMPORT (shell_object := @tetrashell;
              origin := @origin;
              neg_y := @ neg_y; );
    END_CALL;


tetrahedron := faceted_brep ('tetrahedron', shell_object) ;

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

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

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

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

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

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

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

transform := cartesian_transformation_operator_3d ('transform',
               'rotate and scale', pos_x, neg_y, origin, 0.75, pos_z );

fbsr := faceted_brep_shape_representation ('fbsr',
                                  [tetrahedron, oldaxes], grc1 ) ;

mapping1 := representation_map (oldaxes, fbsr );
(* tetratrans is a 75% scaled copy of original reflected in ZX plane *)
tetratrans := mapped_item ('tetrarot1', mapping1, transform );

(* Define representation which is an assembly of tetrahedron +
```

```
transformed (scaled and reflected) copy.*)


fbsrass := faceted_brep_shape_representation
        ('fbsrass', [tetrahedron, tetratrans], grc2 ) ;

shape1_def_rep := shape_definition_representation
                       (shape_1_def, fbsrass );
  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

other14: After processing the solid tetrahedron defined by **mapped_item** shall be correctly defined and positioned.

aim21: fbrass should consist of two solid tetrahedra which are in contact in the ZX plane. One is a 3/4 size copy of the other after reflection in this plane.

6.2 Abstract test cases for elementary B-rep

The post-processor abstract test cases in this clause are fully documented in EXPRESS-IȦ 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 1 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 eb1.

6.2.1 Test case eb1

Test case summary:

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. (See C.3.1.)

Other test purpose coverage:

other1: any instance of shape_representation in the model shall be either an **advanced_brep_shape_representation**,
an **elementary_brep_shape_representation**,
or a **faceted_brep_shape_representation**; no other subtypes shall occur.

other8: **elementary_brep_shape_representation** with context as **geometric_context** with **items** as **manifold_solid_brep**.

6.2.1.1 Preprocessor

Input specification:

See table 7

NOTE 1 See Test Case ps2 for details of product and part.

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The **items** attribute of the **elementary_brep_shape_representation** entity shall consist only of the identifier of the B-rep.

Specific verdict criteria:

ae33: Each Geometric_element in the model shall be used as part of the geometric description of a B-rep.

aim61: Model shall contain an example of face as face_surface with bounds as a SET of more than one face_bound with bound as an edge_loop and orientation FALSE.

aim63: Model shall contain an example of face as face_surface with bounds as a SET of one face_bound as face_outer_bound with orientation TRUE.

aim64: Model shall contain an example of face as face_surface with bounds as a SET of one face_bound as face_outer_bound with orientation FALSE.

aim76: Model shall contain an example of face as face_surface with bounds as a SET of more than one face_bound with bound as an edge_loop and orientation TRUE.

aim78: Model shall contain an example of manifold_solid_brep with outer (voids absent) as closed_-shell (NOT oriented_closed_shell subtype).

6.2.1.2 Postprocessor

AIM test purpose coverage:

aim8, aim10, aim14, aim25, aim26, aim27, aim28, aim32, aim33, aim36, aim37, aim39, aim41, aim44, aim45, aim46, aim47, aim48, aim49, aim53, aim55, aim57, aim58, aim59, aim60, aim61, aim62, aim63,

Table 7 – Preprocessor details: test case eb1

Id	V	Application Element	Value	Req
@113	*	elementary_B_rep	#ebsr	C2
@100	*	B-rep	manifold_solid_brep = #cysp_solid	M
@73	*	B-rep to closed_shell (as outer)	@100 to @9	
@6	*	circle	#circ	M
@8	*	circle.radius	25.0	C1
@7		circle.location	#a1	C1
@9	*	closed_shell	#shell_object	M
@11	aim10	shell to face	@9 to [#face_1, #face_2, #face_3]	C1
@12	*	conic as circle	#circ	M
@13	*	conic as ellipse	#elli	M
@26	*	curve	#elli	M
@58	*	curve to edge	@13 to @24	C1
@20	*	cylindrical_surface	#cyl	M
@21	*	cylindrical_surface.axis	#a1	C1
@22	*	cylindrical_surface.radius	25.0	C1
@23	*	direction	#dslope	C1
@24	*	edge as edge_curve	#edge2	C1
@69	*	edge to B-rep	@24 to @100	
@24	*	edge to edge curve	@24 to @13	
@25	*	edge to vertex	@24 to [@71, @71]	
@104	*	elementary_surface	#cyl	M
@26	*	ellipse	#elli	M
@28	*	ellipse.major_radius	25*rt(2)	C1
@29	*	ellipse.minor_radius	25.0	C1
@27	*	ellipse.location	#a2	S
@107.1	*	face	#curved_face	C1
@65	*	face to surface	@107.1 to @104	C1
@64		face.loops	[#bcylbot, #bcyltop]	C1
@62	*	face to face	#face_1 to #face_2	C1
@107.2	*	face	#face_2	C1
@404	*	global_unit	#its_units	M

Table 7 – concluded

Id	V	Application Element	Value	Req
@34	*	location as axis2_placement_3d	#a2	C1
@35	*	loop as edge_loop	#loopc	M
	*	loop to edge	@35 to #edge1	C1
@102	*	elementary_surface as plane	#p1	M
@38.1	*	point	#ctop	
@38.2	*	point	#epoint	
@105	*	spherical_surface	#sphere	M
@41	*	spherical_surface.centre	#origin	C1
@42	*	spherical_surface.radius	25.0	C1
@53	*	vertex	#verte	M
@71	*	vertex to point	@71 to @38.1	

aim64, aim65, aim66, aim67, aim68, aim70, aim71, aim76, aim78, aim83, aim85, aim88, aim89, aim90, aim91, aim92, aim95, aim96, aim97, aim103, aim106, aim260, aim261, aim267, aim273, aim275, aim276, aim277, aim286, aim290, aim294, aim295.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

NOTE 2 The **cylinder_sphere_shell** context is used, in its simplest form with default values, to define the faces of the B-rep.

```
*)
TEST_CASE example_ebrep_1; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    shell_object : closed_shell ;
    cysp_solid : manifold_solid_brep ;
    ebsr : elementary_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;
```

```
   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

   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 := named_unit(len_exp) || 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 );
      shape1_def_rep := shape_definition_representation
                       (shape_1_def, ebsr );

 END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim8: axis2_placement with **ref_direction** present shall be correctly interpreted to locate surface.

aim10: axis2_placement with **axis** present shall be correctly interpreted to locate surface.

aim26: Shell normals shall point out of solid.

aim36: Faces shall be connected along edges, no other face intersections shall occur.

aim45: edge with identical start and end vertices and **edge_geometry** as ellipse shall be correctly interpreted as closed ellipse.

aim46: edge with identical start and end vertices and **edge_geometry** as circle shall be correctly interpreted as a closed circle.

aim51: edge_curves and vertices of bounding **edge_loop**s shall lie on surface defining **face_geometry**. **[**aim55] Correct portion of **spherical_surface** shall be defined by **edge_loop**s.

aim57: Unbounded **cylindrical_surface** shall be bounded by **edge_loop**s.

aim58: bounding loops of face with **face_geometry** as **plane** shall be co-planar.

aim61: Multiple bounds shall be correctly interpreted to trim face.

aim63: Face geometry shall be correctly trimmed by **face_bound**.

aim64: **face_bound** with orientation FALSE shall be correctly interpreted to define correct portion of face surface.

aim64: **face_bound**s with different orientations shall be correctly interpreted.

aim103: All **vertex_point**s shall lie on **edge_curve**s.

aim106: All WRs on **elementary_brep_shape_representation** shall be verified.

other8: Length units shall be correctly interpreted, model re-created shall contain no **polyloop**s or **vertex_loop**s.

6.2.2 Test case eb2

Test case summary:

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. (See C.3.1.) The result is a hollow cylindrical solid with void(s) of a similar shape, or spherical.

NOTE 1 If required this test can easily be modified to test geometric precision by varying the parameters to define voids which 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.

6.2.2.1 Preprocessor

Input specification:

See table 8.

NOTE 1 See Test Case ps2 for details of product and part.

NOTE 2 See Test Case eb1 for details of shell_object.

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Table 8 – Preprocessor details: test case eb2

Id	V	Application Element	Value	Req
@113	*	elementary_B_rep	#ebsr1	C2
@113.1	*	elementary_B_rep	#ebsr2	C3
@56.1	aim13	B-rep	brep_with_voids = #cysp_with_void	M
@56.2	aim79	B-rep	brep_with_voids = #cysp_with_voids	M
@31	*	face	#sp_face	M
@65		face.face_geometry	#sph2	C1
@31	*	bounds	[#s_bound]	M
@38		point	#top_pt = (-5, -5, 5)	C1
@54.1	*	void	oriented_closed_shell = #void1	M
@55.1	*	void.orientation	FALSE	M
@54.2	*	void	oriented_closed_shell = #void2	C1
@55.2	*	void.orientation	FALSE	M
@105		spherical_surface	#sph2	C1
@42		spherical_surface.radius	10	C1
@41		spherical_surface.centre	(-5,-5,-5)	C1
@53		vertex	#top_v	C1
@36	aim77	loop as vertex_loop	#v_loop	M

C2: The **items** attribute of the first **elementary_brep_shape_representation** entity shall consist only of the identifier of the B-rep with a single void.

C3: The **items** attribute of the second **elementary_brep_shape_representation** entity shall consist only of the identifier of the B-rep with voids.

Specific verdict criteria:

aim13: Model shall contain an example of **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).

aim26: Model shall contain an example of **closed_shell** with **cfs_faces** as a SET of one **face_surface**.

aim77: Model shall contain an example of **loop** as **vertex_loop**.

aim79: Model shall contain an example of **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**.

aim81: Model shall contain an example of **oriented_closed_shell** with orientation as FALSE.

Extra details:

This test requires the creation of an **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 hemisphere 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.

6.2.2.2 Postprocessor

AIM test purpose coverage:

aim27, aim36, aim77, aim 79, aim81, aim82, aim96, aim104 Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

NOTE 2 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 3 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 part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    prod2_name : STRING := 'prod2_name';
    shape_1_def, shape_2_def : product_definition_shape ;
    shape1_def_rep, shape2_def_rep : shape_definition_representation ;
    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 ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    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 basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;

    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 -large void  *)
      IMPORT (hollow1 := @cyspshell; );
      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;

    CALL basic_product_structure ; -- parameters for second product.
      IMPORT (shape_2_def := @prod_def_shape; );
      WITH   (prod_name := @prod2_name;
              pdef_desc := 'test product definition 2';
              propd_desc := 'shape of test product 2';
              prod_name :=  'second test product';
              prod_id   := 'P02' ;
              pdf_id    := 'PDF02' ; );
    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 := named_unit(len_exp) || length_unit() ||
                              si_unit ('milli', 'metre') ;

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

    context2 := geometric_representation_context ,
                 ('context2', '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 ) ;

   shape1_def_rep := shape_definition_representation
                        (shape_1_def, ebsr1 );
   shape2_def_rep := shape_definition_representation
                        (shape_2_def, ebsr2 );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim13: Void shell shall not intersect outer shell, void shell shall be completely within outer shell.

aim79: 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.

aim80: Normal to void shells shall point into voids.

aim96: closed_shell with single face with **spherical_surface** geometry shall be correctly processed.

aim104: vertex_loop shall be correctly processed as **face_bound** to define face as complete spherical surface.

6.2.3 Test case eb3

Test case summary:

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. (See C.3.3.)

Other test purpose coverage:

other3: Polyline shall be the only permissible subtype of bounded curve used in defining an elementary B-rep.

6.2.3.1 Preprocessor

Input specification:

See table 9.

NOTE 1 See Test Case ps2 for details of product and part.

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Specific verdict criteria:

aim128: Model shall contain example of **elementary_surface** as **toroidal_surface**.

aim133: Model shall contain example of **edge** as **oriented_edge** with orientation as FALSE.

aim134: Model shall contain example of **edge_curve** with **edge_geometry** as **line**.

aim135: Model shall contain example of **edge_curve** with **edge_geometry** as **polyline**.

Extra details:

Table 9 – Preprocessor details: test case eb3

Id	V	Application Element	Value	Req
@101	*	elementary_B_rep	#ebsr	M
@100	*	B-rep	#torus_solid	M
@73	*	B-rep.outer	#shell_object	S
@25	aim84	edge	#oeb1f	S
@24.1	aim74	edge	#edgeb1	C1
@24.2	aim135	edge	#edget1	C1
@107.face_1		face	#face_1	C1
@31.1		face_bound	#btop	C1
@31.2		face_bound	#bbase	C1
@19	*	line	#l1	M
@35.1		loop	#loopb	C1
@68.1		loop.edges	[#oeb4t, #oeb3f, #oeb1f, #oeb2t]	C1
@35.2		loop	#loopt	C1
@114	*	polyline	#poly	M
@9	*	shell	#shell_object	M
@107	*	toroidal_surface	#torus	M
@112.1	*	toroidal_surface.major_radius	100	C1
@112.2	*	toroidal_surface.minor_radius	20	C1
@110	*	toroidal_surface.centre	(0,0,0)	C1
@111	*	toroidal_surface.axis	(0,0,1)	C1

The test requires the creation of a **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.

6.2.3.2 Postprocessor

AIM test purpose coverage:

aim54, aim74, aim84, aim94, aim98, aim99

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

NOTE 2 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 part_204_brep_product_schema;

  REALIZATION

  LOCAL
    shell_object : closed_shell ;
    torus_solid : manifold_solid_brep ;
    ebsr : elementary_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

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

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

      ebsr := elementary_brep_shape_representation
                  ('ebsr', [torus_solid], its_context );
     shape1_def_rep := shape_definition_representation
                        (shape_1_def, ebsr );
  END_REALIZATION;
```

```
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim54: toroidal_surface face shall be processed and bounded correctly.

aim74: line shall be correctly trimmed by vertices.

aim84: When edge is re-used with FALSE orientation it shall be correctly interpreted.

aim94: All **polyline** points shall lie on **toroidal_surface** with a tolerance of less than 0.0000001. Polyline points shall be coplanar.

6.2.4 Test case eb4

Test case summary:

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 which 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. (see C.3.4)

Other test purpose coverage:

other3: Polyline shall be the only permissible subtype of bounded curve used in defining an elementary B-rep.

6.2.4.1 Preprocessor

Input specification:

See table 10.

NOTE 1 See Test Case ps2 for details of product and part.

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Specific verdict criteria:

aim9: Model shall contain an example of **elementary_surface** with position as **axis2_placement_3d** with **ref_direction** absent.

Table 10 – Preprocessor details: test case eb4

Id	V	Application Element	Value	Req
@101	*	elementary_B_rep	#ebsr	M
@73	*	B-rep	#cylxcyl_solid	M
@104.1	*	cylindrical_surface	#cyl_1	M
@21.1	*	cylindrical_surface.axis	#pos_z	C1
@22.1	*	cylindrical_surface.radius	50.0	C1
@104.2		cylindrical_surface	#cyl_2	C1
@21.2		cylindrical_surface.axis	#pos_y	C1
@22.2		cylindrical_surface.radius	20.0	C1
@24	*	edge_curve	#edgemo	M
@107.1	*	face	#cyl_1f	C1
@64		face.bounds	[#bcyltop, #bcy1m, #bcy1b]	C1
@65	*	face.	#cyl_1	C1
@107.2		face	#cyl_2f	C1
@34	aim9	location	#at	M
@17	*	curve as bounded_curve	#poly	M
@114	*	polyline	#poly	M
@9	*	shell	#shell_object	C1

aim11: Model shall contain an example of **elementary_surface** with position as **axis2_placement_3d** with **axis** absent.

aim12: Model shall contain an example of **edge_curve** with **edge_geometry** as (closed 3D) polyline.

Extra details:

Create a **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.

6.2.4.2 Postprocessor

AIM test purpose coverage:

aim9, aim11, aim12

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context 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-rep.

```
*)
TEST_CASE example_ebrep_4; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    shell_object : closed_shell ;
    cylxcyl_solid : manifold_solid_brep ;
    ebsr : elementary_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : geometric_representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

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

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

      ebsr := elementary_brep_shape_representation
                 ('ebsr',  [cylxcyl_solid], its_context );
     shape1_def_rep := shape_definition_representation
```

```
                          (shape_1_def, ebsr );
  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim9: Default value of **ref_direction** attribute shall be correctly supplied.

aim11: Default value of **axis** attribute shall be correctly supplied.

aim12: All polyline points shall lie on BOTH **cylindrical_surface**s with a tolerance of less than 0.000001.

6.2.5 Test case eb5

Test case summary:

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_shell** context using the original parameters. (see C.3.2)

6.2.5.1 Preprocessor

Input specification:

NOTE 1 See Test Case ps2 for details of product and part.

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The **items** attribute of the **elementary_brep_shape_representation** entity shall consist of the identifiers of the two B-rep cones.

Specific verdict criteria:

ae52: Model shall contain examples of unbounded_curve used as edge_geometry (hyperbola and parabola).

ae70: Model shall contain an example of edge used to define two loops in a B-rep.

Table 11 – Preprocessor details: test case eb5

Id	V	Application Element	Value	Req
@101	*	elementary_b_rep	#ebsr	M
@100.1	*	B-rep	#cone1	C2
@100.2	*	B-rep	#cone2	C2
@16	*	conical_surface	#cone	M
@25.1	*	edge	#edge2	M
@61.1	*	edge.start_vertex	#ppb1	C1
@61.2	*	edge.end_vertex	#ppb2	C1
@25.2	*	edge	#edge3	C1
@52	*	edge.edge_geometry	#hyp	M
@70	*	edge to loop	#edge3 to #loophyp, #loopcone	C1
@25.2	aim50	edge	#edgeb4	C1
@13	*	ellipse	#elli	M
@27		ellipse.location	#ae	C1
@28		ellipse.major_radius	20*sqrt(3)/sqrt(2)	C1
@29		ellipse.minor_radius	10*sqrt(2)	C1
@107.1		face	#bconef	S
@64.1	*	face.bounds	[#bcone, #be_f]	C1
@103.1	*	face.face_geometry	@16	C1
@107.2		face	#tconef	C1
@64.2	*	face.bounds	[#be_t, #vbound]	M
@103.2	*	face_geometry	@16	C1
@64.3	*	face.bound	#vbound	M
@36	*	loop	#apexloop	M

aim106: Model shall contain an example of **elementary_brep_shape_representation** with context as **geometric_representation_context** with **items** as a SET of more than one **manifold_solid_brep**.

aim77: Model shall contain an example of **face** as **face_surface** with bounds as a SET of at least two **face_bound**s including one **vertex_loop**.

Extra details:

This test requires the creation of 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.

Table 11 – concluded

Id	V	Application Element	Value	Req
@14	*	hyperbola	#hyp	M
@75	*	hyperbola.location	#ah	C1
@34.1	*	location	#ah	C1
@34.2	*	location	#ap	S
@35	*	edge_loop	#looppar	M
@68	*	edge_loop.edges	[#ob1t, #oe2f]	C1
@15	*	parabola	#parab	M
@37	*	location	#ap	C1
@102		plane	#ple	C1
@9.1	*	shell	#shell1	C1
@9.2	*	shell	#shell2	C1
@53	*	vertex	#vertorc	C1

6.2.5.2 Postprocessor

AIM test purpose coverage:

aim30, aim31, aim32, aim34, aim35, aim50, aim56, aim72, aim73, aim77, aim86, aim106, aim269, aim270, aim282, aim284, aim287, aim289, aim291, aim292.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

NOTE 2 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 part_204_brep_product_schema;

  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 ;
    len_exp   : dimensional_exponents :=
                    dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
```

```
    ang_exp   : dimensional_exponents :=
                   dimensional_exponent(0, 0, 0, 0, 0, 0, 0) ;
      its_units := named_unit(len_exp) || length_unit() ||
                              si_unit (’milli’, ’metre’) ;


    its_context : geometric_representation_context ;
    prod1_name : STRING := ’prod1_name’;
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

   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 := named_unit(ang_exp) || plane_angle_unit() ||
                              si_unit (?, ’radian’) ;
      ang_m_wu := plane_angle_measure_with_unit(0.017453293 ,
                                                   angle_c_u ) ;
      angle_u := plane_angle_unit() ||
                     conversion_based_unit(’degree’, ang_m_wu ) ;
      len_u  := named_unit(len_exp) || length_unit() ||
                              si_unit (’milli’, ’metre’) ;

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

      ebsr := elementary_brep_shape_representation
                  (’ebsr’, [cone1, cone2], its_context );
     shape1_def_rep := shape_definition_representation
                        (shape_1_def, ebsr );
  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

ae70: Edge shall be correctly oriented when used to define two loops in a B-rep.

aim30: All parabolic edge points shall lie exactly on BOTH **conical_surface** and on intersecting plane, parabola shall be properly trimmed by **vertex_point**s.

aim31: All hyperbolic edge points shall lie exactly on BOTH **conical_surface** and on intersecting plane, hyperbola shall be properly trimmed by vertex_points.

aim32: All points on the ellipse shall lie exactly on BOTH **conical_surface**s and on intersecting plane.

aim50: Edge_curve with same_sense FALSE shall be correctly incorporated in edge_loop.

aim56: **conical_surface**s shall be correctly trimmed by bounding edges.

aim77: **face** with **face_bound** as **vertex_loop** shall be correctly processed.

aim106: Two B-reps should exactly fit together with a common face surface, two distinct B-reps shall not intersect.

6.2.6 Test case eb6

Test case summary:

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 (see C.3.4).

Other test purpose coverage:

other9: **elementary_brep_shape_representation** with context as **geometric_context** with **items** as **mapped_item**;

other10: **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**.

6.2.6.1 Preprocessor

Input specification:

See table 12.

NOTE 1 See Test Case ps1 for details of product and part.

NOTE 2 See Test Case eb4 for details of shell_object.

Table 12 – Preprocessor details: test case eb6

Id	V	Application Element	Value	Req
@101.1		elementary_B_rep	#ebsr	
@101.2	other9	elementary_B_rep	#ebsr1	M
@100.1	*	elementary_B_rep	#ebass	M
@100.2 @73		B-rep brep.outer	#cylxcyl #shell_object	C2 C1
@100.3	other9	B-rep	#transrot1	M
@100.4	other10	mapped B-rep	#trans2	M
@34.1		location	#baseaxes	C2
@34.2 @34.3		location location	#refaxes #topaxes	C1 C1

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The **items** attribute of the **elementary_brep_shape_representation** entity shall consist of the identifiers of the B-rep, the transformation and the reference axes.

Specific verdict criteria:

other9: Model shall contain an example of an **elementary_brep_shape_representation** with context as **geometric_context** with **items** as **mapped_item**;

other10: Model shall contain an example of an **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**.

Extra details:

This test requires the creation of 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.

6.2.6.2 Postprocessor

AIM test purpose coverage:

aim214, aim215, aim262, aim263, aim274, aim275.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used twice to define the product contexts 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 part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    prod2_name : STRING := 'prod2_name';
    shape_1_def, shape_2_def : product_definition_shape ;
    shape1_def_rep, shape2_def_rep : shape_definition_representation ;
    origin : cartesian_point ;
    pos_z, neg_y  : direction ;
    refaxes, topaxes, baseaxes : axis2_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 ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
      its_units := named_unit ;
  END_LOCAL;

    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;
```

```
    CALL basic_product_structure ;  -- parameters for second product.
      IMPORT (shape_2_def := @prod_def_shape; );
      WITH   (prod_name := @prod2_name;
              pdef_desc := 'test product definition 2';
              propd_desc := 'shape of test product 2';
              prod_name :=  'second test product';
              prod_id   := 'P02' ;
              pdf_id    := 'PDF02' ; );
    END_CALL;

    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) ;

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

grc1 := geometric_representation_context ('grc1',
                            'context for cylinder union', 3) ||
                    global_unit_assigned_context ( [its_units] ) ;

grc2 := geometric_representation_context ('grc2',
                     'context for rotated cylinder union', 3) ||
                    global_unit_assigned_context ( [its_units] ) ;

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 which 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 ) ;
shape1_def_rep := shape_definition_representation
                        (shape_1_def, ebsr1 );
shape2_def_rep := shape_definition_representation
                        (shape_2_def, ebsrass );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

other9: After processing the mapped_item shall be correctly interpreted, result is a rotated and translated copy of original B-rep.

other10: 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.

other9 and other10: The two distinct **elementary_brep_shape_representation**s shall not be spatially related.

6.3 Abstract test cases for advanced B-rep

The set of test cases given here covers the required scope of the test purposes defined for the advanced B-rep UoF.

The post-processor abstract test cases in this clause are fully documented in EXPRESS-I .
A simple textual description and table of instances 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 1 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 ab1.

6.3.1 Test case ab1

Test case summary:

Test case ab1 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. (See C.4.1.)

Other test purpose coverage:

other1: any instance of shape_representation in the model shall be either an **advanced_brep_shape_-representation**, an **elementary_brep_shape_representation**, or a **faceted_brep_shape_representation**; no other subtypes shall occur. (see 6.3.1, 6.2.1, 6.1.1)

other5: **advanced_brep_shape_representation** with context as **geometric_context** with **items** as **manifold_solid_brep**.

6.3.1.1 Preprocessor

Input specification:

See table 13

NOTE 1 See Test Case ps1 for details of product and part.

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The sole **item** of the **advanced_brep_shape_representation** must have the same identifier as the manifold solid B-rep.

Specific verdict criteria:

ae1: Any instance of shape_representation in the model shall be an **advanced_brep_shape_representation**; no other subtypes shall occur.

ae33: Each Geometric_element in the model shall be used as part of the geometric description of a B-rep.

aim37: faces of B-rep shall be a SET of more than one face_surface;

aim45: All edges shall be bounded by vertex_points, all edges shall be of type edge_curve.

aim53: face_geometry attribute for each face shall be a subtype of elementary_surface.

aim61: Model shall contain example of face with more than one face_bound in bounds SET; each face_bound shall reference an edge_loop.

aim67: Some face shall have a single bound of type face_outer_bound; orientation attribute shall be TRUE for some face_bound.

aim68: Each face surface shall be defined so that orientation is TRUE.

Table 13 – Preprocessor details: test case ab1

Id	V	Application Element	Value	Req
@73	*	advanced_B_rep	#absr	M
@3	*	B-rep	#cysp_solid	C2
@74	*	B-rep to closed_shell	@3 to @9	
@6	*	circle	#circ	M
@8	*	circle.radius	25.0	C1
@7	*	circle.location	#a1	C1
@9	*	closed_shell	#shell_object	M
@11	aim37	closed_shell to faces	@9 to [#face_1, #face_2, #face_3]	C1
@12	*	conic as circle	#circ	M
@13	*	conic as ellipse	#elli	M
@26	*	curve	#elli	M
	*	curve to edge	@13 to @24	C1
@20	*	cylindrical_surface	#cyl	M
@21	*	cylindrical_surface.axis	#a1	C1
@22	*	cylindrical_surface.radius	25.0	C1
@23	*	direction	#dslope	C1
@24	*	edge as edge_curve	#edge2	M
@69	*	edge to B-rep	@24 to @3	
@24	*	edge to curve	@24 to @13	
@25	*	edge to vertex	@24 to @verte, @verte	
@104	*	elementary_surface (as cylindrical_surface)	#cyl	M
@26	*	curve as ellipse	#elli	M
@28	*	ellipse.major_radius	25*rt(2)	C1
@29	*	ellipse.minor_radius	25.0	C1
@27	*	ellipse.location	#a2	S
@30	*	face as advanced_face	#face_2	C1
@64		face to loops	@30 to #bcylbot, #bcyltop	C1
@62	*	face to face	#face_1 to #face_2	C1
@65	*	face to surface	@30 to @104	C1
@31	*	face	#face_1 (= #top_face)	C1
@63	*	face to loop	@31 to #loope	C1
@404	*	global_unit	#its_units	M

Table 13 – concluded

Id	V	Application Element	Value	Req
@34	*	location	#a2	M
@35	*	loop	#loopc	M
@67	*	loop to edge	#loopc to #edge1	C1
@27	*	plane	#p1	M
@38.1	*	point	#ctop	
@38.2	*	point	#epoint	
@105	*	spherical_surface	#sphere	M
@41	*	spherical_suface.centre	#origin	C1
@42	*	spherical_surface.radius	25.0	C1
@53	*	vertex as vertex_point	#verte	M
@71	*	vertex to point	@53 to @38.1 #epoint	

aim70: face_geometry attribute of face shall be subtype of surface.

aim76: All loops shall be of type edge_loop.

aim78: manifold_solid_brep instance shall be simple supertype, outer attribute shall be of type closed_shell, oriented_closed_shell subtype not to be used.

aim107: representation as advanced_brep_shape_representation shall contain a single representation item. model created shall contain no polyloops or vertex_loops.

Extra details:

This test requires the creation of an **advanced_brep_shape_representation** 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 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, all shall be of type **advanced_face**. A suitable set of dimensions is defined in the EXPRESS-I specification below.

6.3.1.2 Postprocessor

AIM test purpose coverage:

aim8, aim10, aim14, aim25, aim26, aim27, aim28, aim32, aim33, aim36, aim37, aim39, aim41, aim44, aim45, aim46, aim47, aim48, aim49, aim53, aim55, aim57, aim58, aim59, aim60, aim61, aim62, aim63, aim64, aim65, aim66, aim67, aim68, aim70, aim71, aim76, aim78, aim83, aim85, aim88, aim89, aim90, aim91, aim92, aim95, aim96, aim97, aim103, aim105, aim107, aim260, aim261, aim268, aim273, aim275, aim276, aim277, aim286, aim290, aim294, aim295.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

NOTE 2 The **cylinder_sphere_faces** is used, in its simplest form with default values, to define the faces of the B-rep.

```
*)
TEST_CASE example_abrep_1; WITH part_204_brep_product_schema;


  REALIZATION

  LOCAL
    face_1, face_2, face_3 : advanced_face ;
    shell_object : closed_shell ;
    cysp_solid : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; ) ;
      WITH   (prod_name := @prod1_name; );
   END_CALL;

  CALL cylinder_sphere_faces ; -- uses default values, so no WITH
      IMPORT (face_1 := @top_face ;
              face_2 := @curved_face ;
              face_3 := @bottom_face ; ) ;
  END_CALL;
      shell_object := closed_shell('shell_object', [face_1, face_2,
                                                    face_3] );
      cysp_solid := manifold_solid_brep ('cysp_solid', shell_object);

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

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

```
      absr := advanced_brep_shape_representation
                 ('absr',  [cysp_solid], its_context );
       shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr );

 END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim9: axis2_placement with ref_direction present shall be correctly interpreted to locate surface.

aim10: axis2_placement with axis present shall be correctly interpreted to locate surface.

aim32: Edge with identical start and end vertices shall be correctly interpreted as closed ellipse.

aim33: Edge with identical start and end vertices shall be correctly interpreted as closed circle.

aim48: faces shall be connected along edges.

aim51: All edge_curves shall lie on face_surface, all vertex_points shall lie on edge_curve. conic as circle.

aim55: Correct portion of spherical_surface shall be defined by edge_loops.

aim57: Unbounded cylindrical_surface shall be bounded by edge_loops.

aim58: Bounding loops for planar face shall be co-planar.

aim61: Multiple bounds shall be correctly interpreted to trim face.

aim64: face_bound with orientation FALSE shall be correctly interpreted to define correct portion of face surface.

aim70: edge_curves and vertices of bounding edge_loops shall lie on surface for each face_surface.

aim107: All WRs on advanced_brep_shape_representation shall be verified.

other5: length units shall be correctly interpreted, model created shall contain no polyloops or vertex_loops.

6.3.2 Test case ab2

Test case summary:

Test case ab2 is designed to test the definition of an advanced 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. (See C.4.1.) The result is a hollow solid with void(s) of a similar shape, or spherical.

NOTE 1 If required this test can easily be modified to test geometric precision by varying the parameters to define voids which 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.

6.3.2.1 Preprocessor

Input specification:

See table 14

NOTE 1 See Test Case ps1 for details of product and part.

NOTE 2 See Test Case ab1 for details of shell_object.

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Specific verdict criteria:

aim13: Model shall contain example of Brep_with_voids subtype with one shell in voids SET.

aim26: Model shall contain example of closed_shell with a single face_surface, this is spherical.

ae56: Model shall contain an example of B-rep consisting of more than one closed_shell, one closed_shell as outer shell.

ae73: Model shall contain an example of void with topological_normal pointing into void.

aim79: Model shall contain example of Brep_with_voids subtype with two shells in voids SET.

aim80: Shells in voids SET shall be of type oriented_closed_shell.

Extra details:

This test requires the creation of an **advanced_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 base 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**.

Table 14 – Preprocessor details: test case ab2

Id	V	Application Element	Value	Req
@74.1	*	advanced_B_rep	#absr1	C1
@74.2	*	advanced_B_rep	#absr2	C1
@56.1	aim79	B-rep as brep_with_voids	#cysp_with_void	M
@56.1	*	B-rep to shell (as outer)	@56.1 to #shell_object	C1
@56.1	*	B-rep to shell (as void)	@56.1 to @54	C1
@56.2	aim13	B-rep as brep_with_voids	#cysp_with_voids	M
@56.2	*	B-rep to shells (as voids)	@56.2 to [@54, @55]	C1
@30	*	face as advanced_face	#sp_face	M
@65		face to surface	@30 to #sph2	C1
@31	*	face to bounds	@30 to [#s_bound]	M
@38		point as cartesian_point	#top_pt = (-5, -5, 5)	C1
@72		point to vertex	#top_pt to #top_v	C1
@54.1	*	shell as closed_shell	#void1	M
@11		closed_shell to faces	#void1 to [#vface_1, #vface_2, #vface_3]	C1
@55.1	*	shell.orientation	FALSE	M
@54.2	*	shell as oriented_closed_shell	#void2 S	
@10	aim26	closed_shell to face	#void2 to [@sp_face]	S
@55.2	*	shell.orientation	FALSE	M
@105		elementary_surface as spherical_surface	#sph2	S
@42		spherical_surface.radius	10	S
@41		spherical_surface.centre	(-5,-5,-5)	S
@53		vertex as vertex_point	#top_v	S
@36	aim77	loop as vertex_loop	#v_loop	M
@36	*	loop to vertex	#v_loop to #top_v	C1

Each face shall be of type **advanced_face**. The centre of the hemisphere 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.

6.3.2.2 Postprocessor

AIM test purpose coverage:

aim13, aim26, aim56, aim73, aim79, aim81, aim82, aim96, aim104, aim267.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

NOTE 2 The **cylinder_sphere_faces** context is used, with appropriate parameters, to define the faces of the B-rep outer shells and void shells.

NOTE 3 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_abrep_2; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    prod2_name : STRING := 'prod2_name';
    shape_1_def, shape_2_def : product_definition_shape ;
    shape1_def_rep, shape2_def_rep : shape_definition_representation ;
    face_1, face_2, face_3 : advanced_face ;
    vface_1, vface_2, vface_3 : advanced_face ;
    shell_object, hollow1, hollow2 : closed_shell ;
    void1, void2 : oriented_closed_shell;
    cylsp_with_void : brep_with_voids ;
    cylsp_with_voids : brep_with_voids ;
    absr1, absr2 : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    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 : advanced_face ;
    sp_shell : closed_shell ;
  END_LOCAL;

    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; ) ;
      WITH   (prod_name := @prod1_name; ) ;
    END_CALL;

    CALL cylinder_sphere_faces ; -- uses default values, so no WITH
      IMPORT (face_1 := @top_face ;
              face_2 := @curved_face ;
              face_3 := @bottom_face ; );
```

```
    END_CALL;

    CALL cylinder_sphere_faces;
(*  parameters re-set for dimensions -large void  *)
      IMPORT (vface_1 := @top_face ;
              vface_2 := @curved_face ;
              vface_3 := @bottom_face ; );
      WITH (orc := 10; rad := 12; ht := 50; );
    END_CALL;

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

    CALL basic_product_structure ;  -- parameters for second product.
      IMPORT (shape_2_def := @prod_def_shape; ) ;
      WITH   (prod_name := @prod2_name;
              pdef_desc := 'test product definition 2';
              propd_desc := 'shape of test product 2';
              prod_name :=  'second test product';
              prod_id   := 'P02' ;
              pdf_id    := 'PDF02' ; ) ;
    END_CALL;

    shell_object := closed_shell ('shell_object', [face_1, face_2,
                                                    face_3]) ;
    hollow_1 :=closed_shell ('hollow_1',  [vface_1, vface_2, vface_3]);
    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 := advanced_face ('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 := brep_with_voids ('cylsp_w_v', shell_object,
                                                      [void1]) ;

    cylsp_with_voids := brep_with_voids ('cylsp_with_voids',
                                    shell_object, [void1, void2]);
```

```
    its_units := named_unit(len_exp) || length_unit() ||
                              si_unit (’milli’, ’metre’) ;

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

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

    absr1 := advanced_brep_shape_representation
             ( ’absr1’, [cylsp_with_void], context1 ) ;

    absr2 := advanced_brep_shape_representation
             ( ’absr2’, [cylsp_with_voids], context2 ) ;
  shape1_def_rep := shape_definition_representation
                       (shape_1_def, absr1 );
  shape2_def_rep := shape_definition_representation
                       (shape_2_def, absr2 );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim13: Void shell shall not intersect outer shell, void shell shall be completely within outer shell.

aim267: Void shells shall not intersect outer shell or each other, each void shell shall be completely within outer shell, two advanced_brep_shape_representations absr1 and absr2 shall not be spatially related.

aim81: Void shell normal shall point into void after correctly interpreting orientation, void shell shall be completely within outer shell.

aim96: Closed_shell with single face with spherical_surface geometry shall be correctly processed.

aim104: Vertex_loop shall be correctly processed as face_bound.

6.3.3 Test case ab3

Test case summary:

Test case ab3 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_advanced** context using the original parameters. (See C.4.3.)

Other test purpose coverage:

other2: Polyline and b_spline curve shall be the only permissible subtypes of bounded curve used in defining an advanced B-rep.

Input specification:

See table 15

NOTE 1 See Test Case ps1 for details of product and part.

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Specific verdict criteria:

aim12: Model shall contain an example of **edge_curve** with **edge_geometry** as **polyline**. Polyline points shall lie on toroidal surface with a tolerance of less than 0.0000001.

aim40: Model shall contain an example of **edge_curve** with **edge_geometry** as **line**. Polyline shall not be used to define geometry of straight edge.

Extra details:

This test requires the creation of an **advanced_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, all of type **advanced_face**.. A suitable set of dimensions is defined in the EXPRESS-I specification below.

6.3.3.1 Postprocessor

AIM test purpose coverage:

aim12, aim40, aim54, aim84, aim98, aim99.

Input specification:

Table 15 – Preprocessor details: test case ab3

Id	V	Application Element	Value	Req
@2	*	advanced_b_rep	#absr	M
@3	*	B-rep	#torus_solid	M
@73		B-rep to shell (outer)	#shell_object	S
@25	aim84	edge	#oeb1f	S
@24.1	aim40	edge as edge_curve	#edgeb1	
@24.1	*	edge.edge_geometry	@19	M
@24.2		edge as edge_curve	#edget1	C1
@24.2	aim94	edge.edge_geometry	@114	M
@30		face as advanced_face	#face_1	C1
@31		face (with one outer bound)	#face_1 to #btop	C1
@65		face.face_geometry	#torus	C1
@63		face to loop	#face_1 to #loopt	C1
@19	*	curve as line	#l1	M
@35.1		loop as edge_loop	#loopb	C1
@68.1		loop to edges	#loopb to [#oeb4t, #oeb3f,#oeb1f, #oeb2t]	C1
@35.2		loop as edge_loop	#loopt	C1
@68.2		edge_loop.edges	[@oeb2f, @oet1t, @oeb3t, @oet2f]	C1
@114	*	curve as polyline	#poly	M
@9	*	shell as closed_shell	#shell_object	M
@10		closed_shell.faces	[#face_1, #face_2, #face_3, #face_4]	C1
@107	*	toroidal_surface	#torus	M
@112.1	*	toroidal_surface.major_radius	100	C1
@112.2	*	toroidal_surface.minor_radius	20	C1
@110	*	toroidal_surface.centre	(0,0,0)	C1
@111	*	toroidal_surface.axis	(0,0,1)	C1

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

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

```
*)
TEST_CASE example_abrep_3; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    face_1, face_2, face_3, face_4 : advanced_face ;
    shell_object : closed_shell ;
    torus_solid : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : geometric_representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; ) ;
      WITH   (prod_name := @prod1_name; ) ;
   END_CALL;

  CALL toroidal_segment_advanced ; -- uses default values, so no WITH
      IMPORT (face_1 :=  @curved_face ;
              face_2 := @base_face ;
              face_3 := @left_face ;
              face_4 := @right_face ;) ;
    END_CALL;
      shell_object := closed_shell('shell_object',
                           [face_1, face_2, face_3, face_4] ) ;
      torus_solid := manifold_solid_brep ('torus_solid', shell_object);

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

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

```
      absr := advanced_brep_shape_representation
                 ('absr', [torus_solid], its_context );
     shape1_def_rep := shape_definition_representation
                          (shape_1_def, absr );
END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim12: All polyline points shall lie on toroidal_surface with a tolerance of less than 0.0000001.

aim40: Edge geometry as line shall be correctly trimmed by vertices.

aim54: Toroidal_surface face shall be processed and bounded correctly.

aim84: When edge is re-used with FALSE orientation it shall be correctly interpreted.

6.3.4 Test case ab4

Test case summary:

Test case ab4 is a test case consisting of the faces needed to define a single solid resulting from the union of two cylinders of different radii which 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_advanced** context using the original parameters. (See C.4.4.)

Other test purpose coverage:

other2: Polyline and b_spline curve shall be the only permissible subtypes of bounded curve used in defining an advanced B-rep.

6.3.4.1 Preprocessor

See table 16

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Specific verdict criteria:

aim12: Model shall contain an example of **edge_curve** with **edge_geometry** as (closed 3D) **polyline** .

Table 16 – Preprocessor details: test case ab4

Id	V	Application Element	Value	Req
@2	*	advanced_B_rep	#absr	M
@73	*	B-rep	#cylxcyl_solid	M
@73		b-rep.outer	#shell_object	C1
@104.1	*	cylindrical_surface	#cyl_1	M
@22.1		cylindrical_surface.radius	50.0	C1
@21.1		cylindrical_surface.axis	(0,0,1)	C1
@104.2		cylindrical_surface	#cyl_2	C1
@22.2		cylindrical_surface.radius	20.0	C1
@21.2		cylindrical_surface.axis	(0,1,0)	C1
@24	*	edge	#edgemo	M
@24	*	edge.edge_geometry	@17	C1
@25.1		edge.start_vertex	#v1	C1
@25.2		edge.end_vertex	#v1	C1
@30.1	*	face	#cyl_1f	C1
@64		face to loops	30.1 to [#bcyltop, #bcy1m, #bcy1b]	C1
@65	*	face.face_geometry	@cyl_1	C1
@30.2	*	face	#cyl_2f	C1
@34	aim9, aim11	location as axis2_placement_3d	#at	M
@17	*	curve as bounded_curve	#poly	M
@114	*	bounded_curve as polyline	#poly	M
@9	*	shell	#shell_object	C1
@11		closed_shell.faces	[#cyl_1f, #cyl2_2f, #top, #bottom, #endface]	C1

aim61: Model shall contain an example of **polyline** with **points** as a LIST of 3 or more **cartesian_point**s.

Extra details:

This test requires the creation of an **advanced_brep_shape_representation** consisting of a single **manifold_solid_brep**. The **manifold_solid_brep** should be in the form of two perpendicular intersecting cylinders forming a T shape. 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. All faces shall be of type **advanced_face**. A suitable set of dimensions is defined in the EXPRESS-I specification below.

NOTE 1 See Test Case ps1 for details of product and part.

6.3.4.2 Postprocessor

AIM test purpose coverage:

aim9, aim11, aim12, aim38, aim94.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

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

```
*)
TEST_CASE example_abrep_4; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    cyl_1f, cyl_2f, top, bottom, end_face: advanced_face ;
    shell_object : closed_shell ;
    cylxcyl_solid : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

    CALL cylinder_union_polyline_advanced ;
    (*  uses default values, so no WITH    *)
      IMPORT (cyl_1f := @cyl_face1 ;
              cyl_2f := @cyl_face2 ;
              top   := @top_face ;
              bottom := @base_face ;
              end_face := @right_face ; ) ;
    END_CALL;
      shell_object := closed_shell('shell_object',
```

```
                        [cyl_1f, cyl_2f, top, bottom, end_face] ) ;
      cylxcyl_solid := manifold_solid_brep ('cylxcyl_solid',
                                                  shell_object) ;

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

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

      absr := advanced_brep_shape_representation
                 ('absr', [cylxcyl_solid], its_context );
     shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr );
  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim9: Default value of ref_direction attribute shall be correctly supplied to define axis2_placement_-3d.

aim11: Default value of axis attribute shall be correctly supplied. (See build_axes function for definition)

aim94: All polyline points shall lie on BOTH cylindrical_surfaces with a tolerance of less than 0.000001.

6.3.5 Test case ab5

Test case summary:

Test case ab5 is a test case consisting of the faces needed to define a single solid resulting from the union of two cylinders of different radii which have orthogonal axes. The intersection curve is a closed 3D curve represented by a B-spline curve of degree 11. The definition of the shell is provided by the **cylinder_union_b_spline** context using the original parameters. (See C.4.5.)

Other test purpose coverage:

other2: Polyline and b_spline curve shall be the only permissible subtypes of bounded curve used in defining an advanced B-rep.

6.3.5.1 Preprocessor

Input specification:

See table 17

NOTE 1 See Test Case ps1 for details of product and part.

Table 17 – Preprocessor details: test case ab5

Id	V	Application Element	Value	Req
@2	*	advanced_B_rep	#absr	M
@73	*	B-rep	#cylxcyl_solid	M
@73		B-rep.outer	#shell_object	C1
@104.1	*	cylindrical_surface	#cyl_1	M
@22.1		cylindrical_surface.radius	50.0	C1
@21.1		cylindrical_surface.axis	(0,0,1)	C1
@104.2		cylindrical_surface	#cyl_2	C1
@22.2		cylindrical_surface.radius	20.0	C1
@21.2		cylindrical_surface.axis	(0,1,0)	C1
@24	*	edge	#edgemo	M
@24	*	edge.edge_geometry	@17	C1
@25.1		edge.start_vertex	#v1	C1
@25.2		edge.end_vertex	#v1	C1
@30.1	*	face	#cyl_1f	C1
@64		face to loops	30.1 to [#bcyltop, #bcy1m, #bcy1b]	C1
@65	*	face.face_geometry	@cyl_1	C1
@30.2	*	face	#cyl_2f	C1
@34	aim9, aim11	location as axis2_placement_3d	#at	M
@5	*	bounded_curve as twisted_curve	#spline	M
@51	*	twisted_curve as b_spline_curve	#spline	M
@9	*	shell as closed_shell	#shell_object	C1
@11		shell.faces	[#cyl_1f, #cyl2_2f, #top, #bottom, #endface]	C1

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Specific verdict criteria:

aim118: Model shall contain an example of **b_spline_curve** with **curve_form** as 'unspecified'.

Extra details:

Create an **advanced_brep_shape_representation** consisting of a single **manifold_solid_brep**. The **manifold_solid_brep** should be in the form of two perpendicular intersecting cylinders with intersecting axes. Defaults shall be used in defining the **axis2_placement_3d** to locate one of these cylinders. The intersection curve shall be represented by a closed B-spline curve of degree 11 and of Bezier type. This curve shall lie on both cylinders with a tolerance of less than 0.001. All faces shall be of type **advanced_face**. A suitable set of dimensions is defined in the EXPRESS-I specification below.

6.3.5.2 Postprocessor

AIM test purpose coverage:

aim113, aim115, aim118, aim119, aim121, aim149.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

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

```
*)
TEST_CASE example_abrep_5; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    cyl_1, cyl_2, top, bottom, end_face: advanced_face ;
    shell_object : closed_shell ;
    cylxcyl_solid : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : geometric_representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;
```

```
   CALL cylinder_union_b_spline ; -- uses default values, so no WITH
      IMPORT (cyl_1 := @cyl_face1 ;
              cyl_2 := @cyl_face2 ;
              top   := @top_face ;
              bottom := @base_face ;
              end_face:= @right_face ; ) ;
    END_CALL;
      shell_object := closed_shell('shell_object',
                              [cyl_1, cyl_2, top, bottom, end_face] ) ;
      cylxcyl_solid := manifold_solid_brep ('cylxcyl_solid',
                                                 shell_object) ;

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

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

      absr := advanced_brep_shape_representation
                  ('absr', [cylxcyl_solid], its_context );
     shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr );
  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim113: b_spline_curve shall not be self-intersecting.

aim115: First and last points of b_spline_curve shall be coincident.

aim149: b_spline_curve shall lie on BOTH cylindrical_surfaces with a tolerance of less than 0.001; a two segment Bezier curve of degree 11 shall be correctly interpreted.

6.3.6 Test case ab6

Test case summary:

Test case 6 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. (See C.4.2.)

6.3.6.1 Preprocessor

Input specification:

See table 18

NOTE 1 See Test Case ps1 for details of product and part.

Table 18 – Preprocessor details: test case ab6

Id	V	Application Element	Value	Req
@2	aim79	advanced_b_rep	#absr	M
@3.1	*	B-rep	#cone1	M
@73.1		B-rep.outer	#shell1	C2
@3.2	*	B-rep	#cone2	M
@73.2		B-rep.outer	#shell2	C2
@16	*	conical_surface	#cone	M
@25.1	*	edge	#edge2	M
@24.1	*	edge.edge_geometry	#parab	M
@61	*	edge to edge	#edge2 to #edgeb4 by #vertpb2	C1
@24.2	*	edge as edge_curve	#edge3	S
@52	*	unbounded_curve as edge_geometry	#hyp	M
@24.3	aim50	edge as edge_curve	#edgeb4	C1
@13	*	conic as ellipse	#elli	M
@27	*	location	#ae	C1
@28		major_radius	20*sqrt(3)/sqrt(2)	C1
@29		minor_radius	10*sqrt(2)	C1
@30.1		face as advanced_face	#bconef	S
@32.1	*	face.bounds	[#bcone, #be_f]	C1
@65.1	*	face.face_geometry	@16.cone	C1
@30.2	*	face as advanced_face	#tconef	C1
@32.2	*	face.bounds	[#be_t, #vbound]	M
@65.2	*	face.face_geometry	@16	C1
@64	*	face.bound	#vbound	M
@36	*	loop	#apexloop	M

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Table 18 – concluded

Id	V	Application Element	Value	Req
@57	*	face to B-reps	#tope_face	M
@14	*	hyperbola	#hyp	M
@75	*	hyperbola.location	#ah	C1
@34.1	*	location	#ah	C1
@34.2	*	location	#ap	S
@35	*	edge_loop	#looppar	M
@68	*	edge_loop.edges	[#ob1t, #oe2f]	C1
@15	*	parabola	#parab	M
@37	*	location	#ap	C1
@102		plane	#ple	C1
@9.1	*	shell	#shell1	C1
@9.2	*	shell	#shell2	C1
@53	*	vertex	#vertorc	C1

C2: The items attribute of the advanced_brep_shape_representation entity must consist of the identifiers of the two B-reps.

Specific verdict criteria:

ae52: Model shall contain examples of unbounded_curve used as edge_geometry (hyperbola and parabola).

ae70: Model shall contain an example of edge used to define two loops in a B-rep.

ae57: Model shall contain an example of Face used to define more than one closed shell in different B-reps. (Note: faces #tope_face and #bottome_face in EXPRESS-I model are identical apart from orientation.)

aim77: Model shall contain an example of **face** as **face_surface** with **bounds** as SET of at least two **face_bound**s (including one **vertex_loop**); no additional edges shall be defined for curved faces, (no prop edges);

aim107: Model shall contain an example of **advanced_brep_shape_representation** with context as **geometric_representation_context** with **items** as a SET of more than one **manifold_solid_brep**.

Extra details:

Create an **advanced_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. All faces are of type **advanced_face**. A suitable set of dimensions is defined in the EXPRESS-I specification below.

6.3.6.2 Postprocessor

AIM test purpose coverage:

aim30, aim31, aim32, aim34, aim35, aim50, aim56, aim72, aim73, aim77, aim86, aim107, aim269, aim270, aim282, aim284, aim287, aim289, aim291, aim292.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

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

```
*)
TEST_CASE example_abrep_6; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    tcone, parab, hyperb, top, bottom, cone, base : advanced_face ;
    shell1, shell2 : closed_shell ;
    cone1, cone2 : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    angle_u, len_u, angle_c_u : named_unit ;
    ang_m_wu : plane_angle_measure_with_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    ang_exp   : dimensional_exponents :=
                   dimensional_exponent(0, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

   CALL cone_faces ; -- uses default values, so no WITH
```

```
      IMPORT (bconef := @curved_face ;
              parab  := @par_face ;
              hyperb := @hyp_face ;
              bottom := @bottomc_face ;
              top    := @tope_face ;
              tconef := @top_face ;
              base := @bottome_face ; ) ;
   END_CALL;
      shell1 := closed_shell('shell1', [bconef, parab, hyperb, top,
                                                          bottom ]) ;
      shell2 := closed_shell('shell2', [tconef, base] );
      cone1 := manifold_solid_brep ('cone1', shell1) ;
      cone2 := manifold_solid_brep ('cone2', shell2) ;
      angle_c_u := named_unit(ang_exp) || plane_angle_unit() ||
                              si_unit (?, 'radian') ;
      ang_m_wu := plane_angle_measure_with_unit(0.017453293 ,
                                                  angle_c_u ) ;
      angle_u := plane_angle_unit() ||
                     conversion_based_unit('degree', ang_m_wu ) ;
      len_u  := named_unit(len_exp) || length_unit() ||
                              si_unit ('milli', 'metre') ;

      its_context := geometric_representation_context
                          ('its_context', 'context_for_cones', 3) ||
                       global_unit_assigned_context ([len_u, angle_u]);
      absr := advanced_brep_shape_representation
                  ('absr', [cone1, cone2], its_context );
     shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr );
  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

ae52: Unbounded_curve used as edge_geometry shall be correctly trimmed by vertices.

ae61: Vertex shall be located at precise intersection of edges, edges shall not intersect at non-vertex points.

ae70: Edge shall be correctly oriented when used to define two loops in a B-rep.

aim30: All parabolic edge points shall lie exactly on BOTH **conical_surface** and on intersecting plane, parabola shall be properly trimmed by **vertex_point**s.

aim31: All hyperbolic edge points shall lie exactly on BOTH **conical_surface** and on intersecting plane, hyperbola shall be properly trimmed by vertex_points.

aim32: All points on the ellipse shall lie exactly on BOTH **conical_surface**s and on intersecting plane.

aim50: Edge_curve with same_sense FALSE shall be correctly incorporated in edge_loop.

aim56: **conical_surface**s shall be correctly trimmed by bounding edges.

aim77: **face** with **face_bound** as **vertex_loop** shall be correctly processed.

aim107: Two B-reps in representation shall touch over one face but not intersect.

6.3.7 Test case ab7

Test case summary:

Test case ab7 is designed to test the use of **mapped_item**s in the creation of a simple assembly of advanced 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_advanced** context to define the geometry and topology. (See C.4.4.)

Other test purpose coverage:

other6: **advanced_brep_shape_representation** with context as **geometric_context** with **items** as **mapped_item**;

other7: **advanced_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**.

6.3.7.1 Preprocessor

Input specification:

See table 19

NOTE 1 See EXPRESS-I specification for dimensional and other details.

NOTE 2 See Test Case ps1 for details of product and part.

NOTE 3 See Test Case ab4, table 16, for details of cylxcyl B-rep object.

Constraints on values:

Table 19 – Preprocessor details: test case ab7

Id	V	Application Element	Value	Req
@2.absr		advanced_B_rep	#absr	
@2	other6	advanced_b_rep	#absr1	M
@3	other7	B-rep as advanced_b_rep	#absrass	M
@74.1		B-rep	#cylxcyl	C1
@73		B-rep.outer	#shell_object	C1
@74.2	other6	mapped B-rep	#transrot1	M
@74.3	other7	mapped B-rep	#trans2	M
@34.1		location as axis2_placement_3d	#baseaxes	C1
@34.2		location as axis2_placement_3d	#refaxes	C1
@34.3		location as axis2_placement_3d	#topaxes	C1

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Specific verdict criteria:

other6: Model shall contain an example of **advanced_brep_shape_representation** with context as **geometric_context** with **items** as **mappeditem**. The **mapped_item** shall be defined using **axis2_placement_3d**.

other7: Model shall contain an example of **advanced_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**.

Extra details:

Construct a **manifold_solid_brep** consisting of two perpendicular intersecting cylinders as in test case ab4. The first **advanced_brep_shape_representation** consists of this original solid. The **mapped_item** construct is then used to produce a translated and rotated copy of this solid which then defines a second **advanced_brep_shape_representation** in the same **geometric_representation_context**. (The two solids should touch but not intersect). A third **advanced_brep_shape_representation**, in a different context is made from the original solid, an axis2_placement and the **mapped_item**.

6.3.7.2 Postprocessor

AIM test purpose coverage:

aim71, aim214, aim215, aim261, aim262, aim263.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used twice to define the product contexts for the geometric definitions.

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

NOTE 3 **absr1** should be a rotated and translated copy of **absr**.

NOTE 4 **absrass** should be equivalent to 2 copies of **absr** 'glued' together.

```
*)

TEST_CASE example_abrep_7; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    prod2_name : STRING := 'prod2_name';
    shape_1_def, shape_2_def : product_definition_shape ;
    shape1_def_rep, shape2_def_rep : shape_definition_representation ;
    cyl_1, cyl_2, top, bottom, end_face: advanced_face ;
    origin : cartesian_point ;
    pos_z, neg_y  : direction ;
    refaxes, topaxes, baseaxes : axis2_placement_3d;
    shell_object  : closed_shell ;
    cylxcyl : manifold_solid_brep ;
    absr, absr1, absrass : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    grc1, grc2 : representation_context ;
    transrot1, trans2  : mapped_item ;
    mapping1, mapping2   : representation_map ;
  END_LOCAL;

    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;

    CALL basic_product_structure ;  -- parameters for second product.
      IMPORT (shape_2_def := @prod_def_shape; );
      WITH   (prod_name := @prod2_name;
              pdef_desc := 'test product definition 2';
```

```
            propd_desc := 'shape of test product 2';
            prod_name :=  'second test product';
            prod_id   := 'P02' ;
            pdf_id    := 'PDF02' ; );
    END_CALL;

    CALL cylinder_union_polyline_advanced ;
    (*  uses default values, so no WITH  *)
      IMPORT (cyl_1 := @cyl_face1 ;
            cyl_2 := @cyl_face2 ;
            top   := @top_face ;
            bottom := @base_face ;
            end_face:= @right_face ;
            origin := @origin;
            baseaxes := @ ab; refaxes := @ar; topaxes := @at; );
    END_CALL;

shell_object := closed_shell('shell_object',
                             [cyl_1, cyl_2, top, bottom, end_face] );

cylxcyl := manifold_solid_brep ('cylxcyl', shell_object) ;
  its_units := named_unit(len_exp) || length_unit() ||
                              si_unit ('milli', 'metre') ;

grc1 := geometric_representation_context ('grc1',
                           'context for cylinder union', 3) ||
          global_unit_assigned_context ([its_units] ) ;
grc2 := geometric_representation_context ('grc2',
                       'context for rotated cylinder union', 3) ||
          global_unit_assigned_context ([its_units] ) ;

absr := advanced_brep_shape_representation ('absr', [cylxcyl], grc1) ;

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

(* Define representation using transrot1 only  *)
absr1 := advanced_brep_shape_representation ('absr1',
                                        [transrot1], grc1 ) ;

(* Define representation which is an assembly of intersecting
    cylinders + mapped  (translated) copy.
*)
mapping2 := representation_map (baseaxes, absr );
trans2 := mapped_item ('trans2', mapping2, topaxes );
```

```
absrass := advanced_brep_shape_representation
         ('absrass', [cylxcyl, trans2, baseaxes], grc2 ) ;

shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr1 );
shape2_def_rep := shape_definition_representation
                        (shape_2_def, absrass );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

other6: After processing the mapped_item shall be correctly interpreted, result is a rotated and translated copy of original B-rep.

other7: The **advanced_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.

aim261: The two distinct **advanced_brep_shape_representation**s shall not be spatially related.

6.3.8 Test case ab8

Test case summary:

Test case ab8 tests the use of **swept_surfaces** in the definition of a simple closed solid. The surfaces used are a **surface_of_linear_extrusion**, a **surface_of_revolution** and planes. The swept curve is a rational B-spline curve. This test also includes examples of reversal of the surface normal to define a **face_surface**. The definition of the faces is provided by the **swept_faces** context using the original parameters. (See C.4.6.)

6.3.8.1 Preprocessor

Input specification:

See 20

NOTE 1 See Test Case ps1 for details of product and part.

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Table 20 – Preprocessor details: test case ab8

Id	V	Application Element	Value	Req
@2		advanced_B_rep	#absr	C1
@73	*	B-rep	#swept_surf_solid	M
@73	*	B-rep.outer	#shell_object	C1
@30.1	*	face as advanced_face	#rotf	C1
@65.1		face.face_geometry	@48.1	C1
@30.2	*	face as advanced_face	#tran	C1
@65.2		face.face_geometry	@45	C1
@30.3	*	face as advanced_face	#base	C1
@65.3	aim171	face.face_geometry	#plbase	C1
@9		shell as closed_shell	#shell_object	C1
@11		shell.faces	[#rotf, #tran, #base	
			#front, #side, #back]	C1
@45	*	surface_of_extrusion	#strans	M
@46	*	surface_of_extrusion. extruded_curve	@51	C1
@47	*	surface_of_extrusion.direction	#vec_z	C1
@48	*	surface_of_revolution	#srot	M
@49	*	surface_of_revolution. revolved_curve	@51	C1
@50	*	surface_of_revolution.axis	#arot	C1
@51	aim147, 159	twisted_curve	#bspl	M

Specific verdict criteria:

aim159: Model shall contain an example of **b_spline_curve** as **bezier_curve** and **rational_b_-spline_curve**. Some weight values shall differ from 1.0.

Extra details:

This test requires the definition of an open rational_bspline_curve and the use of this to define two touching swept_surfaces, these should be a surface_of_linear_extrusion, and a surface_of_revolution which touch along the original B-spline curve. Bounding planes are added to construct a closed figure defined as an manifold_solid_brep; one face_surface shall have same_sense as FALSE. This is then the sole item in the definition of an advanced_brep_shape_representation.

NOTE 2 See EXPRESS-I specification for dimensional and other details.

6.3.8.2 Postprocessor

AIM test purpose coverage:

aim69, aim111, aim120, aim121, aim147, aim159, aim168, aim169, aim170, aim171, aim172.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

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

```
*)
TEST_CASE example_abrep_8; WITH part_204_brep_product_schema;

 REALIZATION

 LOCAL
    rotf, tran, base, front, back, side : advanced_face ;
    shell_object : closed_shell ;
    swept_surf_solid : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;


   CALL swept_faces ; -- uses default values, so no WITH
     IMPORT (rotf  := @afrot ;
             tran  := @aftran ;
             base  := @afbase ;
             front := @affront ;
             back  := @afback ;
             side := @afside; ) ;
   END_CALL;
      shell_object := closed_shell('shell_object',
                          [rotf, tran, base, front, side, back] ) ;
      swept_surf_solid := manifold_solid_brep('swept_surf_solid',
                                                  shell_object);
```

```
      its_units := named_unit(len_exp) || length_unit() ||
                              si_unit (’milli’, ’metre’) ;
     its_context := geometric_representation_context
               (’its_context’, ’context_for_swept_surf_solid’, 3) ||
                    global_unit_assigned_context ( [its_units] ) ;

      absr := advanced_brep_shape_representation
                (’absr’, [swept_surf_solid], its_context );
      shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim69: Face normal shall be correctly defined when face_surface has same_sense = FALSE (face normal opposite to surface normal).

aim116: B_spline_curve shall not be closed, start point and end point distinct.

aim159: Geometry of rational_b_spline curve shall be exactly represented.

aim168: Surface_of_linear_extrusion shall be correctly interpreted and located.

aim169: Surface_of_revolution shall be correctly interpreted and located (to join surface_of_-linear_extrusion after trimming with edges). Model shall contain no bounded_surfaces.

6.3.9 Test case ab9

Test case summary:

Test case ab9 tests the use of a **b_spline_surface** of Bezier type in the definition of a closed solid in the form of an inverted pyramid (or ’parachute’ shape) with curved top face and 4 planar faces. **edge_curve**s are straight lines or **b_spline_curve**s. The definition of the faces is provided by the **b_spline_pyramid_-faces** context using the original parameters. (See C.4.7.)

6.3.9.1 Preprocessor

Input specification:

See table 21.

NOTE 1 See Test Case ps1 for details of product and part.

Table 21 – Preprocessor details: test case ab9

Id	V	Application Element	Value	Req
@2		advanced_B_rep	#absr	C2
@73		B-rep	#inv_pyramid	C2
@73		B-rep.outer	@9	C1
@25		edge	#ec1	C1
@24	*	edge.edge_geometry	#bsplc4	C1
@30		face as advanced_face	#top	C1
@65	*	face.face_geometry	@39	M
@63	*	face.bounds	[#fbtop]	C1
@51	aim117	twisted_curve	#bsplc4	C1
@39	*	sculptured_surface	#bsplsurf	C1
	aim131, 134	as b_spline_surface		
@9		shell as closed_shell	#shell_object	C1
@11		shell.faces	[#left, #right,	
			#front, #top, #back]	C1

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The items attribute of the advanced_brep_shape_representation entity shall reference only the identifier of the B-rep.

Specific verdict criteria:

Extra details:

This test requires the definition of a manifold_solid_brep as a closed solid in the form of an inverted pyramid (or 'parachute' shape) with curved top face, whose geometry is defined by a b_spline_surface of degree 3 in each parameter, and 4 planar faces. **edge_curve**s are straight lines or **b_spline_curve**s. This B-rep defines an **advanced_brep_shape_representation**.

6.3.9.2 Postprocessor

AIM test purpose coverage:

aim114, aim117, aim131, aim134, aim136, aim137, aim139, aim148.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

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

```
*)
TEST_CASE example_abrep_9; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    left, right, front, back, top : advanced_face ;
    shell_object : closed_shell ;
    inv_pyramid  : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : geometric_representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

    CALL b_spline_pyramid_faces ; -- uses default values, so no WITH
      IMPORT (left  := @afleft ;
              right := @afright ;
              front := @affront ;
              back  := @afback ;
              top   := @aftop; ) ;
    END_CALL;
      shell_object := closed_shell('shell_object',
                                   [left, right, front, back, top] ) ;
      inv_pyramid := manifold_solid_brep('inv_pyramid', shell_object);

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

      its_context := geometric_representation_context
                      ('its_context', 'context_for_inv_pyramid', 3) ||
```

```
                    global_unit_assigned_context ( [its_units] ) ;

      absr := advanced_brep_shape_representation
                ('absr', [inv_pyramid], its_context );

      shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim114: B-spline curves defining edge geometry shall not self-intersect, this shall be verified for curve with **self_intersect** as UNKNOWN.

aim117: **b_spline_curve** subtype with**closed_curve** as UNKNOWN shall be correctly processed; all B-spline curves in this test case shall in fact be open.

aim131: B-spline surface shall be open in the v parameter direction.

aim134: B-spline surface shall be open in the u parameter direction.

aim148: **bezier_surface** subtype of **b_spline_surface** shall be correctly interpreted.

6.3.10 Test case ab10

Test case summary:

Test case ab10 tests the use of a **rational_b_spline_surface** in the definition of a closed solid in the form of an inverted pyramid (or 'parachute' shape) with curved top face and 4 planar faces. **edge_curve**s are straight lines or **rational_b_spline_curve**s. The definition of the faces is provided by the **rational_b_-spline_pyramid_faces** context using the original parameters. (See C.4.8.)

6.3.10.1 Preprocessor

Input specification:

See table 22

NOTE 1 See Test Case ps1 for details of product and part.

Constraints on values:

Table 22 – Preprocessor details: test case ab10

Id	V	Application Element	Value	Req
@2		advanced_B_rep	#absr	C2
@73 @73		B-rep B-rep.outer	#inv_pyramid #shell_object	C2 C1
@25 @24	 *	edge edge.edge_geometry	#ec1 #rbsplc1	C1 M
@30 @65	 *	face as advanced_face face.face_geometry	#top #rbspsurf	C1 M
@63	*	face.bounds	[#fbtop]	C1
@44	aim138, 139	sculptured_surface as rational_b_spline_surface	#rbspsurf	M
@9 @11		shell as closed_shell shell.faces	#shell_object [#left, #right, #front, #top, #back]	C1 C1
@51	aim120	twisted_curve	#rbsplc1	M

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The **items** attribute of the **advanced_brep_shape_representation** entity shall consist only of the identifier of the B-rep.

Specific verdict criteria:

aim138: Model shall contain an example of **b_spline_surface** as **rational_b_spline_surface**.

NOTE 2 The definition of this entity requires that this subtype shall be instantiated with one of the other subtypes of **b_spline_surface**.

Extra details:

This test case requires the definition of a manifold_solid_brep as a closed solid in the form of an inverted pyramid (or 'parachute' shape) with curved top face, whose geometry is defined by a **rational_b_spline_surface** of degree 3 in each parameter and of Bezier type, and by 4 planar faces. **edge_curve**s are straight lines or **rational_b_spline_curve**s. This B-rep defines an **advanced_brep_shape_representation**.

6.3.10.2 Postprocessor

AIM test purpose coverage:

aim120, aim121,aim116, aim138, aim139.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

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

```
*)
TEST_CASE example_abrep_10; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    left, right, front, back, top : advanced_face ;
    shell_object : closed_shell ;
    inv_pyramid  : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

    CALL rational_b_spline_pyramid_faces ;
(*    uses default values, so no WITH  *)
      IMPORT (left  := @afleft ;
              right := @afright ;
              front := @affront ;
              back  := @afback ;
              top   := @aftop; ) ;
    END_CALL;
      shell_object := closed_shell('shell_object',
                                   [left, right, front, back, top] );
      inv_pyramid := manifold_solid_brep('inv_pyramid', shell_object);

      its_units := named_unit(len_exp) || length_unit() ||
```

```
                              si_unit (’milli’, ’metre’) ;

    its_context := geometric_representation_context
                     (’its_context’, ’context_for_inv_pyramid’, 3) ||
                     global_unit_assigned_context ( [its_units] ) ;

      absr := advanced_brep_shape_representation
                (’absr’, [inv_pyramid], its_context );

      shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim116: B-spline curves shall all be verified to be open.

aim120: Geometry of **rational_b_spline_curve** shall be correctly interpreted, curve shall lie along edge of **rational_b_spline_surface** and shall be planar.

aim138: Geometry of **rational_b_spline_surface**, with some weights not equal to 1.0, shall be correctly interpreted. Complex subtype shall be correctly processed.

(geometry should NOT, with specified weights be identical to that of test case ab9)

6.3.11 Test case ab11

Test case summary:

Test case 11 tests the use of a **b_spline_surface** which is closed in both parametric directions to define a solid of genus 1 of a similar form to a torus. The test includes the use of an **edge_loop** traversed in opposite directions to define both surface boundaries.

The definition of the **face** is provided by the **genus_1_face_surface** context using the original parameters. (See C.4.9.)

6.3.11.1 Preprocessor

Input specification:

See table 23

NOTE 1 See Test Case ps1 for details of product and part.

Table 23 – Preprocessor details: test case ab11

Id	V	Application Element	Value	Req
@2		advanced_B_rep	#absr	C2
@73 @73	*	B-rep B-rep.outer	#genus_1_solid #shell_object	M C1
@25.1 @24.1	*	edge edge.edge_geometry	#ec #bsplc	 M
@25.2 @24.2	*	edge edge.edge_geometry	#ec_t #bsplc	C1 C1
@30 @65 @64.1	*	face as advanced_face face.face_geometry face.bounds	#complex_face #bspsurf [#fbleft, #fbright]	C1 M C1
@64.2 @35.1		face_bound loop as edge_loop	#fbleft #eloop	C1 C1
@64.3 @35.2 @67	*	face_bound loop as edge_loop loop.edges	#fbright #eloop [#ec_t]	C1 C1 C1
@39	aim140 aim133, 135	sculptured_surface as b_spline_surface_with_knots	#bspsurf	M
@9 @10		shell as closed_shell shell.faces	#shell_object [#complex_face]	C1 C1
@51	aim122	twisted_curve as b_spline_curve_with_knots	#bsplc	M

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The **items** attribute of the **advanced_brep_shape_representation** entity shall consist only of the identifier of the B-rep.

Specific verdict criteria:

Extra details:

This test requires the Creation of a single **b_spline_surface** which is closed in both parametric directions and is of a similar geometric form to a torus. Use this, together with one **edge_loop** which is re-used in opposite senses to create a genus 1 **face_surface** and hence a **manifold_solid_brep**.

NOTE 2 See EXPRESS-I specification for dimensional and other details.

6.3.11.2 Postprocessor

AIM test purpose coverage:

aim61, aim64, aim122, aim123, aim127, aim129, aim130, aim133, aim140, aim143, aim146.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

NOTE 2 The **genus_1_face_surface** context is used, with default parameters, to define the face and all geometry and topology of the B-rep.

```
*)
TEST_CASE example_abrep_11; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    complex_face : advanced_face ;
    shell_object : closed_shell ;
    genus_1_solid : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

    CALL genus_1_face_surface ; -- uses default values, so no WITH
      IMPORT (complex_face  := @g_1_face ; ) ;
    END_CALL;
      shell_object := closed_shell('shell_object', [complex_face] ) ;
      genus_1_solid := manifold_solid_brep ('genus_1_solid',
                                                shell_object) ;

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

```
      its_context := geometric_representation_context
                  ('its_context', 'context_for_genus_1_solid', 3) ||
                    global_unit_assigned_context ( [its_units] ) ;

      absr := advanced_brep_shape_representation
                ('absr', [genus_1_solid], its_context );

      shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr );

END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim61: face without outer bound shall be correctly interpreted.

aim64: face_surface with **edge_loop** with orientation FALSE shall be correctly interpreted - normal to face shall point out of enclosed volume.

aim122: b_spline_curve_with_knots shall be correctly processed, entire curve shall lie on b_-spline_surface.

aim133: B-spline surface shall be closed in u-parameter direction.

aim130: B-spline surface shall be closed in both parametric directions to define a toroidal like surface of genus 1.

aim146: b_spline_surface_with_knots shall be correctly processed, surface has different degrees for the two parameters.

6.3.12 Test case ab12

Test case summary:

Test case 12 tests the use of a **degenerate_toroidal_surface** to define a simple solid the halves of which are bounded by a degenerate toroidal surface of a different type, 'apple' or 'lemon'. The solid is closed by planar faces. The definition of the faces is provided by the **degenerate_torus** context using the original parameters. (See C.4.10.)

6.3.12.1 Preprocessor

Input specification:

See table 24

NOTE 1 See Test Case ps1 for details of product and part.

Table 24 – Preprocessor details: test case ab12

Id	V	Application Element	Value	Req
@2		advanced_b_rep	#absr	C2
@73		B-rep	#deg_tor_solid	C2
@73		B-rep.outer	#shell_object	C1
@30.1		face as advanced_face	#apple_face	C1
@65.1	*	face.face_geometry	#apple	M
@63.1		face.bounds	[#fbapple]	C1
@30.2		face as advanced_face	#lemon_face	C1
@65.2	*	face.face_geometry	#lemon	M
@63.2		face.bounds	[#fblemon]	S
@107.1	aim42	toroidal_surface	#apple	C1
@112.1	*	torus.major_radius	30.0	C1
@112.2	*	torus.minor_radius	50.0	C1
@107.2	aim43	toroidal_surface	#lemon	C1
@112.3	*	torus.major_radius	30.0	C1
@112.4	*	torus.minor_radius	50.0	C1
@9		shell as closed_shell	#shell_object	C1
@11		shell.faces	[#apple_face, #lemon_face, #left, #right]	M

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The **items** attribute of the **advanced_brep_shape_representation** entity shall consist only of the identifier of the B-rep.

Specific verdict criteria:

Extra details:

This test requires the creation of two **degenerate_toroidal_surface**s with the same radii centred at the origin, one surface is of 'apple' form (**select_outer** TRUE), the other is a 'lemon' (**select_outer** FALSE). Each surface is then bounded by circular arcs which are their intersections with the plane z = 0. A **manifold_solid_brep** is then constructed with 4 **advanced_face**s which correspond to opposing halves of these degenerate surfaces and the enclosed sections of the central plane which are bounded by inter-

secting circular arcs of the major and minor radius respectively. This **manifold_solid_brep** is then the sole **item** in an **advanced_brep_shape_representation**..

NOTE 2 See EXPRESS-I specification for dimensional and other details.

6.3.12.2 Postprocessor

AIM test purpose coverage:

aim42, aim43.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

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

```
*)
TEST_CASE example_abrep_12; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    apple_face, lemon_face, left, right : advanced_face ;
    shell_object : closed_shell ;
    deg_tor_solid : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

   CALL degenerate_torus ; -- uses default values, so no WITH
      IMPORT (apple_face  := @afapple ;
              lemon_face  := @aflemon ;
```

```
            left        := @afleft ;
            right       := @afright ; ) ;
   END_CALL;
      shell_object := closed_shell('shell_object',
             [apple_face, lemon_face, left, right] ) ;
      deg_tor_solid := manifold_solid_brep ('deg_tor_solid',
                                                shell_object) ;

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

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

      absr := advanced_brep_shape_representation
                ('absr', [deg_tor_solid], its_context );

      shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim42: The 'apple' shaped surface shall intersect the central plane on **edge_curve**s defined by circular arcs.
Normal to this surface correctly interpreted - normal shall point out of enclosed volume.

aim43: The 'lemon' shaped surface shall intersect the central plane on the **edge_curve**s defined.
Normal to this surface correctly interpreted - normal shall point out of enclosed volume.

6.3.13 Test case ab13

Test case summary:

Test case ab13 is geometrically similar to the most basic test case ab1 and consists of the faces needed to define a single solid cylinder with hemispherical base and elliptic top. All edge curves have their geometry defined as surface curves referencing pcurves in the parameter spaces of the cylinder, sphere or plane. The definition of the faces is provided by the **cylinder_sphere_pcurve** context using the original parameters. (See C.4.11.)

6.3.13.1 Preprocessor

Input specification:

See table 25

NOTE 1 See Test Case ps1 for details of product and part.

Table 25 – Preprocessor details: test case ab13

Id	V	Application Element	Value	Req
@2		advanced_B_rep	#absr	C2
@73	*	B-rep	#cysp_solid	C2
@20		cylindrical_surface	#cyl	
@21		cylindrical_surface.axis	#a1	C1
@22		cylindrical_surface.radius	25.0	C1
@26.1	*	ellipse	#elli	
@27.1		ellipse.location	#a2	S
@28.1		ellipse.major_radius	25*rt(2)	C1
@29.1		ellipse.minor_radius	25.0	C1
@26.2pelli	*	ellipse	#pelli	M
@27.2		ellipse.location	#pap2	S
@28.2		ellipse.major_radius	25*rt(2)	C1
@29.2		ellipse.minor_radius	25.0	C1
@30		face as advanced_face	#face_3	C1
@65		face.face_geometry	#sphere	C1
@63		face.loops	[#bc]	C1
@34	*	location	#a2	C1
@83	*	location as axis2_placement_2d	#pap2	M

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The **items** attribute of the **advanced_brep_shape_representation** entity shall consist only of the identifier of the B-rep.

Specific verdict criteria:

ae82: The two pcurves in the **associated_geometry** set shall represent identical geometry in 3D space.

Table 25 – concluded

Id	V	Application Element	Value	Req
@102		plane	#pl	C1
@76.1	*	curve as pcurve	#pcrv1	M
@77.1	*	pcurve.basis_surface	#sphere	M
@78.1	*	pcurve.curve_2d	#dr1.items = [#plin1]	M
@76.2	*	curve as pcurve	#pcrv2	M
@77.2	*	pcurve.basis_surface	#cyl	M
@78.2	*	pcurve.curve_2d	#dr2.items = [#plin2]	M
@76.3	*	curve as pcurve	#pcrv3	M
@77.2	*	pcurve.basis_surface	@pl	M
@77.3	*	pcurve.curve_2d	#dr3.items = [#pelli]	M
@41		spherical_surface	#sphere	C1
@42		spherical_surface.radius	25.0	C1
@79.1	*	surface_curve	#scrv_top	C1
@80.1		surface_curve.curve_3d	#circ	C1
@82	aim165	suface_curve.associated_geometry	[#pcrv1, #pcrv2]	M
@79.2	*	surface_curve	#scrv_bot	C1
@80.2		curve_3d	#elli	S
@81	aim164	associated_geometry	[#pcrv3]	M

Extra details:

Create an **advanced_brep_shape_representation** 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 base and a sloping planar top. All **edge_curve**s shall have their geometry defined by instances of **surface_curve**s. The **surface_curve** at the intersection of the sphere and the cylinder shall reference 2 **pcurve**s, the top boundary shall reference a single **pcurve** defined in the parameter space of the planar surface. 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, all shall be of type **advanced_face**. A suitable set of dimensions is defined in the EXPRESS-I specification below.

6.3.13.2 Postprocessor

AIM test purpose coverage:

aim87, aim151, aim152, aim155, aim156, aim157, aim164, aim165, aim166, aim167, aim256, aim259.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

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

```
*)
TEST_CASE example_abrep_13; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    face_1, face_2, face_3 : advanced_face ;
    shell_object : closed_shell ;
    cysp_solid : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

   CALL cylinder_sphere_pcurve ; -- uses default values, so no WITH
      IMPORT (face_1 := @top_face ;
              face_2 := @curved_face ;
              face_3 := @bottom_face ; ) ;
   END_CALL;
      shell_object := closed_shell('shell_object', [face_1, face_2,
                                                    face_3] );
      cysp_solid := manifold_solid_brep ('cysp_solid', shell_object) ;

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

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


      absr := advanced_brep_shape_representation
                 ('absr',  [cysp_solid], its_context );
```

```
      shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim151: surface_curve shall be correctly interpreted with parametrisation defined from master_-representation.

aim152: pcurve shall be correctly interpreted, after interpretation pcurve shall be geometrically identical to corresponding **curve_3d** identified as **master_representation**.

aim166: 2 **pcurve**s shall, after evaluation, coincide with each other and with the corresponding **curve_3d** instance.

6.3.14 Test case ab14

Test case summary:

Test case ab14 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 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. (See C.4.1.)

Other test purpose coverage:

other14: **mapped_item** with **mapping_target** as **cartesian_transformation_operator_3d**.

6.3.14.1 Preprocessor

Input specification:

See table 26.

NOTE 1 See Test Case ps1 for details of product and part.

NOTE 2 See Test Case ab1 for details of shell_object.

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

Table 26 – Preprocessor details: test case ab14

Id	V	Application Element	Value	Req
@2.1	*	advanced_B_rep	#absr	M
@2.2	*	advanced_brep	#absrass	M
@73.1	*	B-rep	#cysp_solid	C1
@73.1	*	B-rep.outer	#shell_object	C1
@73.2	other14	B-rep (transformed)	#cstrans	M
@34		location as axis2_placement_3d	#oldaxes	C1
@49	*	transformation	#transform	M
@50	*	scale	0.8	C1

C2: The context of the transformed B-rep and the assembly must not be the same as that of the original B-rep object.

Specific verdict criteria:

ae50: Model shall contain an example of transformation with scaling factor.

other14: Model shall contain an example of **mapped_item** with **mapping_target** as **cartesian_transformation_operator_3d**.

Extra details:

This test requires the creation of an **advanced_brep_shape_representation** consisting of a single **manifold_solid_brep**. The B-rep should be in the form of a solid 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.

6.3.14.2 Postprocessor

AIM test purpose coverage:

aim15, aim16, aim18, aim20, aim21, aim23, aim211, aim212, aim213, aim268.

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

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

NOTE 3 **csrans** should be a rotated scaled and translated copy of **cysp_solid**.

```
  *)

TEST_CASE example_abrep_14; WITH part_204_brep_product_schema;

  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 ;
    absr, absrass : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    grc1, grc2 : representation_context ;
    cstrans  : mapped_item ;
    mapping1  : representation_map ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

   CALL cylinder_sphere_faces ; -- uses default values, so no WITH
      IMPORT (face_1 := @top_face ;
              face_2 := @curved_face ;
              face_3 := @bottom_face ;
              pos_x  := @pos_x ;
              pos_y  := @pos_y ;
              pos_z  := @pos_z ;
              origin := @origin ;
              radius := @rad ; ) ;
   END_CALL;
```

```
shell_object := closed_shell('shell_object', [face_1, face_2, face_3]);
cysp_solid := manifold_solid_brep ('cysp_solid', shell_object) ;

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

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

grc2 := geometric_representation_context ('grc2',
      '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',
            'rotate and scale', pos_x, neg_z, neworigin, 0.8, pos_y );

absr := advanced_brep_shape_representation ('absr',
                                [cysp_solid, oldaxes], grc1 ) ;

mapping1 := representation_map (oldaxes, absr );
(* 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 which is an assembly of original B-rep +
transformed (scaled and translated and rotated) copy.*)


absrass := advanced_brep_shape_representation
         ('absrass', [cysp_solid, cstrans], grc2 ) ;

   shape1_def_rep := shape_definition_representation
                      (shape_1_def, absrass );


  END_REALIZATION;
```

```
END_TEST_CASE;
(*
```

Specific verdict criteria:

other14: After processing the B-rep solid defined by **mapped_item** shall be correctly scaled and positioned.

aim268: 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.

6.3.15 Test case ab15

Test case summary:

Test case ab15 is a test case consisting of the six faces needed to define a single solid cylinder with vertical axis trimmed by two parallel vertical planes. All geometry is explicitly defined with no defaults. The definition of the faces is provided by the **trimmed_cylinder** context using the original parameters. (See C.4.12.) The purpose of this test case is to test the re-use of a surface to define two faces and the re-use of curves to define the geometry of more than one edge.

6.3.15.1 Preprocessor

Input specification:

See table 27

NOTE 1 See Test Case ps1 for details of product and part.

Constraints on values:

C1: The values of the geometric dimensions are interdependent and can only be varied in a manner which ensures the geometric and topological integrity of the B-rep being defined. In general this is possible by varying the parameters in the EXPRESS-I context used in the definition of the postprocessor test.

C2: The **items** attribute of the **advanced_brep_shape_representation** entity shall consist only of the identifier of the B-rep.

Specific verdict criteria:

ae59: Model shall contain an example of curve used to define the geometry of more than one edge. (This curve is a circle.)

ae66: Model shall contain an example of surface used to define the geometry of more than one face. (this surface is a cylindrical_surface.)

Table 27 – Preprocessor details: test case ab15

Id	V	Application Element	Value	Req
@2.absr		advanced_B_rep	#absr2	C2
@73		B-rep	#trimmed_cyl	C2
@73		B-rep.outer	#shell_object	S
@6	*	circle	#circbas	M
@8		circle.radius	25.0	C1
@7		circle.location	#a1	C1
@59	*	curve to edge	@6 to (@24.1 = #edbasb AND @24.2 = #edbasf)	M
@9		shell as closed_shell	#shell_object	
@11		shell.faces	[#back ,#base, #front, #left, #right, #top]	S
@20	*	cylindrical_surface	#cyl	C1
@21		cylinder.axis	#a1	C1
@22		cylinder.radius	25.0	C1
@66	*	surface to face	@20 to (@30.1, @30.1)	M
@24.1	*	edge as edge_curve	#edbasf	C1
@25.1		edge.start_vertex	#vblf	
@25.2		edge.end_vertex	#vbrf	
@59.1	*	edge_geometry	@6.circbas	M
@24.2	*	edge as edge_curve	#edbasb	C1
@25.3		edge.start_vertex	#vbrb	
@25.4		edge.end_vertex	#vblb	
@59.2	*	edge.edge_geometry	@6.circbas	M
@30.1		face as advanced_face	#front	C1
@66.1	*	face.geometry	@20.cyl	M
@63.1		face.loops	[#bfront]	C1
@30.2		face as advanced_face	#back	C1
@66.2	*	face.geometry	@20.cyl	M
@63.2		face.loops	[#bback]	C1

Extra details:

This test requires the creation of an **advanced_brep_shape_representation** consisting of a single **manifold_solid_brep**. The **manifold_solid_brep** should be in the form of a solid cylinder with vertical axis. The cylinder is trimmed by two planes normal to the y axis. The centre of the base of the cylinder is at the origin. The B-rep object is defined by a single closed shell with 6 faces, all shall be of type **advanced_face**. A suitable set of dimensions is defined in the EXPRESS-I specification below.

6.3.15.2 Postprocessor

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used to define the product context for the geometric definitions.

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

```
*)
TEST_CASE example_abrep_15; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    back, base, front, left, right, top : advanced_face ;
    shell_object : closed_shell ;
    trimmed_cyl : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    its_context : representation_context ;
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
  END_LOCAL;

   CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
   END_CALL;

  CALL trimmed_cylinder ; -- uses default values, so no WITH
      IMPORT (top := @top_face ;
              base := @base_face ;
              front := @front_face ;
```

```
            back := @back_face ;
            left := @left_face ;
            right := @right_face ; ) ;
   END_CALL;
      shell_object := closed_shell('shell_object', [top, base, front,
                                                back, left, right] );
      trimmed_cyl := manifold_solid_brep('trimmed_cyl', shell_object);

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

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


      absr := advanced_brep_shape_representation
                ('absr',  [cysp_solid], its_context );
       shape1_def_rep := shape_definition_representation
                       (shape_1_def, absr );

 END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

ae59: The two edges referencing each circle shall be distinct and have their geometry correctly trimmed by their vertices.

ae66: The two faces lying on the cylindrical surface shall be correctly bounded by their edge loops.

6.4 Abstract test cases for name preservation UoF

6.4.1 Test case np1

Test case summary:

Test case np1 is designed to test the definition of an advanced B-rep with, or without names for some of the geometric and topological elements. The **cylinder_sphere_shell** context is used with different parameters to define the outer shell of cylindrical form with a hemispherical base and a void shell. The result is a hollow solid with void(s) of a similar shape, or spherical.

6.4.1.1 Preprocessor

Input specification:

See table 28

NOTE 1 See Test Case ps1 for details of product and part.

NOTE 2 See Test Case ab1 for details of shell_object.

Table 28 – Preprocessor details: test case np1

Id	V	Application Element	Value	Req
@2.1		advanced_B_rep	#absr1	S
@2.2		advanced__rep	#absr2	S
@3.1	*	B-rep	#cysp_with_void	
@406	aim301	name	""	M
@3.2		B-rep	#cyl_sphere	S
@301	aim251	name	'cyl_sphere'	S
@30.1		advanced_face	#vface_1	S
@30.2		advanced_face	#vface_2	S
@30.3		advanced_face	#vface_3	S
@9		shell	#shell_object	S
@303	*	name	""	M
@9		shell	#hollow1	S
@302	aim187	name	'hollow1'	S
@55		shell	#void1	S

Specific verdict criteria:

aim187: Model shall contain an example of geometric_representation_item with user defined non-null name.

aim187: Model shall contain an example of topological_representation_item with user defined non-null name,

aim188: Model shall contain example of topological_representation_item subtype with name as ",

aim189: Model shall contain an example of geometric_representation_item subtype with name as ".

Extra details:

This test requires the creation of an **advanced_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 base and a sloping planar top. One such B-rep shall contain a void of a similar shape and orientation. The centre of the hemisphere 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. Some of the geometric elements and topological elements shall be named, others shall have the name attribute as a null string.

6.4.1.2 Postprocessor

AIM test purpose coverage:

aim187, aim188, aim189, aim251

Input specification:

NOTE 1 In the specification below the **basic_product_structure** context is used twice to define the product contexts for the geometric definitions.

NOTE 2 The **cylinder_sphere_faces** context is used, to define the faces of the B-reps.

```
*)
TEST_CASE example_name_1; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    prod2_name : STRING := 'prod2_name';
    shape_1_def, shape_2_def : product_definition_shape ;
    shape1_def_rep, shape2_def_rep : shape_definition_representation;
    face_1, face_2, face_3 : advanced_face ;
    vface_1, vface_2, vface_3 : advanced_face ;
    shell_object, hollow1  : closed_shell ;
    void1 : oriented_closed_shell;
    cyl_sphere : manifold_solid_brep ;
    cylsp_with_void : brep_with_voids ;
    absr1, absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    context1, its_context : representation_context ;
  END_LOCAL;

    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;

    CALL cylinder_sphere_faces ; -- uses default values, so no WITH
```

```
      IMPORT (face_1 := @top_face ;
              face_2 := @curved_face ;
              face_3 := @bottom_face ; );
    END_CALL;

    CALL cylinder_sphere_faces;
(*   parameters re-set for dimensions -large void *)
      IMPORT (vface_1 := @top_face ;
              vface_2 := @curved_face ;
              vface_3 := @bottom_face ; );
      WITH (orc := 10; rad := 12; ht := 50; );
    END_CALL;

    CALL basic_product_structure ;  -- parameters for second product.
      IMPORT (shape_2_def := @prod_def_shape; );
      WITH   (prod_name := @prod2_name;
              pdef_desc := 'test product definition 2';
              propd_desc := 'shape of test product 2';
              prod_name :=  'second test product';
              prod_id   := 'P02' ;
              pdf_id    := 'PDF02' ; );
    END_CALL;

    shell_object := closed_shell ('', [face_1, face_2, face_3]) ;
    hollow1 :=closed_shell ('hollow1',  [vface_1, vface_2, vface_3]) ;
    void1 := oriented_closed_shell ('void1', hollow1, FALSE) ;
    cyl_sphere := manifold_solid_brep ('cyl_sphere', shell_object ) ;
    cylsp_with_void := brep_with_voids ('', shell_object, [void1]) ;

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

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

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


    absr1 := advanced_brep_shape_representation
             ( 'absr1', [cylsp_with_void], context1 ) ;

    absr2 := advanced_brep_shape_representation
             ( 'absr2', [cyl_sphere], its_context ) ;
   shape1_def_rep := shape_definition_representation
```

```
                        (shape_1_def, absr1 );
   shape2_def_rep := shape_definition_representation
                        (shape_2_def, absr2 );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim187: name 'hollow1' shall be preserved for void shell.

aim189: name = '' shall have the same semantics as name missing. outer shell shall be correctly interpreted with **name** = ''.

aim251: name = '' shall have the same semantics as name missing. hollow B-rep shall be correctly interpreted with **name** = ''.

aim251: name 'cyl_sphere' shall be preserved for solid B-rep.

6.5 Abstract test cases for product structure

The set of test cases given here covers the required scope of the test purposes defined for the product structure UoF.

The post-processor abstract test cases in this clause are fully documented in EXPRESS-I.

A simple textual description and table of instances 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. Criteria for these are documented separately as pre-processor or as post-processor criteria.

NOTE 1 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 all of the purposes documented in ps1.

6.5.1 Test case ps1

Test case summary:

Test case ps1 is the most basic test case, it defines and identifies a product whose shape is defined as a single solid cylinder with hemispherical base and elliptic top. The general purpose of this test is to verify that the associations between product, part and shape are correctly maintained. The application protocol and associated contexts shall also be defined in this and all other test cases. With the substitution of the appropriate **shape_representation** this test basic product definition test case should be included in all other test cases defined in this part of ISO 10303.

Other test purpose coverage:

other5: **advanced_brep_shape_representation** with context as **geometric_context** with **items** as **manifold_solid_brep**.

6.5.1.1 Preprocessor

Input specification:

See table 29.

NOTE 1 See Test Case ab1 for details of advanced_brep_shape_representation.

Table 29 – Preprocessor details: test case ps1

Id	V	Application Element	Value	Req
@426 @405	* *	part part to B-rep	#shape_1_def #shape_1_def to #absr	M
@407	*	product	#prod_d_f1	M
@409	*	coordinate_system	#prod1_context	M
@410 @411	* *	user_defined_name version_and_id	"prod1_name" #prod_d_f1.id and #prod_id	S M
@404	aim260	global_units	#its_units	M
@73		B-rep	#absr	S

Constraints on values:

C1: The part shape, the shape definition representation and the shape representation shall be associated with the **advanced_brep_shape_representation**.rb

Specific verdict criteria:

aim230: Model shall contain an example of product as **product_definition_formation** with **of_product** as **product**.

aim245: Model shall contain an example of **property_definition** as **product_definition_shape** with **definition** as**characterised_definition** as **shape_definition**.

aim247: Model shall contain an example of **shape_definition_representation** with **definition** as **product_definition_shape**.

aim246: Model shall contain an example of **shape_definition_representation** with **used_representation** as **representation** as **shape_representation**.

Extra details:

This test requires the creation of a named product consisting of a single part. The part shape shall be defined by a B-rep shape representation. For the purposes of this test case it is defined as a**advanced_-brep_shape_representation** consisting of a single **manifold_solid_brep**. The **manifold_solid_brep** is in the form of a solid cylindrical solid 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, all shall be of type **advanced_face**. A suitable set of dimensions is defined in the EXPRESS-I specification below.

6.5.1.2 Postprocessor

AIM test purpose coverage:

aim199, aim200, aim201, aim202, aim203, aim204, aim205, aim206, aim207, aim210, aim217, aim218, aim219, aim220, aim221, aim222, aim223, aim224, aim225, aim226, aim227, aim228, aim229, aim230, aim231, aim232, aim231, aim232, aim239, aim241, aim242, aim243, aim244, aim245, auim246, aim247, aim248, aim249, aim250, aim251, aim252, aim257, aim258, aim259, aim260, aim265, aim268.

Input specification:

NOTE 1 The **basic_product_structure** context is used to define the basic product with some new names.

NOTE 2 The most simple form of **cylinder_sphere_faces** context is used with default values to define shape.

```
*)
TEST_CASE example_prod_struc_1; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep : shape_definition_representation;
    face_1, face_2, face_3 : advanced_face ;
    shell_object : closed_shell ;
    cysp_solid : manifold_solid_brep ;
    absr : advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    prod1_context : representation_context ;
  END_LOCAL;

    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; ) ;
```

```
      WITH   (prod_name := @prod1_name; );
    END_CALL;

    CALL cylinder_sphere_faces ; -- uses default values, so no WITH
      IMPORT (face_1 := @top_face ;
              face_2 := @curved_face ;
              face_3 := @bottom_face ; ) ;
    END_CALL;
      shell_object := closed_shell('shell_object',
                              [face_1, face_2, face_3] );
      cysp_solid := manifold_solid_brep ('cysp_solid', shell_object);

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

    prod1_context := geometric_representation_context
                     ('context_1', 'context_for_cylinder_sphere', 3) ||
                     global_unit_assigned_context ( [its_units] ) ;


      absr := advanced_brep_shape_representation
                 ('absr',  [cysp_solid], prod1_context );
      shape1_def_rep := shape_definition_representation
                        (shape_1_def, absr );


  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim206: Postprocessor shall interpret **part_204_brep_product_schema**.

aim204: Postprocessor shall correctly interpret **application_protocol_definition** with **application** as **application_context**.

aim200: Postprocessor shall correctly interpret **application_context_element** with **frame_of_-reference** as **application_context**.

aim245: Postprocessor shall correctly interpret **property_definition** as **product_definition_shape**.

aim248: Postprocessor shall correctly interpret **property_definition_representation** as **shape_-definition_representation**.

aim268: Postprocessor shall correctly interpret **shape_representation** as **advanced_brep_shape_-representation**.

6.5.2 Test case ps2

Test case summary:

Test case ps2 is very similar to test case ps1 but uses an **elementary_brep_shape_representation**. It defines and identifies a product consisting of a single part whose shape is defined as a single solid cylinder with hemispherical base and elliptic top.

Other test purpose coverage:

other8: **elementary_brep_shape_representation** with context as **geometric_context** with **items** as **manifold_solid_brep**.

6.5.2.1 Preprocessor

Input specification:

NOTE 1 See Test Case eb1 for details of elementary_brep_shape_representation.

Table 30 – Preprocessor details: test case ps2

Id	V	Application Element	Value	Req
@405	*	part as B-rep	#shape_1_def	M
@407	*	product as part	#prod_d_f1	M
@409	*	product.coordinate_system	#prod1_context	M
@410	*	product.name	"prod1_name"	S
@411	*	product.version_and_id	#prod_d_f1.id and #prod_id	M
@426		part to B-rep	#shape1_def_rep to #ebsr	M
@113		B-rep as elementary_brep	#ebsr	S

Constraints on values:

C1: The part shape, the shape definition representation and the shape representation shall be associated with the **elementary_brep_shape_representation**.rb

Specific verdict criteria:

aim267: Model shall contain an example of **shape_representation** as **elementary_brep_shape_-representation**.

Extra details:

Create a named product consisting of a single part. The part shape shall be defined by a B-rep shape representation. For the purposes of this test case it is defined as an **elementary_brep_shape_representation** consisting of a single **manifold_solid_brep**. The **manifold_solid_brep** is in the form of a solid cylindrical solid 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, all shall be of type **face_surface**. A suitable set of dimensions is defined in the EXPRESS-I specification below.

6.5.2.2 Postprocessor

AIM test purpose coverage:

aim199, aim200, aim201, aim202, aim203, aim204, aim205, aim206, aim207, aim210, aim217, aim218, aim219, aim220, aim221, aim222, aim223, aim224, aim225, aim226, aim227, aim228, aim229, aim230, aim231, aim232, aim231, aim232, aim239, aim241, aim242, aim243, aim244, aim245, auim246, aim247, aim248, aim249, aim250, aim251, aim252, aim257, aim258, aim259, aim265, aim267.

Input specification:

NOTE 1 The **basic_product_structure** context is used to define the basic product with some new names.

NOTE 2 The most simple form of **cylinder_sphere_shell** context is used with default values to define shape.

```
*)
TEST_CASE example_prod_struc_2; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep2 : shape_definition_representation ;
    shell_object : closed_shell ;
    cysp_solid : manifold_solid_brep ;
    ebsr1 : elementary_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    prod1_context : representation_context ;
  END_LOCAL;

    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;
```

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

      its_units := named_unit(len_exp) || length_unit() ||
                              si_unit (’milli’, ’metre’) ;

      prod1_context := geometric_representation_context
                     (’context_1’, ’context_for_cylinder_sphere’, 3) ||
                       global_unit_assigned_context ( [its_units] ) ;

      ebsr1 := elementary_brep_shape_representation
                ( ’ebsr1’, [cysp_solid], prod1_context );
      cysp_solid := manifold_solid_brep (’cysp_solid’, shell_object) ;

    prod1_context := geometric_representation_context
                     (’context_1’, ’context_for_cylinder_sphere’, 3) ||
                     global_unit_assigned_context ( [its_units] ) ;


      ebsr1 := elementary_brep_shape_representation
                (’ebsr1’,  [cysp_solid], prod1_context );
      shape1_def_rep2 := shape_definition_representation
                       (shape_1_def, ebsr1 );


  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim267: Postprocessor shall correctly interpret **shape_representation** as **elementary_brep_shape_-representation**.

6.5.3 Test case ps3

Test case summary:

Test case ps3 is very similar to test case ps1 but uses a **faceted_brep_shape_representation**. It defines and identifies a product consisting of a single part whose shape is defined as a single solid tetrahedron with faces defined by polyloops.

Other test purpose coverage:

other11: **faceted_brep_shape_representation** with context as **geometric_context** with **items** as **faceted_brep**.

6.5.3.1 Preprocessor

Input specification:

See table 31

Table 31 – Preprocessor details: test case ps3

Id	V	Application Element	Value	Req
@405	*	part as B-rep	#shape_1_def	M
@407	*	product as part	#prod_d_f1	M
@409	*	product.coordinate_system	#prod1_context	M
@410	*	product.name	"prod1_name"	S
@411	*	product.version_and_id	#prod_d_f1.id and #prod_id	M
@426		shape to B-rep	#shape1_def_rep to #fbsr1	M
@200	*	B-rep as faceted B-rep	#fbsr1	S

Constraints on values:

C1: The part shape, the shape definition representation and the shape representation shall be associated with the **faceted_brep_shape_representation**.rb

Specific verdict criteria:

aim266: Model shall contain an example of **shape_representation** as **faceted_brep_shape_representation**.

Extra details:

This test requires the creation of a named product consisting of a single part. The part shape shall be defined by a B-rep shape representation. For the purposes of this test case it is defined as an **faceted_brep_shape_representation** consisting of a single **manifold_solid_brep** which is a **faceted_brep**. The **faceted_brep** is in the form of a tetrahedron with 3 edges parallel to the coordinate axes. All faces shall be defined by **poly_loop**s and shall be of type **face_surface** with the surface geometry defined by a plane. A suitable set of dimensions is defined in the EXPRESS-I specification below.

6.5.3.2 Postprocessor

AIM test purpose coverage:

aim199, aim200, aim201, aim202, aim203, aim204, aim205, aim206, aim207, aim210, aim217, aim218, aim219, aim220, aim221, aim222, aim223, aim224, aim225, aim226, aim227, aim228, aim229, aim230, aim231, aim232, aim231, aim232, aim239, aim241, aim242, aim243, aim244, aim245, auim246, aim247, aim248, aim249, aim250, aim251, aim252, aim257, aim258, aim259, aim265, aim266.

Input specification:

```
*)
TEST_CASE example_prod_struc_3; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    prod1_name : STRING := 'prod1_name';
    shape_1_def : product_definition_shape ;
    shape1_def_rep3 : shape_definition_representation ;
    shell_object : closed_shell ;
    tetrahedron  : faceted_brep ;
    fbsr1 : faceted_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                  dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    prod1_context : representation_context ;
  END_LOCAL;

    CALL basic_product_structure ;
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;

    CALL tetrashell_instance ; -- uses default values, so no WITH
      IMPORT (shell_object := @tetrashell; ) ;
    END_CALL;

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

      prod1_context := geometric_representation_context
                     ('context_1', 'context_for_cylinder_sphere', 3) ||
                       global_unit_assigned_context ( [its_units] ) ;

      fbsr1 := faceted_brep_shape_representation
                 ( 'fbsr1', [sp_solid], prod1_context );
      tetrahedron := manifold_solid_brep ('tetrahedron', shell_object) ;

    prod1_context := geometric_representation_context
```

```
                    ('context_1', 'context_for_tetrahedron', 3) ||
                    global_unit_assigned_context ( [its_units] ) ;


    fbsr1 := faceted_brep_shape_representation
              ('fbsr1',  [tetrahedron], prod1_context );
    shape1_def_rep3 := shape_definition_representation
                     (shape_1_def, fbsr1 );


  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

aim266: Postprocessor shall correctly interpret **shape_representation** as **faceted_brep_shape_-representation**.

6.5.4 Test case ps4

Test case summary:

Test case ps4 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 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. A simple assembly and a compound assembly are defined using the same basic geometry for the components.

Other test purpose coverage:

other15: **assembly_component_usage** with assembly as **SELF\product_definition_relationship.relating_-product_definition** as **product_definition** related to an instance of **shape_representation** by **product_-definition_shape** and **shape_definition_representation**.

other16: **assembly_component_usage** with component as **SELF\product_definition_relationship.-related_product_definition** as **product_definition** related to an instance of **shape_representation** by **product_definition_shape** and **shape_definition_representation**.

6.5.4.1 Preprocessor

Input specification:

NOTE 1 See Test Case ps1 for details of product and part.

NOTE 2 See Test Case ab1 for details of **cysp_solid**.

Table 32 – Preprocessor details: test case ps4

Id	V	Application Element	Value	Req
@2		advanced_B_rep	#absr	C1
@401.1	*	assembly as representation	#absrass	M
@401.2	*	asembly as representation	#absrass2	M
@412.1	*	assembly to component	#acu_1a	M
@420	*	part to assemblies	#comp1_def to #ass1_def and #ass2_def	C1
@412.2	aim209	assembly to component	#acu_1b	M
@419	*	assembly.part	#comp2_def	C1
@413	aim240	assembly_structure	#acu_2a	C1
@423		assembly	#ass2_def	M
@416	*	assembly.sub_assembly	#ass1_def	M
@418	*	assembly_structure	#acu_2b	C1
@418	*	assembly.part	#comp3_def	M
@2	*	advanced B-rep	#cysp_solid	S
	*	B-rep.outer	#shell_object	S
@419.1	*	transformed part	#cstrans	M
@426.1		part	#absr	C1
@49.1		transformation	#transform	C1
@419.2	*	transformed part	#cstrans2	M
@426.2		part	#absr	C1
@49.2		transformation	#trans2	C1
@50.1	*	transformation.scale	0.8	C1
@403	*	assembly.coordinate_system	#oldaxes	M

Constraints on values:

C1: The geometry of the assembly components and the definitions of the transformations are interdependent to ensure that the assembly components fit together.

Specific verdict criteria:

ae413:] Model shall contain an example of an Assembly containing one assembly in role of sub-assembly.

ae415: Model shall contain an example of an Assembly as part of one product.

ae416: Model shall contain an example of an Assembly as part of more than one product.

ae420: Model shall contain an example of a Part contained in more than one assemblies.

Table 32 – concluded

Id	V	Application Element	Value	Req
@408.1	aim208	product as assembly	#ass1_def	M
@402	*	product.name	"ass1_name"	S
@408.2	*	product as assembly	#ass2_def	S
@407.1	*	product as part	#comp1_def	C1
@426		part.shape	#absr	C1
@407.2		product as part	#comp2_def	C1
@407.3		product as part	#comp3	C1
@49.3		transformation	#transform	M
@50.2	*	transformation.scale	0.8	C1
@49.4	*	transformation	#trans2	M

ae423: Model shall contain an example of a Product containing many parts.

ae425: Model shall contain an example of a Part contained in many products.

ae427: Model shall contain an example of Part with geometry defined by many B-rep models. (See also 6.5.2)

aim240: Model shall contain an example of a **product_definition_relationship** as **product_definition_-usage** as **assembly_component_usage**.

other15: Model shall contain an example of **assembly_component_usage** with assembly as **SELF\product_definition_relationship.relating_product_definition** as **product_definition** related to an instance of **shape_representation** by **product_definition_shape** and **shape_definition_-representation**.

other16: Model shall contain an example of **assembly_component_usage** with component as **SELF\product_definition_relationship.related_product_definition** as **product_definition** related to an instance of **shape_representation** by **product_definition_shape** and **shape_definition_-representation**.

Extra details:

This test requires the creation of an **advanced_brep_shape_representation** consisting of a single **manifold_solid_brep**. The B-rep should be in the form of a solid 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**, two further representations consisting of rotated, translated and scaled (not by 1.0) copies 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. An assembly consisting

of two components and a compound assembly, using the first as sub-assembly are then defined using the assembly_component_usage construct.

6.5.4.2 Postprocessor

AIM test purpose coverage:

aim208, aim209, aim214, aim215, aim219, aim220, aim221, aim233, aim234, aim235, aim236, aim237, aim238, aim240.

Input specification:

NOTE 1 Test case of **mapped_item** and 'assembly' using simple cylindrical solid.

NOTE 2 Cylindrical shell is created by using **cylinder_sphere_faces** context with default parameters.

NOTE 3 **cstrans** and **cstrans2** should be rotated scaled and translated copies of **cysp_solid**.

```
  *)

TEST_CASE example_prod_struc_4; WITH part_204_brep_product_schema;

  REALIZATION

  LOCAL
    radius : length_measure ;
    origin, neworigin, negorigin : cartesian_point ;
    pos_x, pos_y, pos_z, neg_z  : direction ;
    oldaxes : axis2_placement_3d;
    transform : cartesian_transformation_operator_3d;
    shell_object  : closed_shell ;
    cysp_solid : manifold_solid_brep ;
    absr, absrass, absrass2, comp2, comp3 :
                    advanced_brep_shape_representation ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                   dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
    grc1, grc2, grc3 : representation_context ;
    cstrans  : mapped_item ;
    mapping1  : representation_map ;

    ass1_name : STRING := 'ass1_name';
    ass2_name : STRING := 'ass2_name';
    comp1_name : STRING := 'comp1_name' ;
    comp2_name : STRING := 'comp2_name' ;
    comp3_name : STRING := 'comp3_name' ;
```

```
  shape_1_def, shape_2_def : product_definition_shape ;
  comp1_shape, comp2_shape, comp3_shape  : product_definition_shape;
  shape1_def_rep, shape2_def_rep : shape_definition_representation;
  comp1_d_rep, comp2_d_rep, comp3_d_rep :
                                    shape_definition_representation;
  ass1_def, ass2_def, comp1_def, comp2_def, comp3_def :
                                                product_definition;
  acu_1a, acu_1b, acu_3a, acu_3b : assembly_component_usage ;
END_LOCAL;

 CALL basic_product_structure ;
    IMPORT (shape_1_def := @prod_def_shape;
             ass1_def   := @prod_def ; );
    WITH   (prod_name := @ass1_name; );
 END_CALL;

 CALL basic_product_structure ;
    IMPORT (shape_2_def := @prod_def_shape; );
    WITH   (prod_name := @ass2_name; );
 END_CALL;

 CALL basic_product_structure ;
    IMPORT (comp1_shape := @prod_def_shape;
             comp1_def   := @prod_def ; );
    WITH   (prod_name := @comp1_name; );
 END_CALL;

 CALL basic_product_structure ;
    IMPORT (comp2_shape := @prod_def_shape;
             comp2_def   := @prod_def ; );
    WITH   (prod_name := @comp2_name; );
 END_CALL;

 CALL basic_product_structure ;
    IMPORT (comp3_shape := @prod_def_shape;
             comp3_def   := @prod_def ; );
    WITH   (prod_name := @comp3_name; );
 END_CALL;

 CALL cylinder_sphere_faces ; -- uses default values, so no WITH
    IMPORT (face_1 := @top_face ;
            face_2 := @curved_face ;
            face_3 := @bottom_face ;
            pos_x  := @pos_x ;
            pos_y  := @pos_y ;
```

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

shell_object := closed_shell('shell_object',[face_1, face_2, face_3]);
cysp_solid := manifold_solid_brep ('cysp_solid', shell_object) ;

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


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

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

grc3 := geometric_representation_context ('grc3',
                      'context for compound 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] );
negorigin := 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',
           'translate rotate and scale', pos_x, neg_z, neworigin,
                                                   0.8, pos_y );
trans2 := cartesian_transformation_operator_3d ('trans2',
  'negtrans rotate and scale', pos_x, neg_z, negorigin, 0.8, pos_y );


absr := advanced_brep_shape_representation ('absr',
                                  [cysp_solid, oldaxes], grc1 ) ;

mapping1 := representation_map (oldaxes, absr );
```

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

(* cstrans2 is an 80% scaled copy of original rotated about x axis
and translated along this axis in the negative direction *)
cstrans2 := mapped_item ('cstrans2', mapping1, trans2 );


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

absrass := advanced_brep_shape_representation
        ('absrass', [cysp_solid, cstrans], grc2 ) ;

   shape1_def_rep := shape_definition_representation
                     (shape_1_def, absrass );

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

absrass2 := advanced_brep_shape_representation
        ('absrass', [cysp_solid, cstrans, cstrans3], grc3 ) ;

   shape2_def_rep := shape_definition_representation
                     (shape_2_def, absrass3 );

(* Define representations of individual components of assembly. *)

comp2 := advanced_brep_shape_representation
        ('comp2', [cstrans], grc2 ) ;

comp3 := advanced_brep_shape_representation
        ('comp3', [cstrans2], grc3 ) ;

comp1_d_rep := shape_definition_representation
                     (comp1_shape, absr );
comp2_d_rep := shape_definition_representation
                     (comp2_shape, comp2 );
comp3_d_rep := shape_definition_representation
                     (comp3_shape, comp3 );

(* Assemblies and components are designated using
assembly_component_usage. Simple 2 component assembly *)
acu_1a := assembly_component_usage('acu_1a', 'ass1_comp1',
      'part 1 of ass1', ass1_def, comp1_def, ? );
```

```
acu_1b := assembly_component_usage('acu_1a', 'ass1_comp2',
      'part 2 of ass1',  ass1_def, comp2_def, ? );

(* Compound assembly  = assembly1 + component 3 *)
acu_2a := assembly_component_usage('acu_2a', 'ass2_comp1',
       'part 1 of ass2', ass2_def, ass1_def, ? );
acu_2b := assembly_component_usage('acu_2b', 'ass2_comp2',
       'part 2 of ass2', ass2_def, comp3_def, 'rd1' );

  END_REALIZATION;
END_TEST_CASE;
(*
```

Specific verdict criteria:

ae416: Re-use of Assembly as part of more than one product shall be permissible.

ae419: Assembly containing many parts with associated transformations shall be correctly interpreted with touching components.

ae420: Re-use of Part in more than one assembly shall be supported.

aim238: **product_definition_relationship** as **product_definition_usage** as **assembly_component_usage** shall be correctly translated to define assembly structures.

6.6 Abstract test cases for visual presentation UoF

There are five basic presentation-related abstract test cases. These presentation attributes need to be tested separately for each of the three conformance classes (that is, for facetted, for elementary, and for advanced boundary representation solid models).

In order to demonstrate the extensibilty to other types of B-rep model test case vp4 has been modified as vp6 and vp7 respectively to add presentation attributes to an elementary B-rep model, or to an advanced B-rep model.

The intention is not to provide a specific (two-dimensional) screen image, but to set the rendering, viewing and presentation parameters, and to associate these with the product and its shape representation.

6.6.1 Test case vp1

Test case summary:

This test case presents a model with a positional light-source and some ambient light, with constant shading for each facet. Each curve is also rendered with constant colour. Hidden lines and hidden surfaces are removed.

test case	lights	surface_style _rendering	surface _style	curve_style _rendering	B-rep type	model
1	positional	constant_shading	ambient	constant_colour	F	tetrahedron
2	directional	colour_shading	ambient	linear_colour	F	tetrahedron
3	spot	dot_shading	diffuse		F	tetrahedron
4	spot, dir, pos	normal_shading	specular		F	tetrahedron
5	spot	dot_shading	diffuse		F	tetrahedron
6	spot, dir, pos	normal_shading	specular		E	cylinder/sphere
7	spot, dir, pos	normal_shading	specular		A	cylinder/sphere

Table 33 – Summary of rendering and models for visual presentation abstract test cases

6.6.1.1 Preprocessor

Input specification:

See table 34.

NOTE 1 See Test Case ps3 for details of product and part. See Test Case fb1 for details of shape representation. See Test Case vp4 for rendering and viewing values.

Table 34 – Preprocessor details: test case vp1

Id	V	Application Element	Value	Req
@511.point_style	*	point_appearance	(unset)	M

Specific verdict criteria:

aim301: **camera_model_d3** with **perspective_of_volume** as view_volume

aim302: **camera_model_d3** with **view_reference_system** as axis2_placement_3d

aim309: **colour** with **name** as label

aim300: **camera_model** as **camera_model_d3**

aim303: **camera_model_d3** as **camera_model_d3_with_light_sources**

aim304: **camera_model_d3** as **camera_model_d3_with_hlhsr**

aim307: **camera_model_with_light_sources** with **sources** having at least one element as light_source

aim305: **camera_model_d3_with_hlhsr** with **hidden_line_surface_removal** = true

aim344: light_source_positional with **position** as cartesian_point

aim337: light_source with **light_colour** as colour

aim339: light_source as **light_source_positional**

aim341: light_source as **light_source_ambient**

aim417: surface_style_rendering as **surface_style_rendering_with_properties**

aim412: surface_style_rendering with **surface_colour** as colour

aim416: surface_style_rendering with **rendering_method** = constant_shading

aim418: surface_style_rendering_with_properties with **properties** having at least one element as surface_style_transparent

aim419: surface_style_rendering_with_properties with **properties** having at least one element as surface_style_reflectance_ambient

aim391: surface_rendering_properties with **rendered_colour** as colour

aim328: curve_style_rendering with **rendering_properties** as surface_rendering_properties

aim330: curve_style_rendering with **rendering_method** = constant_colour

aim429: view_volume with **view_window** as planar_box

aim430: view_volume with **view_volume_sides_clipping** = true

aim432: view_volume with **back_plane_clipping** = true

aim434: view_volume with **back_plane_distance** as length_measure

aim435: view_volume with **front_plane_clipping** = true

aim437: view_volume with **front_plane_distance** as length_measure

aim438: view_volume with **view_plane_distance** as length_measure

aim439: view_volume with **projection_point** as cartesian_point

aim440: view_volume with **projection_type** = parallel

aim???: planar_box with **placement** as axis2_placement

ae511: Point_appearance using default marker_size

6.6.1.2 Postprocessor

AIM test purpose coverage:

See specific verdict criteria for the preprocessor model.

Input specification:

```
TEST_CASE vp1 ; WITH part_204_brep_product_schema ;
  REALIZATION
    LOCAL
      light_settings : SET [1:?] OF light_source ;
      curve_method   : shading_curve_method ;
      surface_method : shading_surface_method ;
      camera_info    : camera_model ;
      rendering_info : surface_style_rendering ;
      curve_style    : curve_or_render ;

      prod_name : STRING := 'prod_name';
      shape_def : product_definition_shape ;
      shape_def_rep : shape_definition_representation ;
      shell_object : closed_shell ;
      tetrahedron  : faceted_brep ;
      fbsr : faceted_brep_shape_representation ;
      its_units : named_unit ;
      prod_context : representation_context ;

      psc : presentation_style_by_context ;
      rm  : representation_map ;
    END_LOCAL ;

    CALL set_curve_style_rendering ;
      IMPORT (curve_style := @curve_style; ) ;
      WITH   (curve_method := constant_colour; ) ;
    END_CALL ;

    CALL set_ambient_style_rendering ;
      IMPORT (rendering_info := @surface_style; ) ;
      WITH   (transparency := 1.0;
              surface_method := constant_shading; ) ;
    END_CALL ;

    CALL set_ambience ;
      IMPORT (light_settings[1] := @ambient_light_source;) ;
    END_CALL ;
```

```
    CALL set_positional_light ;
      IMPORT (light_settings[2] := @positional_light_source;) ;
      WITH   (colour_name := 'red'; x := 100; y := 150; z := 200; ) ;
    END_CALL ;

    CALL set_camera_model_d3_shading ;
      IMPORT (camera_info := @camera_model; ) ;
      WITH   (lights := @light_settings; ) ;
    END_CALL ;

    CALL set_presentation_styles ;
      IMPORT (all_styles := @presentation_styles;) ;
      WITH   (curve_style := @curve_style;) ;
    END_CALL ;

    CALL basic_product_structure ;
      IMPORT (shape_def := @prod_def_shape; );
      WITH   (prod_name := @prod_name; );
    END_CALL;

    CALL tetrashell_instance ;
      IMPORT (shell_object := @tetrashell; ) ;
    END_CALL;

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

    prod_context := geometric_representation_context
                    ('context_1', 'context_for_tetrahedron', 3) ||
                    global_unit_assigned_context ( [its_units] ) ;

    tetrahedron := faceted_brep ('tetrahedron', shell_object) ;

   fbsr := faceted_brep_shape_representation
                ('fbsr',  [tetrahedron], its_context );

   shape_def_rep := shape_definition_representation (shape_def, fbsr);

   psc := presentation_style_by_context ( all_styles, fbsr ) ;

   rm := representation_map (camera_info, fbsr) ;

  END_REALIZATION ;
END_TEST_CASE ;
```

Specific verdict criteria:

See general verdict criteria in clause 5 and the specific test purposes for the preprocessor.

6.6.2 Test case vp2

Test case summary:

This test case presents a model with a directional light-source and some ambient light, with colour shading for each facet. Each curve is rendered with linear colour.

6.6.2.1 Preprocessor

Input specification:

See table 35.

NOTE 1 See Test Case ps3 for details of product and part. See Test Case fb1 for details of shape representation. See Test Case vp4 for rendering and viewing values.

Table 35 – Preprocessor details: test case vp2

Id	V	Application Element	Value	Req
@502.curve_appearance	*	curve_appearance	#curve_style	S
@503.curve_appearance.colour	*	curve_appearance	#curve_style.colour	S
@506.point_appearance	*	point_appearance	#point_style	S
@508.point_appearance.colour	*	point_appearance	#point_style.colour	S
@509.point_appearance.marker	*	point_appearance	#point_style.marker	S
@510.point_appearance.marker_size	*	point_appearance	#point_style.marker_size	S
@514.surface_appearance	*	surface_appearance	#surface_style.styles	S
@515.surface_appearance	*	surface_appearance	#surface_style.styles.rendering	S

Specific verdict criteria:

aim323: curve_style_font_and_scaling with **curve_font** as pre_defined_curve_font

aim325: curve_style_font_and_scaling with **name** as label

aim405: surface_style_parameter_line with **direction_counts** having at least one element as v_direction_count

aim406: surface_style_parameter_line with **direction_counts** having at least one element as u_direction_count

aim408: surface_style_parameter_line with **style_of_parameter_lines** as curve_style

aim403: surface_style_control_grid with **style_of_control_grid** as curve_style

aim421: **surface_style_segmentation_curve** with **style_of_segmentation_curve** as curve_style

aim423: **surface_style_silhouette** with **style_of_silhouette** as curve_style

aim401: **surface_style_boundary** with **style_of_boundary** as curve_style

aim404: **surface_style_fill_area** with **fill_area** as fill_area_style

aim393: **surface_side_style** with **styles** having at least one element as surface_style_parameter_line

aim394: **surface_side_style** with **styles** having at least one element as surface_style_control_grid

aim395: **surface_side_style** with **styles** having at least one element as surface_style_segmentation_-curve

aim396: **surface_side_style** with **styles** having at least one element as surface_style_silhouette

aim397: **surface_side_style** with **styles** having at least one element as surface_style_boundary

aim398: **surface_side_style** with **styles** having at least one element as surface_style_fill_area

aim399: **surface_side_style** with **name** as label

aim425: **surface_style_usage** with **style** as surface_side_style

aim426: **surface_style_usage** with **side** = both

aim335: **fill_area_style_colour** with **fill_colour** as colour

aim336: **fill_area_style_colour** with **name** as label

aim333: **fill_area_style** with **fill_styles** having at least one element as fill_area_style_colour

aim334: **fill_area_style** with **name** as label

aim314: **curve_style** with **curve_colour** as colour

aim316: **curve_style** with **curve_width** as positive_length_measure

aim319: **curve_style** with **curve_font** as predefined_curve_font

aim320: **curve_style** with **name** as label

aim358: **point_style** with **marker_colour** as colour

aim360: **point_style** with **marker_size** as positive_length_measure

aim361: point_style with **marker** as marker_type (= dot, x, plus, asterisk, ring, square, triangle)

aim368: point_style with **name** as label

aim386: presentation_style_by_context with **style_context** as representation

aim381: presentation_style_assignment with **styles** having at least one element as surface_style_-usage

aim382: presentation_style_assignment with **styles** having at least one element as curve_style

aim383: presentation_style_assignment with **styles** having at least one element as point_style

aim388: styled_item with **item** as representation_item

aim389: styled_item with **styles** having at least one element as presentation_style_assignment

aim343: light_source_directional with **orientation** as direction

aim340: light_source as **light_source_directional**

aim415: surface_style_rendering with **rendering_method** = colour_shading

aim329: curve_style_rendering with **rendering_method** = linear_colour

ae502: Curve_appearance as curve_style

ae503: Curve_appearance with colour as colour_rgb

ae506: Point_appearance as point_style

ae508: Point_appearance with colour

ae509: Point_appearance with marker

ae510: Point_appearance with marker_size

ae514: Surface_appearance with grid_indicator

ae515: Surface_appearance with shading_method

6.6.2.2 Postprocessor

AIM test purpose coverage:

See specific verdict criteria for the preprocessor model.

Input specification:

```
TEST_CASE vp2 ; WITH part_204_brep_product_schema ;
  REALIZATION
    LOCAL
      light_settings : SET [1:?] OF light_source ;
      surface_method : shading_surface_method ;
      camera_info    : camera_model ;
      rendering_info : surface_style_rendering ;

      prod_name : STRING := 'prod_name';
      shape_def : product_definition_shape ;
      shape_def_rep : shape_definition_representation ;
      shell_object : closed_shell ;
      tetrahedron  : faceted_brep ;
      fbsr : faceted_brep_shape_representation ;
      its_units : named_unit ;
      prod_context : representation_context ;

      point_style   : point_style ;
      curve_style   : curve_style ;
      surface_style : surface_style_usage ;
      all_styles    : SET [1:?] OF presentation_style_select ;

      psc : presentation_style_by_context ;
      rm  : representation_map ;
    END_LOCAL ;

    CALL set_ambient_style_rendering ;
      IMPORT (rendering_info := @surface_style; ) ;
      WITH   (transparency := 1.0;
              surface_method := colour_shading; ) ;
    END_CALL ;

    CALL set_ambience ;
      IMPORT (light_settings[1] := @ambient_light_source;) ;
    END_CALL ;

    CALL set_directional_light ;
      IMPORT (light_settings[2] := @directional_light_source;) ;
      WITH   (colour_name := 'red'; ) ;
    END_CALL ;

    CALL set_camera_model_d3_shading ;
      IMPORT (camera_info := @camera_model; ) ;
      WITH   (lights := @light_settings; ) ;
```

```
END_CALL ;

CALL set_point_style ;
  IMPORT (point_style := @point_style;) ;
END_CALL ;

CALL set_curve_style ;
  IMPORT (curve_style := @curve_style;) ;
END_CALL ;

CALL set_rendered_surface_style ;
  IMPORT (surface_style := @surface_style;) ;
  WITH (rendering := @rendering_info;) ;
END_CALL ;

CALL set_presentation_styles ;
  IMPORT (all_styles := @presentation_styles;);
  WITH (point_style := @point_style; curve_style := @curve_style;
        surface_style := @surface_style ;) ;
END_CALL ;

CALL basic_product_structure ;
  IMPORT (shape_def := @prod_def_shape; );
  WITH   (prod_name := @prod_name; );
END_CALL;

CALL tetrashell_instance ;
  IMPORT (shell_object := @tetrashell; ) ;
END_CALL;

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

prod_context := geometric_representation_context
                 (’context_1’, ’context_for_tetrahedron’, 3) ||
                 global_unit_assigned_context ( [its_units] ) ;

tetrahedron := faceted_brep (’tetrahedron’, shell_object) ;

fbsr := faceted_brep_shape_representation
            (’fbsr’,  [tetrahedron], its_context );

shape_def_rep := shape_definition_representation (shape_def, fbsr);

psc := presentation_style_by_context ( all_styles, fbsr ) ;

rm := representation_map (camera_info, fbsr) ;
```

```
  END_REALIZATION ;
END_TEST_CASE ;
```

Specific verdict criteria:

See general verdict criteria in clause 5 and the specific test purposes for the preprocessor.

6.6.3 Test case vp3

Test case summary:

This test case presents a model with a spot light-source and some ambient light, with dot shading for each facet and diffuse reflectance.

6.6.3.1 Preprocessor

Input specification:

See table 36.

NOTE 1 See Test Case ps3 for details of product and part. See Test Case fb1 for details of shape representation. See Test Case vp4 for rendering and viewing values.

Table 36 – Preprocessor details: test case vp3

Id	V	Application Element	Value	Req
@501.camera_info	*	3d_projection	#camera_info	S
@519.camera_info	*	3d_projection to B-rep	#rm	M
@522.fbsr	*	3d_projection to B-rep	#rm	M
@512	*	geometry with presentation	#psc	M
@513	*	geometry with presentation	#psc	M
@526	*	geometry with presentation	#psc	M
@530	*	geometry with presentation	#psc	M
@534	*	geometry with presentation	#psc	M
@537	*	geometry with presentation	#psc	M
@542	*	geometry with presentation	#psc	M
@544	*	geometry with presentation	#psc	M
@547	*	geometry with presentation	#psc	M
@551	*	geometry with presentation	#psc	M
@555	*	geometry with presentation	#psc	M

Specific verdict criteria:

aim345: light_source_spot with **spread_angle** as positive_plane_angle_measure

aim346: light_source_spot with **orientation** as direction

aim347: light_source_spot with **position** as cartesian_point

aim338: light_source as **light_source_spot**

aim414: surface_style_rendering with **rendering_method** = dot_shading

aim409: surface_style_reflectance_ambient as **surface_style_reflectance_ambient_diffuse**

6.6.3.2 Postprocessor

AIM test purpose coverage:

See specific verdict criteria for the preprocessor model.

Input specification:

```
TEST_CASE vp3 ; WITH part_204_brep_product_schema ;
  REALIZATION
    LOCAL
      light_settings : SET [1:?] OF light_source ;
      surface_method : shading_surface_method ;
      camera_info    : camera_model ;
      rendering_info : surface_style_rendering ;

      prod_name : STRING := 'prod_name';
      shape_def : product_definition_shape ;
      shape_def_rep : shape_definition_representation ;
      shell_object : closed_shell ;
      tetrahedron  : faceted_brep ;
      fbsr : faceted_brep_shape_representation ;
      its_units : named_unit ;
      prod_context : representation_context ;

      point_style   : point_style ;
      curve_style   : curve_style ;
      surface_style : surface_style_usage ;
      all_styles    : SET [1:?] OF presentation_style_select ;

      psc : presentation_style_by_context ;
      rm  : representation_map ;
    END_LOCAL ;

    CALL set_diffuse_style_rendering ;
      IMPORT (rendering_info := @surface_style; ) ;
```

```
  WITH (surface_method := dot_shading) ;
END_CALL ;

CALL set_ambience ;
  IMPORT (light_settings[1] := @ambient_light_source;) ;
END_CALL ;

CALL set_spotlight ;
  IMPORT (light_settings[2] := @spot_light_source;) ;
END_CALL ;

CALL set_camera_model_d3_shading ;
  IMPORT (camera_info := @camera_model; ) ;
  WITH   (lights := @light_settings; ) ;
END_CALL ;

CALL set_point_style ;
  IMPORT (point_style := @point_style;) ;
END_CALL ;

CALL set_curve_style ;
  IMPORT (curve_style := @curve_style;) ;
END_CALL ;

CALL set_rendered_surface_style ;
  IMPORT (surface_style := @surface_style;) ;
  WITH (rendering := @rendering_info;) ;
END_CALL ;

CALL set_presentation_styles ;
  IMPORT (all_styles := @presentation_styles;);
  WITH (point_style := @point_style; curve_style := @curve_style;
        surface_style := @surface_style ;) ;
END_CALL ;

CALL basic_product_structure ;
  IMPORT (shape_def := @prod_def_shape; );
  WITH   (prod_name := @prod_name; );
END_CALL;

CALL tetrashell_instance ;
  IMPORT (shell_object := @tetrashell; ) ;
END_CALL;

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

```
    prod_context := geometric_representation_context
                    ('context_1', 'context_for_tetrahedron', 3) ||
                    global_unit_assigned_context ( [its_units] ) ;

    tetrahedron := faceted_brep ('tetrahedron', shell_object) ;

    fbsr := faceted_brep_shape_representation
                ('fbsr',  [tetrahedron], its_context );

    shape_def_rep := shape_definition_representation(shape_def, fbsr);

    psc := presentation_style_by_context ( all_styles, fbsr ) ;

    rm := representation_map (camera_info, fbsr) ;

  END_REALIZATION ;
END_TEST_CASE ;
```

Specific verdict criteria:

See general verdict criteria in clause 5 and the specific test purposes for the preprocessor.

6.6.4 Test case vp4

Test case summary:

This test case is similar to vp3, except that normal shading is used with a collection of light sources and specular reflectance.

6.6.4.1 Preprocessor

Input specification:

See table 37.

NOTE 1 See Test Case ps3 for details of product and part. See Test Case fb1 for details of shape representation.

Specific verdict criteria:

aim413: surface_style_rendering with **rendering_method** = normal_shading

aim410: surface_style_reflectance_ambient_diffuse as surface_style_reflectance_ambient_diffuse_specular

aim411: surface_style_reflectance_ambient_diffuse_specular with **specular_colour** as colour

Table 37 – Preprocessor details: test case vp4

Id	V	Application Element	Value	Req
@501.camera_info	*	3d_projection	#camera_info	S
@519.camera_info	*	3d_projection to B-rep	#rm	M
@522.fbsr	*	3d_projection to B-rep	#rm	M
@512	*	geometry with presentation	#psc	M
@513	*	geometry with presentation	#psc	M
@526	*	geometry with presentation	#psc	M
@530	*	geometry with presentation	#psc	M
@534	*	geometry with presentation	#psc	M
@537	*	geometry with presentation	#psc	M
@542	*	geometry with presentation	#psc	M
@544	*	geometry with presentation	#psc	M
@547	*	geometry with presentation	#psc	M
@551	*	geometry with presentation	#psc	M
@555	*	geometry with presentation	#psc	M

ae501: 3D_projection as camera_model_3d

ae512: Surface_appearance as surface_style_usage

ae513: Surface_appearance with colour

ae519: 3D_projection presenting one B-rep model

ae522: B-rep model associated with one 3D_projection

ae526: Curve_appearance associated with many curves

ae530: Curve_appearance associated with many edges

ae534: Point_appearance associated with many points

ae537: Presentation_appearance associated with one B-rep

ae542: Presentation_appearance associated with many shells

ae544: Shell presented as many presentation_appearances

ae547: Surface_appearance associated with many faces

ae551: Surface_appearance associated with many surfaces

ae555: Topological_representation_item associated with many presentation attributes

6.6.4.2 Postprocessor

AIM test purpose coverage:

See specific verdict criteria for the preprocessor model.

Input specification:

```
TEST_CASE vp4 ; WITH part_204_brep_product_schema ;
  REALIZATION
    LOCAL
      light_settings : SET [1:?] OF light_source ;
      camera_info    : camera_model ;
      rendering_info : surface_style_rendering ;

      prod_name : STRING := 'prod_name';
      shape_def : product_definition_shape ;
      shape_def_rep : shape_definition_representation ;
      shell_object : closed_shell ;
      tetrahedron  : faceted_brep ;
      fbsr : faceted_brep_shape_representation ;
      its_units : named_unit ;
      prod_context : representation_context ;

      point_style   : point_style ;
      curve_style   : curve_style ;
      surface_style : surface_style_usage ;
      all_styles    : SET [1:?] OF presentation_style_select ;

      psc : presentation_style_by_context ;
      rm  : representation_map ;
    END_LOCAL ;

    CALL set_specular_style_rendering ;
      IMPORT (rendering_info := @surface_style; ) ;
    END_CALL ;

    CALL set_ambience ;
      IMPORT (light_settings[1] := @ambient_light_source;) ;
    END_CALL ;

    CALL set_spotlight ;
      IMPORT (light_settings[2] := @spot_light_source;) ;
    END_CALL ;

    CALL set_directional_light ;
```

```
  IMPORT (light_settings[3] := @directional_light_source;) ;
  WITH   (colour_name := 'red'; ) ;
END_CALL ;

CALL set_positional_light ;
  IMPORT (light_settings[4] := @positional_light_source;) ;
  WITH   (colour_name := 'red'; x := 100; y := 150; z := 200; ) ;
END_CALL ;

CALL set_camera_model_d3_shading ;
  IMPORT (camera_info := @camera_model; ) ;
  WITH   (lights := @light_settings; ) ;
END_CALL ;

CALL set_point_style ;
  IMPORT (point_style := @point_style;) ;
END_CALL ;

CALL set_curve_style ;
  IMPORT (curve_style := @curve_style;) ;
END_CALL ;

CALL set_rendered_surface_style ;
  IMPORT (surface_style := @surface_style;) ;
  WITH (rendering := @rendering_info;) ;
END_CALL ;

CALL set_presentation_styles ;
  IMPORT (all_styles := @presentation_styles;);
  WITH (point_style := @point_style; curve_style := @curve_style;
        surface_style := @surface_style ;) ;
END_CALL ;

CALL basic_product_structure ;
  IMPORT (shape_def := @prod_def_shape; );
  WITH   (prod_name := @prod_name; );
END_CALL;

CALL tetrashell_instance ;
  IMPORT (shell_object := @tetrashell; ) ;
END_CALL;

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

prod_context := geometric_representation_context
                 ('context_1', 'context_for_tetrahedron', 3) ||
```

```
                    global_unit_assigned_context ( [its_units] ) ;

    tetrahedron := faceted_brep ('tetrahedron', shell_object) ;

    fbsr := faceted_brep_shape_representation
                ('fbsr',  [tetrahedron], its_context );

   shape_def_rep := shape_definition_representation (shape_def, fbsr);

   psc := presentation_style_by_context ( all_styles, fbsr ) ;

   rm := representation_map (camera_info, fbsr) ;

  END_REALIZATION ;
END_TEST_CASE ;
```

Specific verdict criteria:

See general verdict criteria in clause 5 and the specific test purposes for the preprocessor.

6.6.5 Test case vp5

Test case summary:

This test case is used to test the absence of clipping and hidden line and hidden surface removal, with a user-defined curve font. [3)]

6.6.5.1 Preprocessor

Input specification:

See table 38.

NOTE 1 See Test Case ps3 for details of product and part. See Test Case fb1 for details of shape representation. See Test Case vp3 for rendering and viewing values.

Specific verdict criteria:

aim324: curve_style_font_and_scaling with **curve_font** as curve_style_font

aim326: curve_style_font_pattern with **invisible_segment_length** as positive_length_measure

aim327: curve_style_font_pattern with **visible_segment_length** as positive_length_measure

aim321: curve_style_font with **pattern_list** having at least one element as curve_style_font_pattern

3) invisibility could be tested here (via styled_item)

Table 38 – Preprocessor details: test case vp5

Id	V	Application Element	Value	Req
@504.curve_style	*	curve_appearance	#curve_style	S
@505.curve_style	*	curve_appearance	#curve_style	S
@518	*	3d_projection to B-rep	#unused_camera_info	M
@521	*	B-rep to 3d_projection	#unpresented_fbsr	M
@527	*	curve to curve_appearance	#unpresented_fbsr	M
@531	*	edge to curve_appearance	#unpresented_fbsr	M
@535	*	point to point_appearance	#unpresented_fbsr	M
@539	*	B-rep to presentation_appearance	#unpresented_fbsr	M
@543	*	shell to presentation_appearance	#unpresented_fbsr	M
@548	*	face to surface_appearance	#unpresented_fbsr	M
@552	*	surface to surface_appearance	#unpresented_fbsr	M
@553	*	top_rep_item to presentation_appearance	#unpresented_fbsr	M
@524	*	curve_appearance to curve	#unallocated_styles	M
@528	*	curve_appearance to edge	#unallocated_styles	M
@532	*	point_appearance to point	#unallocated_styles	M
@536	*	presentation_appearance to B-rep	#unallocated_styles	M
@540	*	presentation_appearance to shell	#unallocated_styles	M
@545	*	surface_appearance to face	#unallocated_styles	M
@549	*	surface_appearance to surface	#unallocated_styles	M

aim322: **curve_style_font** with **name** as label

aim318: **curve_style** with **curve_font** as curve_style_font

aim431: **view_volume** with **view_volume_sides_clipping** = false

aim433: **view_volume** with **back_plane_clipping** = false

aim436: **view_volume** with **front_plane_clipping** = false

aim441: **view_volume** with **projection_type** = central

aim306: **camera_model_d3_with_hlhsr** with **hidden_line_surface_removal** = false

ae504: Curve_appearance with curve_font

ae505: Curve_appearance with curve_width

ae518: 3D_projection presenting no B-rep models

ae521: B-rep model associated with no 3D_projections

ae524: Curve_appearance associated with no curves

ae527: Curve represented as zero curve_appearances

ae528: Curve_appearance associated with no edges

ae531: Edge represented as zero curve_appearances

ae532: Point_appearance associated with zero points

ae535: Point presented as zero point_appearances

ae536: Presentation_appearance associated with zero B-reps

ae539: B-rep model presented as zero presentation_appearances

ae540: Presentation_appearance associated with zero shells

ae543: Shell presented as zero presentation_appearances

ae545: Surface_appearance associated with zero faces

ae548: Face presented as zero surface_appearances

ae549: Surface_appearance associated with zero surfaces

ae552: Surface presented as zero surface_appearances

ae553: Topological_representation_item associated with no presentation attributes

6.6.5.2 Postprocessor

AIM test purpose coverage:

See specific verdict criteria for the preprocessor model.

Input specification:

```
TEST_CASE vp5 ; WITH part_204_brep_product_schema ;
  REALIZATION
    LOCAL
      light_settings : SET [1:?] OF light_source ;
      surface_method : shading_surface_method ;
      camera_info    : camera_model ;
      unused_camera_info : camera_model ;
      rendering_info : surface_style_rendering ;
```

```
  prod_name : STRING := 'prod_name';
  shape_def : product_definition_shape ;
  shape_def_rep : shape_definition_representation ;
  shell_object : closed_shell ;
  tetrahedron  : faceted_brep ;
  fbsr : faceted_brep_shape_representation ;
  unpresented_fbsr : faceted_brep_shape_representation ;
  base : plane ;
  its_units : named_unit ;
  prod_context : representation_context ;

  point_style   : point_style ;
  curve_style   : curve_style ;
  surface_style : surface_style_usage ;
  all_styles    : SET [1:?] OF presentation_style_select ;
  similar_styles     : SET [1:?] OF presentation_style_select ;
  unallocated_styles : SET [1:?] OF presentation_style_select ;

  psc : presentation_style_by_context ;
  psc_special  : presentation_style_by_context ;
  rm  : representation_map ;
END_LOCAL ;

CALL set_diffuse_style_rendering ;
  IMPORT (rendering_info := @surface_style; ) ;
  WITH (surface_method := dot_shading) ;
END_CALL ;

CALL set_ambience ;
  IMPORT (light_settings[1] := @ambient_light_source;) ;
END_CALL ;

CALL set_spotlight ;
  IMPORT (light_settings[2] := @spot_light_source;) ;
END_CALL ;

CALL set_camera_model_d3_shading ;
  IMPORT (camera_info := @camera_model;) ;
  WITH   (lights := @light_settings;
          hlhsr := FALSE ;
          front_clipping := FALSE ;
          back_clipping := FALSE ;
          sides_clipping := FALSE ;
         );
END_CALL;
```

```
CALL set_camera_model_d3_shading ;
(*  set up another (unused) camera model  *)
  IMPORT (unused_camera_info := @camera_model;) ;
  WITH   (lights := @light_settings;
          ppx := 30; ppy := 20; ppz := 40;
         );
END_CALL;

CALL set_point_style ;
  IMPORT (point_style := @point_style;) ;
END_CALL ;

CALL set_user_defined_curve_style ;
  IMPORT (curve_style := @curve_style;) ;
  WITH (pattern := [0.0, 1.0, 0.0, 2.0, 0.0, 1.0, 0.0, 4.0];) ;
                    (* short   medium   short     long *)
END_CALL ;

CALL set_rendered_surface_style ;
  IMPORT (surface_style := @surface_style;) ;
  WITH (rendering := @rendering_info;) ;
END_CALL ;

CALL set_presentation_styles ;
  IMPORT (all_styles := @presentation_styles;);
  WITH (point_style := @point_style; curve_style := @curve_style;
        surface_style := @surface_style ;) ;
END_CALL ;

CALL set_presentation_styles ;
(*  set up an unused set of presentation styles *)
  IMPORT (unallocated_styles := @presentation_styles;);
  WITH (point_style := @point_style; curve_style := @curve_style;
        surface_style := @surface_style ;) ;
END_CALL ;

CALL set_rendered_surface_style ; -- now change the surface style
  IMPORT (surface_style := @surface_style;) ;
  WITH (rendering := @rendering_info;
        u_iso_lines := 10;
        v_iso_lines := 3;) ;
END_CALL ;

CALL set_presentation_styles ;
(* set up a second set of presentation styles *)
```

```
      IMPORT (similar_styles := @presentation_styles;);
      WITH (point_style := @point_style; curve_style := @curve_style;
            surface_style := @surface_style ;) ;
    END_CALL ;

    CALL basic_product_structure ;
      IMPORT (shape_def := @prod_def_shape; );
      WITH   (prod_name := @prod_name; );
    END_CALL;

    CALL tetrashell_instance ;
      IMPORT (shell_object := @tetrashell;
              base := @p4; ) ;
(*  retrieve the base for special presentation *)
    END_CALL;

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

    prod_context := geometric_representation_context
                    (’context_1’, ’context_for_tetrahedron’, 3) ||
                    global_unit_assigned_context ( [its_units] ) ;

    tetrahedron := faceted_brep (’tetrahedron’, shell_object) ;

    fbsr := faceted_brep_shape_representation
                (’fbsr’,  [tetrahedron], its_context );

    shape_def_rep := shape_definition_representation(shape_def, fbsr);

    psc := presentation_style_by_context ( all_styles, fbsr ) ;

    rm := representation_map (camera_info, fbsr) ;

    psc_special := presentation_style_by_context ( similar_styles,
                                                   base);

    unpresented_fbsr := faceted_brep_shape_representation
                (’fbsr’,  [tetrahedron], its_context );

  END_REALIZATION ;
END_TEST_CASE ;
```

Specific verdict criteria:

See general verdict criteria in clause 5 and the specific test purposes for the preprocessor.

6.6.6 Test case vp6

Test case summary:

This test case is similar to vp4, except that the rendering is used to present an elementary B-rep model rather than a faceted B-rep.

6.6.6.1 Preprocessor

Input specification:

See table 39.

NOTE 1 See Test Case ps3 for details of product and part. See Test Case eb1 for details of shape representation.

Table 39 – Preprocessor details: test case vp6

Id	V	Application Element	Value	Req
@501.camera_info	*	3d_projection	#camera_info	S
@519.camera_info	*	3d_projection to B-rep	#rm	M
@522.ebsr	*	3d_projection to B-rep	#rm	M
@512	*	geometry with presentation	#psc	M
@513	*	geometry with presentation	#psc	M
@526	*	geometry with presentation	#psc	M
@530	*	geometry with presentation	#psc	M
@534	*	geometry with presentation	#psc	M
@537	*	geometry with presentation	#psc	M
@542	*	geometry with presentation	#psc	M
@544	*	geometry with presentation	#psc	M
@547	*	geometry with presentation	#psc	M
@551	*	geometry with presentation	#psc	M
@555	*	geometry with presentation	#psc	M

6.6.6.2 Postprocessor

Input specification:

```
TEST_CASE vp6 ; WITH part_204_brep_product_schema ;
  REALIZATION
    LOCAL
      light_settings : SET [1:?] OF light_source ;
      camera_info    : camera_model ;
      rendering_info : surface_style_rendering ;

      prod_name : STRING := 'prod_name';
```

```
  shape_def : product_definition_shape ;
  shape_def_rep : shape_definition_representation ;
  shell_object : closed_shell ;
  cysp_solid  : manifold_solid_brep ;
  ebsr : elementary_brep_shape_representation ;
  its_units : named_unit ;
  prod_context : representation_context ;

  point_style   : point_style ;
  curve_style   : curve_style ;
  surface_style : surface_style_usage ;
  all_styles    : SET [1:?] OF presentation_style_select ;

  psc : presentation_style_by_context ;
  rm  : representation_map ;
END_LOCAL ;

CALL set_specular_style_rendering ;
  IMPORT (rendering_info := @surface_style; ) ;
END_CALL ;

CALL set_ambience ;
  IMPORT (light_settings[1] := @ambient_light_source;) ;
END_CALL ;

CALL set_spotlight ;
  IMPORT (light_settings[2] := @spot_light_source;) ;
END_CALL ;

CALL set_directional_light ;
  IMPORT (light_settings[3] := @directional_light_source;) ;
  WITH   (colour_name := 'red'; ) ;
END_CALL ;

CALL set_positional_light ;
  IMPORT (light_settings[4] := @positional_light_source;) ;
  WITH   (colour_name := 'red'; x := 100; y := 150; z := 200; ) ;
END_CALL ;

CALL set_camera_model_d3_shading ;
  IMPORT (camera_info := @camera_model; ) ;
  WITH   (lights := @light_settings; ) ;
END_CALL ;

CALL set_point_style ;
  IMPORT (point_style := @point_style;) ;
```

```
    END_CALL ;

    CALL set_curve_style ;
      IMPORT (curve_style := @curve_style;) ;
    END_CALL ;

    CALL set_rendered_surface_style ;
      IMPORT (surface_style := @surface_style;) ;
      WITH (rendering := @rendering_info;) ;
    END_CALL ;

    CALL set_presentation_styles ;
      IMPORT (all_styles := @presentation_styles;);
      WITH (point_style := @point_style; curve_style := @curve_style;
            surface_style := @surface_style ;) ;
    END_CALL ;

    CALL basic_product_structure ;
      IMPORT (shape_def := @prod_def_shape; );
      WITH   (prod_name := @prod_name; );
    END_CALL;

    CALL cylinder_sphere_shell ;
      IMPORT (shell_object := @cyspshell; ) ;
    END_CALL;

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

    prod_context := geometric_representation_context
                    ('context_1', 'context_for_tetrahedron', 3) ||
                    global_unit_assigned_context ( [its_units] ) ;

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

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

    shape_def_rep := shape_definition_representation(shape_def, ebsr);

    psc := presentation_style_by_context ( all_styles, ebsr ) ;

    rm := representation_map (camera_info, fbsr) ;

  END_REALIZATION ;
END_TEST_CASE ;
```

6.6.7 Test case vp7

Test case summary:

This test case is similar to vp4, except that the rendering is used to present an advanced B-rep model rather than a faceted B-rep.

6.6.7.1 Preprocessor

Input specification:

See table 40.

NOTE 1 See Test Case ps3 for details of product and part. See Test Case ab1 for details of shape representation.

Table 40 – Preprocessor details: test case vp7

Id	V	Application Element	Value	Req
@501.camera_info	*	3d_projection	#camera_info	S
@519.camera_info	*	3d_projection to B-rep	#rm	M
@522.absr	*	3d_projection to B-rep	#rm	M
@512	*	geometry with presentation	#psc	M
@513	*	geometry with presentation	#psc	M
@526	*	geometry with presentation	#psc	M
@530	*	geometry with presentation	#psc	M
@534	*	geometry with presentation	#psc	M
@537	*	geometry with presentation	#psc	M
@542	*	geometry with presentation	#psc	M
@544	*	geometry with presentation	#psc	M
@547	*	geometry with presentation	#psc	M
@551	*	geometry with presentation	#psc	M
@555	*	geometry with presentation	#psc	M

6.6.7.2 Postprocessor

Input specification:

```
TEST_CASE vp7 ; WITH part_204_brep_product_schema ;
  REALIZATION
    LOCAL
      light_settings : SET [1:?] OF light_source ;
      camera_info    : camera_model ;
      rendering_info : surface_style_rendering ;

      prod_name : STRING := 'prod_name';
```

```
  shape_def : product_definition_shape ;
  shape_def_rep : shape_definition_representation ;
  face_1, face_2, face_3 : advanced_face;
  shell_object : closed_shell ;
  cysp_solid  : manifold_solid_brep ;
  absr : advanced_brep_shape_representation ;
  its_units : named_unit ;
  prod_context : representation_context ;

  point_style   : point_style ;
  curve_style   : curve_style ;
  surface_style : surface_style_usage ;
  all_styles    : SET [1:?] OF presentation_style_select ;

  psc : presentation_style_by_context ;
  rm  : representation_map ;
END_LOCAL ;

CALL set_specular_style_rendering ;
  IMPORT (rendering_info := @surface_style; ) ;
END_CALL ;

CALL set_ambience ;
  IMPORT (light_settings[1] := @ambient_light_source;) ;
END_CALL ;

CALL set_spotlight ;
  IMPORT (light_settings[2] := @spot_light_source;) ;
END_CALL ;

CALL set_directional_light ;
  IMPORT (light_settings[3] := @directional_light_source;) ;
  WITH   (colour_name := 'red'; ) ;
END_CALL ;

CALL set_positional_light ;
  IMPORT (light_settings[4] := @positional_light_source;) ;
  WITH   (colour_name := 'red'; x := 100; y := 150; z := 200; ) ;
END_CALL ;

CALL set_camera_model_d3_shading ;
  IMPORT (camera_info := @camera_model; ) ;
  WITH   (lights := @light_settings; ) ;
END_CALL ;

CALL set_point_style ;
```

```
  IMPORT (point_style := @point_style;) ;
END_CALL ;

CALL set_curve_style ;
  IMPORT (curve_style := @curve_style;) ;
END_CALL ;

CALL set_rendered_surface_style ;
  IMPORT (surface_style := @surface_style;) ;
  WITH (rendering := @rendering_info;) ;
END_CALL ;

CALL set_presentation_styles ;
  IMPORT (all_styles := @presentation_styles;);
  WITH (point_style := @point_style; curve_style := @curve_style;
        surface_style := @surface_style ;) ;
END_CALL ;

CALL basic_product_structure ;
  IMPORT (shape_def := @prod_def_shape; );
  WITH   (prod_name := @prod_name; );
END_CALL;

CALL cylinder_sphere_faces ;
  IMPORT (face_1 := @top_face;
          face_2 := @curved_face;
          face_3 :=  @bottom_face; ) ;
END_CALL;

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

prod_context := geometric_representation_context
                ('context_1', 'context_for_tetrahedron', 3) ||
                global_unit_assigned_context ( [its_units] ) ;

shell_object := closed_shell('shell_object', [face_1, face_2,
                                                face_3] );
cysp_solid := manifold_solid_brep ('cysp_solid', shell_object) ;

absr := advanced_brep_shape_representation
            ('absr',  [cysp_solid], its_context );

shape_def_rep := shape_definition_representation(shape_def, absr);

psc := presentation_style_by_context ( all_styles, absr ) ;
```

```
    rm := representation_map (camera_info, fbsr) ;

  END_REALIZATION ;
END_TEST_CASE ;
```

Annex A
(normative)

Conformance classes

A.1 Conformance class 1

To conform to conformance class 1 of ISO 10303-204, an implementation must pass executable versions of the following abstract test cases:

— fb1

— fb2

— fb3

— fb4

— fb5

— ps3

A.2 Conformance class 2

To conform to conformance class 2 of ISO 10303-204, an implementation must pass executable versions of the following abstract test cases:

— eb1

— eb2

— eb3

— eb4

— eb5

— eb6

— ps2

A.3 Conformance class 3

To conform to conformance class 3 of ISO 10303-204, an implementation must pass executable versions of the following abstract test cases:

— ab1

— ab2

— ab3

— ab4

— ab5

— ab6

— ab7

— ab8

— ab9

— ab10

— ab11

— ab12

— ab13

— ab14

— ab15

— ps1

— ps4

A.4 Optional test cases

Table A.1 – Optional test cases

Class 1	fb6, vp1, vp2, vp3, vp4, vp5
Class 2	vp6
Class 3	np1, vp7

Annex B
(normative)

Information object registration

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

{ iso standard 10303 part(304) 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.

Annex C
(normative)

Contexts for test case definitions

This annex defines EXPRESS-I contexts which are used in test case definitions. In documenting the PARAMETER block the EXPRESS entity constructor is used to define the expression for default values. This was done to avoid the creation and instantiation of full sub-type trees as currently required in EXPRESS-I instances.

C.1 Basic Product Structure context

The context below provides the minimal entity set required in all implementations of Part 204. It is derived from the product structure UoF. It provides an application protocol definition and an application context in which a B-rep product model may be defined. The application context is mechanical and the definition context is design. This EXPRESS-I context provides the basic structure which enables the shape representation defined by a B-rep model to be linked to a product. This link is achieved by an external **shape_definition_representation** which references the **product_definition_shape** (prod_def_shape) entity of this context.

```
*)
CONTEXT basic_product_structure ;

PARAMETER
  applic    : STRING := 'AP 204 test';
  apyear    : INTEGER := 1996 ;
  apstatus  : STRING  := 'dIS' ;
  prod_id   : STRING  := 'P01' ;
  pdf_id    : STRING  := 'PDF01' ;
  pdf_desc  : STRING  := 'test version 1';
  pdef_desc : STRING  := 'test product definition 1' ;
  prod_name : STRING  := 'test_product' ;
  propd_name : STRING  := 'test_shape_property' ;
  propd_desc : STRING := 'shape property of test part 1' ;
  prod_desc : STRING  := 'simple test product' ;


END_PARAMETER;

SCHEMA_DATA basic_prod_struc;

ap_def = application_protocol_definition { status -> @apstatus ;
        application_interpreted_model_schema_name ->
                                  'part_204_brep_product_shema';
                        application_protocol_year -> @apyear ;
```

```
                                        application -> @apctxt ; } ;

apctxt = application_context {application -> @applic; } ;

apctxp = application_context_element { name -> 'pt_204_product' ;
                         frame_of_reference ->  @apctxt ;
                                  SUPOF(@prodctxt) ; } ;

apctxpd = application_context_element { name -> 'pt_204_prod_def' ;
                          frame_of_reference ->  @apctxt ;
                                  SUPOF(@pdefctxt) ; } ;

prodctxt = product_context { SUBOF(@apctxp ;
                            discipline_type -> 'mechanical';
                                     SUPOF(@mechctx);} ;

pdefctxt = product_definition_context { SUBOF(@apctxpd ;
                       life_cycle_stage     -> 'design';
                                     SUPOF(@desctx);} ;

mechctx  = mechanical_context { SUBOF(@prodctxt) ; } ;

desctx  =  design_context { SUBOF(@pdefctxt) ; } ;


testprod   =  product { id -> @prod_id ;
                      name -> @prod_name ;
                description -> @prod_desc ;
        frame_of_reference -> [@prodctxt] ; } ;

prod_d_f   =  product_definition_formation { id -> @pdf_id ;
                                   description -> @pdf_desc ;
                                   of_product  -> @testprod ;    } ;

prod_def   =  product_definition { id -> @prod_def_id ;
                                   description -> @pdef_desc ;
                                    formation  -> @prod_d_f ;
                              frame_of_reference -> @pdefctxt ;  } ;

prop_def  =  property_definition { name -> @propd_name ;
                              description -> @propd_desc ;
                              definition  -> @prod_def ;
                                SUPOF{ @prod_def_shape ; } ;
prod_def_shape = product_definition_shape { SUBOF(@prop_def) ; } ;

END_SCHEMA_DATA;
```

```
END_CONTEXT ;
(*
```

C.2 Contexts defined for test cases of faceted B-rep

The EXPRESS-I context below is used in the test cases in clause 6.1.

C.2.1 Tetrashell_instance context

This context provides the facility to define a simple **closed_shell** of tetrahedral shape with vertices at (orc,orc,orc), (lx,orc,orc), (orc,ly,orc) and (orc,orc,lz). All bounds are defined by **poly_loop**s.

```
*)
CONTEXT tetrashell_instance ;

PARAMETER
  orc       : length_measure := 0;
  lx        : length_measure := 100;
  ly        : length_measure := 100;
  lz        : length_measure := 100;

  origin    : cartesian_point := cartesian_point ('origin', [orc,
                                                        orc, orc]);
  p_x       : cartesian_point := cartesian_point ('p_x',
                                                  [lx, orc, orc]);
  p_y       : cartesian_point := cartesian_point ('p_y', [orc, ly,
                                                           orc]);
  p_z       : cartesian_point := cartesian_point ('p_z',
                                                   [orc, orc, lz]);

  neg_x     : direction := direction ('neg_x', [-1, 0, 0]);
  neg_y     : direction := direction ('neg_y', [0, -1, 0]);
  neg_z     : direction := direction ('neg_z', [0, 0, -1]);
  dslope    : direction := direction ('dslope', [1, 1, 1]);
  dperp     : direction := direction ('dperp', [1, -1, 0]);

  loop_x    : poly_loop := poly_loop ('loop_x', [origin, p_z, p_y]) ;
  loop_y    : poly_loop := poly_loop ('loop_y', [origin, p_x, p_z]) ;
  loop_z    : poly_loop := poly_loop ('loop_z', [origin, p_y, p_x]) ;
 loop_slope : poly_loop := poly_loop ('loop_slope', [p_z, p_x, p_y]) ;

  a1        : axis2_placement_3d := axis2_placement_3d ('a1', origin,
                                                        neg_x, neg_y) ;
  a2        : axis2_placement_3d := axis2_placement_3d ('a2', origin,
```

```
                                                        neg_y, neg_x) ;
  a3        : axis2_placement_3d := axis2_placement_3d ('a3', origin,
                                                        neg_z, neg_y) ;
  a4        : axis2_placement_3d := axis2_placement_3d
                          ('a4', p_x, dslope, dperp) ;

  p1        : plane := plane ('p1', a1) ;
  p2        : plane := plane ('p2', a2) ;
  p3        : plane := plane ('p3', a3) ;
  p4        : plane := plane ('p4', a4) ;

  b1   : face_outer_bound := face_outer_bound ('b1', loop_x, TRUE) ;
  b2   : face_outer_bound := face_outer_bound ('b2', loop_y, TRUE) ;
  b3   : face_outer_bound := face_outer_bound ('b3', loop_z, TRUE) ;
  b4   : face_outer_bound := face_outer_bound('b4', loop_slope, TRUE);

  fs1 : face_surface := face_surface ('fs1', [b1], p1, TRUE) ;
  fs2 : face_surface := face_surface ('fs2', [b2], p2, TRUE) ;
  fs3 : face_surface := face_surface ('fs3', [b3], p3, TRUE) ;
  fs4 : face_surface := face_surface ('fs4', [b4], p4, TRUE) ;
END_PARAMETER;

SCHEMA_DATA tetra_ctxt;

cfs = connected_face_set {SUBOF(@tri); cfs_faces ->
                              ([@fs1, @fs2, @fs3, @fs4]);
                           SUPOF(@tetrashell);} ;

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

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

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

END_SCHEMA_DATA;

END_CONTEXT ;
(*
```

C.3 Contexts defined for test cases of elementary B-rep

The EXPRESS-I contexts below are used in the test cases in clause 6.2.

C.3.1 Cylinder_sphere_shell context

This context provides the faces needed to define a simple **closed_shell** of cylindrical shape with hemi-spherical 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 ;
(*
```

C.3.2 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. (note: **plane_angle_units** required to be degrees) The normal to each plane face is orthogonal to y axis direction. and is at a fixed angle.

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, cirular 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 ;

(*
```

C.3.3 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 ;

(*
```

C.3.4 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 ;

(*
```

C.4 Contexts defined for test cases of advanced B-rep

C.4.1 Cylinder_sphere_faces context

This context provides the faces needed to define a simple closed shell of cylindrical shape with hemi-spherical base centred at (orc,orc,orc), radius rad and axial height h to oblique face.

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

```
*)
CONTEXT cylinder_sphere_faces ;
  WITH aic_topology_bounded_surface;
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 : advanced_face := advanced_face ('curved_face',
                                    [bcylbot, bcyltop], cyl, TRUE );
 top_face : advanced_face := advanced_face('top_face', [be], pl, TRUE);
 bottom_face : advanced_face := advanced_face ('bottom_face',
                                              [bc], sphere, TRUE );
END_PARAMETER;

SCHEMA_DATA cyl_sph_ctxt;

CONSTANT
  rt2 == sqrt(2.0);
  rt3 == sqrt(3.0);
END_CONSTANT;

END_SCHEMA_DATA;

END_CONTEXT ;
(*
```

C.4.2 Cone_faces 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. (note: **plane_angle_unit**s required to be degrees) The normal to each plane face is orthogonal to y axis direction. and is at a fixed angle.

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, cirular base and hyperbolic and parabolic sides. Base has 2 straight edges parallel to y axis.

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

```
*)
CONTEXT cone_faces ;
  WITH aic_topology_bounded_surface;
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  : advanced_face := advanced_face ('curved_face',
                  [bcone, be_f], cone, TRUE );
  tope_face    : advanced_face := advanced_face ('tope_face',
                                              [be_t], ple, TRUE );
  bottomc_face : advanced_face := advanced_face
                       ('bottomc_face', [bbase],  plc, FALSE );
  par_face     : advanced_face :=
                      advanced_face ('par_face', [bpar], plp, TRUE );
  hyp_face     : advanced_face :=
                      advanced_face ('hyp_face', [bhyp], plh, TRUE );

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

SCHEMA_DATA con_fac_ctxt;

CONSTANT
  rt2 == sqrt(2.0);
  rt3 == sqrt(3.0);
END_CONSTANT;

END_SCHEMA_DATA;

END_CONTEXT ;
(*
```

C.4.3 Toroidal_segment_advanced context

This context provides the **advanced_face**s 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_loops**. Basic dimensional parameters should not be varied.

```
*)
CONTEXT toroidal_segment_advanced ;
  WITH aic_topology_bounded_surface;
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 : advanced_face := advanced_face('curved_face', [btop],
                                        torus, TRUE );
base_face   : advanced_face :=
```

```
                advanced_face (’base_face’, [bbase], base, FALSE );
  left_face : advanced_face :=
                advanced_face (’left_face’, [bleft],  pleft, TRUE);
  right_face : advanced_face :=
               advanced_face (’right_face’, [bright],  pright, TRUE );
END_PARAMETER;

SCHEMA_DATA tor_seg_adv_ctxt;

END_SCHEMA_DATA;

END_CONTEXT ;

(*
```

C.4.4 Cylinder_union_polyline_advanced 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_loops. Basic dimensional parameters should not be varied.

```
*)
CONTEXT cylinder_union_polyline_advanced;
  WITH aic_topology_bounded_surface;
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  : advanced_face := advanced_face ('cyl_face1',
                                [bcy1top, bcy1m, bcy1b], cyl1, TRUE );
  cyl_face2  : advanced_face := advanced_face ('cyl_face2',
                  [bcy2m, bcy2r], cyl2, TRUE );
 base_face   : advanced_face :=
                   advanced_face ('base_face', [bbase], base, FALSE );
 top_face    : advanced_face :=
                     advanced_face ('top_face', [btop],  top, TRUE );
 right_face : advanced_face :=
              advanced_face ('right_face', [bright],  plright, TRUE );
END_PARAMETER;

SCHEMA_DATA cyl_un_poly_adv_ctxt;

END_SCHEMA_DATA;

END_CONTEXT ;

(*
```

C.4.5 Cylinder_union_b_spline 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 closed **b_spline_curve** and of a **face** with 3 bounding loops.

Apart from the spline curve, which should with a tolerance of less than 0.001, lie on both intersecting cylinders this context is geometrically identical to the **cylinder_union_polyline** context.

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

```
*)

CONTEXT cylinder_union_b_spline;
  WITH aic_topology_bounded_surface;
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', [20.0, 50.0, 0.0]);
p2 : cartesian_point :=
        cartesian_point('p2', [20.0, 50.0, 5.7119866]);
p3 : cartesian_point :=
        cartesian_point('p3', [18.1895525, 49.2890498, 11.4181902]);
p4 : cartesian_point :=
        cartesian_point('p4', [14.6964609, 47.8055303, 16.5303827]);
p5 : cartesian_point :=
        cartesian_point('p5', [ 9.2828663, 46.1587661, 20.3115492]);
p6 : cartesian_point :=
        cartesian_point('p6', [ 3.5830226, 44.3603883, 22.4220938]);
p7 : cartesian_point :=
       cartesian_point('p7', [ -3.5830226, 44.3603883, 22.4220938]);
```

```
  p8 : cartesian_point :=
         cartesian_point('p8', [ -9.2828663, 46.1587661, 20.3115492]);
  p9 : cartesian_point :=
         cartesian_point('p9', [-14.6964609, 47.8055303, 16.5303827]);
  p10 : cartesian_point :=
      cartesian_point('p10', [-18.1895525, 49.2890498, 11.4181902]);
  p11 : cartesian_point :=
          cartesian_point('p11', [-20.0, 50.0, 5.71198664]);
  p12 : cartesian_point :=
          cartesian_point('p12', [-20.0, 50.0, 0.0]);
  p13 : cartesian_point :=
          cartesian_point('p13', [-20.0, 50.0, -5.71198664]);
  p14 : cartesian_point :=
       cartesian_point('p14', [-18.1895525, 49.2890498, 11.4181902]);
  p15 : cartesian_point :=
       cartesian_point('p15', [-14.6964609, 47.8055303, -16.530383]);
  p16 : cartesian_point :=
       cartesian_point('p16', [ -9.2828663, 46.1587661, -20.311549]);
  p17 : cartesian_point :=
        cartesian_point('p17', [-3.5830226, 44.3603883, -22.422094]);
  p18 : cartesian_point :=
         cartesian_point('p18', [3.5830226, 44.3603883, -22.422094]);
  p19 : cartesian_point :=
         cartesian_point('p19', [9.2828662, 46.1587661, -20.311549]);
  p20 : cartesian_point :=
         cartesian_point('p20', [14.6964609, 47.8055303, -16.530383]);
  p21 : cartesian_point :=
         cartesian_point('p21', [18.1895525, 49.2890498, -11.41819 ]);
  p22  : cartesian_point :=
            cartesian_point('p22', [20.0, 50.0, -5.7119866]);

(* spline is a closed Bezier curve of degree 11 *)
  spline : bezier_curve := bezier_curve('spline',  11,
                [p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12,
                p13, p14, p15, p16, p17, p18, p19, p20, p21, p22, p1],
                 !unspecified, TRUE, FALSE);

  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, spline, 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  : advanced_face := advanced_face (’cyl_face1’,
                   [bcy1top, bcy1m, bcy1b], cyl1, TRUE );
 cyl_face2 : advanced_face := advanced_face (’cyl_face2’,
                              [bcy2m, bcy2r], cyl2, TRUE );
 base_face   : advanced_face :=
                  advanced_face (’base_face’, [bbase], base, FALSE );
 top_face    : advanced_face :=
                       advanced_face (’top_face’, [btop],  top, TRUE );
  right_face : advanced_face :=
               advanced_face (’right_face’, [bright],  plright, TRUE);
END_PARAMETER;

SCHEMA_DATA cyl_un_bspl_ctxt;

END_SCHEMA_DATA;
```

```
END_CONTEXT ;

(*
```

C.4.6 Swept_faces context

This context provides the faces needed to define a **closed_shell** with **swept_surface** geometry. A rational B-spline curve is used to define the curve to be swept. The curved faces are a blend of a surface of revolution and a surface of linear translation. The closed shell is completed by planar faces.

Parameter orc can be varied to re-locate origin and dimensions l10, l20, l30, l40 re-defined to change scale of figure.

```
*)
CONTEXT swept_faces;
  WITH aic_topology_bounded_surface;
PARAMETER
  orc      : length_measure := 0.0;
  l10      : length_measure := 10.0;
  l20      : length_measure := 20.0;
  l30      : length_measure := 30.0;
  l40      : length_measure := 40.0;
  trans    : length_measure := -50.0;
  ort      : length_measure := orc + trans;
  w1       : REAL := 1.0;
  w2       : REAL := 2.0;

  origin   : cartesian_point := cartesian_point ('origin',
                                                 [orc, orc, orc]);
  p1  : cartesian_point := cartesian_point ('p1',
                                 [l10 + orc, orc, orc]);
  p2  : cartesian_point := cartesian_point ('p2',
                                      [l30 + orc, l10 + orc, orc]);
  p3  : cartesian_point := cartesian_point ('p3',
                                      [l40 + orc, l20 + orc, orc]);
  p4  : cartesian_point := cartesian_point ('p4',
                                      [l20 + orc, l30 + orc, orc]);
  pt1 : cartesian_point := cartesian_point ('pt1',
                                        [l10 + orc, orc, ort]);
  pt2 : cartesian_point := cartesian_point ('pt2',
                                       [l30 + orc, l10 + orc, ort]);
  pt3 : cartesian_point := cartesian_point ('pt3',
                                       [l40 + orc, l20 + orc, ort]);
  pt4 : cartesian_point := cartesian_point ('pt4',
                                       [l20 + orc, l30 + orc, ort]);
```

```
  pr1 : cartesian_point := cartesian_point (’pr1’,
                                            [orc, orc, l10 + orc]);
  pr2 : cartesian_point := cartesian_point (’pr2’,
                                       [orc, l10 + orc, l30 + orc]);
  pr3 : cartesian_point := cartesian_point (’pr3’,
                                       [orc, l20 + orc, l40 + orc]);
  pr4 : cartesian_point := cartesian_point (’pr4’,
                                        [orc, l30 + orc, l20 + orc]);

  pback   : cartesian_point := cartesian_point (’pback’,
                                             [orc, l30 + orc, orc]);
(* base points *)
  pb1  : cartesian_point := cartesian_point (’pb1’, [orc , orc, ort]);
  pb2      : cartesian_point := cartesian_point (’pb2’,
                                             [orc, l30 + orc, ort]);
  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]);
  vec_x    : vector := vector (’vec_x’,  pos_x, 1.0 ) ;
  vec_y    : vector := vector (’vec_y’,  pos_y, 1.0 ) ;
  vec_z    : vector := vector (’vec_z’,  pos_z, 1.0 ) ;

  a1  : axis2_placement_3d := axis2_placement_3d (’a1’, origin,
                                                  pos_y, pos_z);
  a2  : axis2_placement_3d := axis2_placement_3d (’a2’, pback,
                                                  pos_y, pos_z);
  a3  : axis2_placement_3d := axis2_placement_3d (’a3’, pb1,
                                                  pos_z, pos_x);
  a4  : axis2_placement_3d := axis2_placement_3d (’a4’, origin,
                                                  pos_x, pos_y);
 arot : axis1_placement := axis1_placement (’arot’, origin, pos_y) ;

(* rational B-spline curve of degree 3 +
                   translated and rotated copies *)
  bspl  : b_spline_curve := bezier_curve(’bspl’,  3, [p1, p2,p3, p4],
                             !unspecified, FALSE, FALSE ) ||
                          rational_b_spline_curve ( [w1, w2, w2, w1]);
  bsplt : b_spline_curve := bezier_curve(’bsplt’,  3,
          [pt1, pt2, pt3, pt4], !unspecified, FALSE, FALSE ) ||
                          rational_b_spline_curve ( [w1, w2, w2, w1]);
  bsplr  : b_spline_curve := bezier_curve(’bsplr’,  3,
        [pr1, pr2, pr3, pr4], !unspecified, FALSE, FALSE ) ||
                           rational_b_spline_curve ([w1, w2, w2, w1]);

(* swept surfaces + planes *)
   srot  : surface_of_revolution := surface_of_revolution (’srot’,
```

```
                                                    bspl, arot) ;
   stran : surface_of_linear_extrusion :=
                           surface_of_linear_extrusion ('stran',
                                                    bspl, vec_z);
   plbase   : plane := plane ('plbase', a3) ;
   plback   : plane := plane ('plback', a2) ;
   plfront  : plane := plane ('plfront', a1) ;
   plside   : plane := plane ('plside', a4) ;
 (* circles generated by rotating end points of curve *)
  circ1     : circle := circle ('circ1', a1, l10) ;
  circ2     : circle := circle ('circ2', a2, l20) ;

(* bounding lines *)
  lnb1x     : line := line ('lnb1x',  pb1, vec_x) ;
  lnb1y     : line := line ('lnb1y',  pb1, vec_y) ;
  lnb1z     : line := line ('lnb1z',  pb1, vec_z) ;
  lnp1z     : line := line ('lnp1z',  p1, vec_z) ;
  lnp4z     : line := line ('lnp4z',  p4, vec_z) ;
  lnb2x     : line := line ('lnb2x',  pb2, vec_x) ;
  lnb2z     : line := line ('lnb2z',  pb2, vec_z) ;

 (* vertices and edges *)
   v1   : vertex_point := vertex_point ('v1', p1);
   vt1  : vertex_point := vertex_point ('vt1', pt1);
   vr1  : vertex_point := vertex_point ('vr1', pr1);
   v4   : vertex_point := vertex_point ('v4', p4);
   vt4  : vertex_point := vertex_point ('vt4', pt4);
   vr4  : vertex_point := vertex_point ('vr4', pr4);
   vb1  : vertex_point := vertex_point ('vb1', pb1);
   vb2  : vertex_point := vertex_point ('vb2', pb2);

  es    : edge_curve := edge_curve ('es',  v1, v4, bspl, TRUE) ;
  est   : edge_curve := edge_curve ('est',  vt1, vt4, bsplt, TRUE) ;
  esr   : edge_curve := edge_curve ('esr',  vr1, vr4, bsplr, TRUE) ;
  ec1   : edge_curve := edge_curve ('ec1',   v1, vr1, circ1, TRUE) ;
  ec2   : edge_curve := edge_curve ('ec2',   v4, vr4, circ2, TRUE) ;
  eb1x  : edge_curve := edge_curve ('eb1x',  vb1, vt1, lnb1x, TRUE) ;
  eb1y  : edge_curve := edge_curve ('eb1y',  vb1, vb2, lnb1y, TRUE) ;
  eb1z  : edge_curve := edge_curve ('eb1z',  vb1, pr1, lnb1z, TRUE) ;
  eb2x  : edge_curve := edge_curve ('eb2x',  vb2, vt4, lnb2x, TRUE) ;
  eb2z  : edge_curve := edge_curve ('eb2z',  vb2, vr4, lnb2z, TRUE) ;
  et1z  : edge_curve := edge_curve ('et1z',  vt1, v1, lnp1z, TRUE) ;
  et4z  : edge_curve := edge_curve ('et4z',  pt4, v4, lnp4z, TRUE) ;

(* oriented_edges *)
  es_t      : oriented_edge := oriented_edge ('es_t', es, TRUE) ;
```

```
  es_f      : oriented_edge := oriented_edge ('es_f', es, FALSE) ;
  est_t     : oriented_edge := oriented_edge ('est_t', est, TRUE) ;
  est_f     : oriented_edge := oriented_edge ('est_f', est, FALSE) ;
  esr_t     : oriented_edge := oriented_edge ('esr_t', esr, TRUE) ;
  esr_f     : oriented_edge := oriented_edge ('esr_f', esr, FALSE) ;
  ec1_t     : oriented_edge := oriented_edge ('ec1_t', ec1, TRUE) ;
  ec1_f     : oriented_edge := oriented_edge ('ec1_f', ec1, FALSE) ;
  ec2_t     : oriented_edge := oriented_edge ('ec2_t', ec2, TRUE) ;
  ec2_f     : oriented_edge := oriented_edge ('ec2_f', ec2, FALSE) ;
  eb1x_t    : oriented_edge := oriented_edge ('eb1x_t', eb1x, TRUE) ;
  eb1x_f    : oriented_edge := oriented_edge ('eb1x_f', eb1x, FALSE) ;
  eb1y_t    : oriented_edge := oriented_edge ('eb1y_t', eb1y, TRUE) ;
  eb1y_f    : oriented_edge := oriented_edge ('eb1y_f', eb1y, FALSE) ;
  eb1z_t    : oriented_edge := oriented_edge ('eb1z_t', eb1z, TRUE) ;
  eb1z_f    : oriented_edge := oriented_edge ('eb1z_f', eb1z, FALSE) ;
  eb2x_t    : oriented_edge := oriented_edge ('eb2x_t', eb2x, TRUE) ;
  eb2x_f    : oriented_edge := oriented_edge ('eb2x_f', eb2x, FALSE) ;
  eb2z_t    : oriented_edge := oriented_edge ('eb2z_t', eb2z, TRUE) ;
  eb2z_f    : oriented_edge := oriented_edge ('eb2z_f', eb2z, FALSE) ;
  et1z_t    : oriented_edge := oriented_edge ('et1z_t', et1z, TRUE) ;
  et1z_f    : oriented_edge := oriented_edge ('et1z_f', et1z, FALSE) ;
  et4z_t    : oriented_edge := oriented_edge ('et4z_t', et4z, TRUE) ;
  et4z_f    : oriented_edge := oriented_edge ('et4z_f', et4z, FALSE) ;

(* face bounding loops  *)
  elrot     : edge_loop := edge_loop ('elrot',
                                      [es_t, ec2_t, esr_f, ec1_f]);
  eltran    : edge_loop := edge_loop ('eltran',
                                    [est_t, et4z_t, es_f, et1z_f]) ;
  elbase    : edge_loop := edge_loop ('elbase',
                                   [est_f, eb1x_f, eb1y_t, eb2x_t]) ;
  elfront   : edge_loop := edge_loop ('elfront',
                                   [eb1x_t, et1z_t, ec1_t, eb1z_f]) ;
  elback    : edge_loop := edge_loop ('elback',
                                   [eb2z_t, ec2_f, et4z_f, eb2x_f]) ;
  elside    : edge_loop := edge_loop ('elside',
                                    [er_t, eb2z_f, eb1y_f, eb1z_t]) ;


 fbrot  : face_outer_bound := face_outer_bound('fbrot', elrot, TRUE);
 fbtran : face_outer_bound := face_outer_bound('fbtran', eltran, TRUE);
 fbbase : face_outer_bound := face_outer_bound('fbbase', elbase, TRUE);
 fbfront : face_outer_bound := face_outer_bound ('fbfront',
                                                 elfront, TRUE) ;
 fbback : face_outer_bound := face_outer_bound('fbback', elback, TRUE);
 fbside : face_outer_bound := face_outer_bound('fbside', elside, TRUE);
```

```
(* faces *)
  afrot   : advanced_face :=
                advanced_face (’afrot’, [fbrot],  srot, TRUE );
  aftran  : advanced_face :=
                advanced_face (’aftran’, [fbtran], stran, TRUE );
  afbase  : advanced_face :=
              advanced_face (’afbase’, [fbbase], plbase, FALSE );
  affront : advanced_face :=
              advanced_face (’affront’, [fbfront], plfront, FALSE );
  afback  : advanced_face :=
                advanced_face (’afback’, [fbback],  plback, TRUE );
  afside  : advanced_face :=
                advanced_face (’afside’, [fbside],  plside, FALSE );
END_PARAMETER;

SCHEMA_DATA swept_fac_ctxt;

END_SCHEMA_DATA;

END_CONTEXT ;

(*
```

C.4.7 B-spline Pyramid context

This context provides the faces needed to define a **closed_shell** in the shape of an inverted pyramid with vertex at the origin, 4 planar faces and one curved face defined by a B-spline surface. Edge curves are lines or B-spline curves of degree 3. Basic parameters should not be changed, but interior control points of surface (P11, P12, P21, P22) may be varied to alter surface shape.

```
*)
CONTEXT b_spline_pyramid_faces;
  WITH aic_topology_bounded_surface;
PARAMETER
  orc      : length_measure := 0.0;

  origin   : cartesian_point := cartesian_point ([orc, orc, orc]);

(* control points for B-spline surface and for edge curves *)
  p00  : cartesian_point := cartesian_point (’p00’,
                                        [-50.0, -50.0, 50.0]);
  p10  : cartesian_point := cartesian_point (’p10’,
                                          [-30.0, -60.0, 60.0]);
  p20  : cartesian_point := cartesian_point (’p20’,
                                             [20.0, -70.0, 70.0]);
```

```
  p30  : cartesian_point := cartesian_point (’p30’,
                                              [50.0, -50.0, 50.0]);
  p01  : cartesian_point := cartesian_point (’p01’,
                                             [-60.0, -10.0, 60.0]);
  p11  : cartesian_point := cartesian_point (’p11’,
                                             [-20.0, -20.0, 65.0]);
  p21  : cartesian_point := cartesian_point (’p21’,
                                              [30.0, -15.0, 80.0]);
  p31  : cartesian_point := cartesian_point (’p31’,
                                              [65.0, -15.0, 65.0]);
  p02  : cartesian_point := cartesian_point (’p02’,
                                              [-65.0, 20.0, 65.0]);
  p12  : cartesian_point := cartesian_point (’p12’,
                                              [-20.0, 30.0, 60.0]);
  p22  : cartesian_point := cartesian_point (’p22’,
                                               [25.0, 25.0, 85.0]);
  p32  : cartesian_point := cartesian_point(’p32’, [70.0, 15.0, 70.0]);
  p03  : cartesian_point := cartesian_point (’p03’,
                                              [-50.0, 50.0, 50.0]);
  p13  : cartesian_point := cartesian_point (’p13’,
                                              [-25.0, 70.0, 70.0]);
  p23  : cartesian_point := cartesian_point(’p23’, [15.0, 75.0, 75.0]);
  p33  : cartesian_point := cartesian_point(’p33’, [50.0, 50.0, 50.0]);

(* normals to planar faces *)
  nfac1    : direction := direction (’nfac1’, [-1, 0, -1]);
  nfac2    : direction := direction (’nfac2’, [0, -1, -1]);
  nfac3    : direction := direction (’nfac3’, [1, 0, -1]);
  nfac4    : direction := direction (’nfac4’, [0, 1, -1]);

(* directions of edge lines *)
  dl00     : direction := direction (’dl00’, [-1,-1, 1]);
  dl30     : direction := direction (’dl30’, [-1, 1, 1]);
  dl03     : direction := direction (’dl03’, [1, -1, 1]);
  dl33     : direction := direction (’dl33’, [1, 1, 1]);

  vec00    : vector := vector(’vec00’,  dl00, 1.0 ) ;
  vec30    : vector := vector (’vec30’,  dl30, 1.0 ) ;
  vec03    : vector := vector (’vec03’,  dl03, 1.0 ) ;
  vec33    : vector := vector (’vec33’,  dl33, 1.0 ) ;

(* lines from origin *)
  lnp00    : line := line (’lnp00’,  origin, vec00) ;
  lnp30    : line := line (’lnp30’,  origin, vec30) ;
  lnp03    : line := line (’lnp03’,  origin, vec03) ;
  lnp33    : line := line (’lnp33’,  origin, vec33) ;
```

```
(* axis placements for planes *)
  a1  : axis2_placement_3d := axis2_placement_3d ('a1', origin,
                                                   nfac1, dl00) ;
  a2  : axis2_placement_3d := axis2_placement_3d ('a2', origin,
                                                   nfac2, dl30) ;
  a3  : axis2_placement_3d := axis2_placement_3d ('a3', origin,
                                                   nfac3, dl33) ;
  a4  : axis2_placement_3d := axis2_placement_3d ('a4', origin,
                                                  nfac4, dl03) ;

(* Planes for planar faces *)
   pl1     : plane := plane ('pl1', a1) ;
   pl2     : plane := plane ('pl2', a2) ;
   pl3     : plane := plane ('pl3', a3) ;
   pl4     : plane := plane ('pl4', a4) ;

(* B-spline curves of degree 3 for curved edges *)
  bsplc1   : bezier_curve := bezier_curve('bsplc1',  3,
               [p00, p01,p02, p03], !unspecified, FALSE, FALSE ) ;

  bsplc2   : bezier_curve := bezier_curve('bsplc2',  3,
              [p00, p10, p20, p30], !unspecified, FALSE, FALSE ) ;
  bsplc3   : bezier_curve := bezier_curve('bsplc3',  3,
             [p30, p31, p32, p33], !unspecified, FALSE, FALSE ) ;
  bsplc4   : bezier_curve := bezier_curve('bsplc4',  3,
              [p03, p13, p23, p33], !unspecified, UNKNOWN, UNKNOWN ) ;

(* B-spline surface of degree 3 for top face *)

   bsplsurf : bezier_surface := bezier_surface ('bsplsurf',  3, 3 ,
                                [ [p00, p01, p02, p03],
                                  [p10, p11, p12, p13],
                                  [p20, p21, p22, p23],
                                  [p30, p31, p32, p33] ],
                          !unspecified, FALSE, FALSE, FALSE ) ;


 (* vertices and edges *)
   apex  : vertex_point := vertex_point ('apex', origin);
   v00   : vertex_point := vertex_point ('v00', p00);
   v03   : vertex_point := vertex_point ('v03', p03);
   v30   : vertex_point := vertex_point ('v30', p30);
   v33   : vertex_point := vertex_point ('v33', p33);

  el00   : edge_curve := edge_curve('el00',  apex , v00, lnp00, TRUE );
```

```
  el03   : edge_curve := edge_curve('el03',  apex , v03, lnp03, TRUE );
  el30   : edge_curve := edge_curve('el30',  apex , v30, lnp30, TRUE );
  el33   : edge_curve := edge_curve('el33',  apex , v33, lnp33, TRUE );
  ec1    : edge_curve := edge_curve('ec1',  v00, v03, bsplc1, TRUE );
  ec2    : edge_curve := edge_curve('ec2',  v00, v30, bsplc2, TRUE );
  ec3    : edge_curve := edge_curve('ec3',  v30, v33, bsplc3, TRUE );
  ec4    : edge_curve := edge_curve('ec4',  v03, v33, bsplc1, TRUE );

(* oriented_edges *)
  el00_t  : oriented_edge := oriented_edge ('el00_t', el00, TRUE) ;
  el00_f  : oriented_edge := oriented_edge ('el00_f', el00, FALSE) ;
  el03_t  : oriented_edge := oriented_edge ('el03_t', el03, TRUE) ;
  el03_f  : oriented_edge := oriented_edge ('el03_f', el03, FALSE) ;
  el30_t  : oriented_edge := oriented_edge ('el30_t', el30, TRUE) ;
  el30_f  : oriented_edge := oriented_edge ('el30_f', el30, FALSE) ;
  el33_t  : oriented_edge := oriented_edge ('el33_t', el33, TRUE) ;
  el33_f  : oriented_edge := oriented_edge ('el33_f', el33, FALSE) ;
  ec1_t   : oriented_edge := oriented_edge ('ec1_t', ec1, TRUE) ;
  ec1_f   : oriented_edge := oriented_edge ('ec1_f', ec1, FALSE) ;
  ec2_t   : oriented_edge := oriented_edge ('ec2_t', ec2, TRUE) ;
  ec2_f   : oriented_edge := oriented_edge ('ec2_f', ec2, FALSE) ;
  ec3_t   : oriented_edge := oriented_edge ('ec3_t', ec3, TRUE) ;
  ec3_f   : oriented_edge := oriented_edge ('ec3_f', ec3, FALSE) ;
  ec4_t   : oriented_edge := oriented_edge ('ec4_t', ec4, TRUE) ;
  ec4_f   : oriented_edge := oriented_edge ('ec4_f', ec4, FALSE) ;

(* face bounding loops  *)
  elleft  : edge_loop := edge_loop('elleft', [el00_t, ec1_t, el03_f]);
  elright : edge_loop := edge_loop('elright', [el33_t, ec3_f, el30_f]);
  elfront : edge_loop := edge_loop('elfront', [el30_t, ec2_f, el00_f]);
  elback  : edge_loop := edge_loop('elback', [el03_t, ec4_t, el33_f]);
  eltop   : edge_loop := edge_loop('eltop',
                                   [ec1_f, ec2_t, ec3_t, ec4_f]);

  fbleft  : face_outer_bound := face_outer_bound ('fbleft', elleft,
                                                  TRUE) ;
  fbright : face_outer_bound := face_outer_bound ('fbright', elright,
                                                  TRUE) ;
  fbfront : face_outer_bound := face_outer_bound ('fbfront', elfront,
                                                  TRUE) ;
  fbback : face_outer_bound := face_outer_bound ('fbback', elback,
                                                  TRUE) ;
  fbtop : face_outer_bound := face_outer_bound('fbtop', eltop, TRUE) ;

(* faces *)
  afleft    : advanced_face :=
```

```
                    advanced_face ('afleft', [fbleft],  pl1, TRUE );
  afright   : advanced_face :=
                    advanced_face ('afright', [fbright], pl3, TRUE );
  affront   : advanced_face :=
                   advanced_face ('affront', [fbfront], pl2, FALSE );
  afback    : advanced_face :=
                    advanced_face ('afback', [fbback],  pl4, TRUE );
  aftop     : advanced_face :=
                   advanced_face ('aftop', [fbtop],  bsplsurf, TRUE );

END_PARAMETER ;

SCHEMA_DATA bspl_pyr_ctxt ;

END_SCHEMA_DATA;

END_CONTEXT ;

(*
```

C.4.8 Rational B-spline Pyramid context

This context provides the faces needed to define a **closed_shell** in the shape of an inverted pyramid with vertex at the origin, 4 planar feces and one curved face defined by a rational B-spline surface. Edge curves are lines or rational B-spline curves of degree 3.

Basic parameters should not be changed, but interior control points of surface (P11, P12, P21, P22) may be varied to alter surface shape. Surface and edge curve shape may also be modified by altering the values of the weight parameters (w1, w2, w3).

NOTE 1 If the weight parameters are altered to $w1 = w2 = w3 = 1.0$, the surface and edge geometry should be identical to that in the b_spline pyramid context.

```
*)
CONTEXT rational_b_spline_pyramid_faces;
  WITH aic_topology_bounded_surface;
PARAMETER
  orc       : length_measure := 0.0;
  w1        : REAL := 1.0;
  w2        : REAL := 2.0;
  w3        : REAL := 3.0;

  origin    : cartesian_point := cartesian_point ('origin',
                                                  [orc, orc, orc]);
```

```
(* control points for B-spline surface and for edge curves *)
 p00 : cartesian_point := cartesian_point('p00', [-50.0, -50.0, 50.0]);
 p10 : cartesian_point := cartesian_point('p10', [-30.0, -60.0, 60.0]);
 p20 : cartesian_point := cartesian_point('p20', [20.0, -70.0, 70.0]);
 p30 : cartesian_point := cartesian_point('p30', [50.0, -50.0, 50.0]);
 p01 : cartesian_point := cartesian_point('p01', [-60.0, -10.0, 60.0]);
 p11 : cartesian_point := cartesian_point('p11', [-20.0, -20.0, 65.0]);
 p21 : cartesian_point := cartesian_point('p21', [30.0, -15.0, 80.0]);
 p31 : cartesian_point := cartesian_point('p31', [65.0, -15.0, 65.0]);
 p02 : cartesian_point := cartesian_point('p02', [-65.0, 20.0, 65.0]);
 p12 : cartesian_point := cartesian_point('p12', [-20.0, 30.0, 60.0]);
 p22 : cartesian_point := cartesian_point('p22', [25.0, 25.0, 85.0]);
 p32 : cartesian_point := cartesian_point('p32', [70.0, 15.0, 70.0]);
 p03 : cartesian_point := cartesian_point('p03', [-50.0, 50.0, 50.0]);
 p13 : cartesian_point := cartesian_point('p13', [-25.0, 70.0, 70.0]);
 p23 : cartesian_point := cartesian_point('p23', [15.0, 75.0, 75.0]);
 p33 : cartesian_point := cartesian_point('p33', [50.0, 50.0, 50.0]);

(* normals to planar faces *)
  nfac1    : direction := direction ('nfac1', [-1, 0, -1]);
  nfac2    : direction := direction ([0, -1, -1]);
  nfac3    : direction := direction ('nfac3', [1, 0, -1]);
  nfac4    : direction := direction ('nfac4', [0, 1, -1]);

(* directions of edge lines *)
  dl00     : direction := direction ('dl00', [-1,-1, 1]);
  dl30     : direction := direction ('dl30', [-1, 1, 1]);
  dl03     : direction := direction ('dl03', [1, -1, 1]);
  dl33     : direction := direction ('dl33', [1, 1, 1]);

  vec00    : vector := vector ('vec00',  dl00, 1.0 ) ;
  vec30    : vector := vector ('vec30',  dl30, 1.0 ) ;
  vec03    : vector := vector ('vec03',  dl03, 1.0 ) ;
  vec33    : vector := vector ('vec33',  dl33, 1.0 ) ;

(* lines from origin *)
  lnp00    : line := line ('lnp00',  origin, vec00) ;
  lnp30    : line := line ('lnp30',  origin, vec30) ;
  lnp03    : line := line ('lnp03',  origin, vec03) ;
  lnp33    : line := line ('lnp33',  origin, vec33) ;

(* axis placements for planes *)
  a1  : axis2_placement_3d := axis2_placement_3d ('a1', origin,
                                                  nfac1, dl00) ;
  a2  : axis2_placement_3d := axis2_placement_3d ('a2', origin,
                                                  nfac2, dl30) ;
```

```
  a3  : axis2_placement_3d := axis2_placement_3d ('a3', origin,
                                                  nfac3, dl33) ;
  a4  : axis2_placement_3d := axis2_placement_3d ('a4', origin,
                                                  nfac4, dl03) ;

(* Planes for planar faces *)
   pl1     : plane := plane ('pl1', a1) ;
   pl2     : plane := plane ('pl2', a2) ;
   pl3     : plane := plane ('pl3', a3) ;
   pl4     : plane := plane ('pl4', a4) ;

(* B-spline curves of degree 3 for curved edges *)
  rbsplc1  : b_spline_curve := bezier_curve('rbsplc1',  3,
              [p00, p01,p02, p03], !unspecified, FALSE, FALSE ) ||
                        rational_b_spline_curve ( [w1, w2, w3, w1]);
  rbsplc2  : b_spline_curve := bezier_curve('rbsplc2',  3,
           [p00, p10, p20, p30], !unspecified, FALSE, FALSE ) ||
                       rational_b_spline_curve ( [w1, w2, w2, w1]);
  rbsplc3  : b_spline_curve := bezier_curve('rbsplc3',  3,
                [p30, p31, p32, p33], !unspecified, FALSE, FALSE ) ||
                        rational_b_spline_curve ( [w1, w2, w2, w1]);
  rbsplc4  : b_spline_curve := bezier_curve('rbsplc4',  3,
           [p03, p13, p23, p33], !unspecified, FALSE, FALSE ) ||
                        rational_b_spline_curve ( [w1, w2, w2, w1]);
(* B-spline surface of degree 3 for top face *)

 rbspsurf : b_spline_surface := bezier_surface ('rbspsurf',  3, 3 ,
                             [ [p00, p01, p02, p03],
                               [p10, p11, p12, p13],
                               [p20, p21, p22, p23],
                               [p30, p31, p32, p33] ], !unspecified,
                              FALSE, FALSE, FALSE ) ||
                               rational_b_spline_surface ( [
                               [w1, w2, w3, w1 ]
                               [w2, w3, w3, w2 ]
                               [w2, w3, w3, w2 ]
                               [w1, w2, w2, w1 ] ]) ;


 (* vertices and edges *)
   apex     : vertex_point := vertex_point ('apex', origin);
   v00      : vertex_point := vertex_point ('v00', p00);
   v03      : vertex_point := vertex_point ('v03', p03);
   v30      : vertex_point := vertex_point ('v30', p30);
   v33      : vertex_point := vertex_point ('v33', p33);
```

```
  el00 : edge_curve := edge_curve ('el00',  apex , v00, lnp00, TRUE );
  el03 : edge_curve := edge_curve ('el03',  apex , v03, lnp03, TRUE );
  el30 : edge_curve := edge_curve ('el30',  apex , v30, lnp30, TRUE );
  el33 : edge_curve := edge_curve ('el33',  apex , v33, lnp33, TRUE );
  ec1  : edge_curve := edge_curve ('ec1',  v00, v03, rbsplc1, TRUE );
  ec2  : edge_curve := edge_curve ('ec2',  v00, v30, rbsplc2, TRUE );
  ec3  : edge_curve := edge_curve ('ec3',  v30, v33, rbsplc3, TRUE );
  ec4  : edge_curve := edge_curve ('ec4',  v03, v33, rbsplc1, TRUE );

(* oriented_edges *)
  el00_t    : oriented_edge := oriented_edge ('el00_t', el00, TRUE) ;
  el00_f    : oriented_edge := oriented_edge ('el00_f', el00, FALSE);
  el03_t    : oriented_edge := oriented_edge ('el03_t', el03, TRUE) ;
  el03_f    : oriented_edge := oriented_edge ('el03_f', el03, FALSE);
  el30_t    : oriented_edge := oriented_edge ('el30_t', el30, TRUE) ;
  el30_f    : oriented_edge := oriented_edge ('el30_f', el30, FALSE);
  el33_t    : oriented_edge := oriented_edge ('el33_t', el33, TRUE) ;
  el33_f    : oriented_edge := oriented_edge ('el33_f', el33, FALSE);
  ec1_t     : oriented_edge := oriented_edge ('ec1_t', ec1, TRUE) ;
  ec1_f     : oriented_edge := oriented_edge ('ec1_f', ec1, FALSE) ;
  ec2_t     : oriented_edge := oriented_edge ('ec2_t', ec2, TRUE) ;
  ec2_f     : oriented_edge := oriented_edge ('ec2_f', ec2, FALSE) ;
  ec3_t     : oriented_edge := oriented_edge ('ec3_t', ec3, TRUE) ;
  ec3_f     : oriented_edge := oriented_edge ('ec3_f', ec3, FALSE) ;
  ec4_t     : oriented_edge := oriented_edge ('ec4_t', ec4, TRUE) ;
  ec4_f     : oriented_edge := oriented_edge ('ec4_f', ec4, FALSE) ;

(* face bounding loops  *)
  elleft  : edge_loop := edge_loop('elleft', [el00_t, ec1_t, el03_f]);
  elright : edge_loop := edge_loop('elright', [el33_t, ec3_f, el30_f]);
  elfront : edge_loop := edge_loop('elfront', [el30_t, ec2_f, el00_f]);
  elback  : edge_loop := edge_loop('elback', [el03_t, ec4_t, el33_f]);
  eltop   : edge_loop := edge_loop('eltop',
                                   [ec1_f, ec2_t, ec3_t, ec4_f]) ;

  fbleft  : face_outer_bound := face_outer_bound('fbleft', elleft,
                                                  TRUE) ;
  fbright : face_outer_bound := face_outer_bound('fbright', elright,
                                                  TRUE) ;
  fbfront : face_outer_bound := face_outer_bound('fbfront', elfront,
                                                 TRUE) ;
  fbback  : face_outer_bound := face_outer_bound('fbback', elback,
                                                 TRUE) ;
  fbtop   : face_outer_bound := face_outer_bound('fbtop', eltop,
                                                 TRUE) ;
```

```
(* faces *)
  afleft  : advanced_face :=
                advanced_face (’afleft’, [fbleft],  pl1, TRUE );
  afright : advanced_face :=
                advanced_face (’afright’, [fbright], pl3, TRUE );
  affront : advanced_face :=
                advanced_face (’affront’, [fbfront], pl2, TRUE );
  afback  : advanced_face :=
                advanced_face (’afback’, [fbback],  pl4, TRUE );
  aftop   : advanced_face :=
                advanced_face (’aftop’, [fbside],  rbspsurf, TRUE);
END_PARAMETER;

SCHEMA_DATA rat_bspl_pyr_ctxt;

END_SCHEMA_DATA;

END_CONTEXT ;

(*
```

C.4.9 genus_1_face_surface context

This context provides the face needed to define a **closed_shell** of genus 1 of approximately toroidal form. The surface of the face is defined as a doubly closed B-spline surface of degree 3×5. The figure is centred at (0, 0, 0) and all control points are defined using parametric length measures. These basic length measures can be varied to change the size and proportions of the surface.

```
*)
CONTEXT genus_1_face_surface;
  WITH aic_topology_bounded_surface;
PARAMETER
  a        : length_measure := 30.0;
  b        : length_measure := 40.0;
  c        : length_measure := 60.0;
  d        : length_measure := 80.0;
  h        : length_measure := 20.0;
  nega     : length_measure := -30.0;
  negb     : length_measure := -40.0;
  negc     : length_measure := -60.0;
  negd     : length_measure := -80.0;
  negh     : length_measure := -20.0;
(* control points for B-spline surface and for edge curves *)
 p00 : cartesian_point := cartesian_point (’p00’, [b, zero, zero]);
 p01 : cartesian_point := cartesian_point (’p01’, [b, zero, negh]);
 p02 : cartesian_point := cartesian_point (’p02’, [d, zero, negh]);
```

```
p03 : cartesian_point := cartesian_point (’p03’, [d, zero, h);
p04 : cartesian_point := cartesian_point (’p04’, [b, zero, h]);
p10 : cartesian_point := cartesian_point (’p10’, [a, a, zero]);
p11 : cartesian_point := cartesian_point (’p11’, [a, a, negh]);
p12 : cartesian_point := cartesian_point (’p12’, [c, c, negh]);
p13 : cartesian_point := cartesian_point (’p13’, [c, c, h);
p14 : cartesian_point := cartesian_point (’p14’, [a, a, h]);
p20 : cartesian_point := cartesian_point (’p20’, [nega, a, zero]);
p21 : cartesian_point := cartesian_point (’p21’, [nega, a, negh]);
p22 : cartesian_point := cartesian_point (’p22’, [negc, c, negh]);
p23 : cartesian_point := cartesian_point (’p23’, [negc, c, h);
p24 : cartesian_point := cartesian_point (’p24’, [nega, a, h]);
p30 : cartesian_point := cartesian_point (’p30’, [negb, zero, zero]);
p31 : cartesian_point := cartesian_point (’p31’, [negb, zero, negh]);
p32 : cartesian_point := cartesian_point (’p32’, [negd, zero, negh]);
p33 : cartesian_point := cartesian_point (’p33’, [negd, zero, h);
p34 : cartesian_point := cartesian_point (’p34’, [negb, zero, h]);
p40 : cartesian_point := cartesian_point (’p40’, [nega, nega, zero]);
p41 : cartesian_point := cartesian_point (’p41’, [nega, nega, negh]);
p42 : cartesian_point := cartesian_point (’p42’, [negc, negc, negh]);
p43 : cartesian_point := cartesian_point (’p43’, [negc, negc, h);
p44 : cartesian_point := cartesian_point (’p44’, [nega, nega, h]);
p50 : cartesian_point := cartesian_point (’p50’, [a, nega, zero]);
p51 : cartesian_point := cartesian_point (’p51’, [a, nega, negh]);
p52 : cartesian_point := cartesian_point (’p52’, [c, negc, negh]);
p53 : cartesian_point := cartesian_point (’p53’, [c, negc, h);
p54 : cartesian_point := cartesian_point (’p54’, [a, nega, h]);


(* closed B-spline curve of degree 5 for curved edge (used twice) *)
  bsplc  : b_spline_curve_with_knots :=
                          b_spline_curve_with_knots(’bsplc’,  5,
                          [p00, p01, p02, p03, p04, p00],
                    !unspecified, TRUE, UNKNOWN, [6,6], [0.0, 1.0],
                          !piecewise_bezier_knots ) ;
(* B-spline surface of degree 3 x 5  for closed shell of genus 1*)

   bspsurf : b_spline_surface_with_knots :=
              b_spline_surface_with_knots (’bspsurf’,  3, 5 ,
                                [ [p00, p01, p02, p03, p04, p00],
                                  [p10, p11, p12, p13, p14, p10],
                                  [p20, p21, p22, p23, p24, p20],
                                  [p30, p31, p32, p33, p34, p30],
                                  [p40, p41, p42, p43, p44, p40],
                                  [p50, p51, p52, p53, p54, p50],
                                  [p00, p01, p02, p03, p04, p00],
```

```
                                    !unspecified, TRUE, TRUE, UNKNOWN,
                                    [4, 1, 1, 1, 4], [6,6],
                                 [0.0, 1.0, 2.0, 3.0, 4.0], [0.0, 1.0],
                                    !unspecified ) ;

 (* vertex and edges *)
   v00      : vertex_point := vertex_point (’v00’, p00);

   ec       : edge_curve := edge_curve (’ec’,  v00, v00, bsplc, TRUE );

(* oriented_edge *)
   ec_t     : oriented_edge := oriented_edge (’ec_t’, ec, TRUE) ;

(* face bounding loop  *)
  eloop    : edge_loop := edge_loop (’eloop’, [ec_t]) ;

  fbleft    : face_bound := face_bound (’fbleft’, eloop, TRUE) ;
  fbright   : face_bound := face_outer_bound(’fbright’, eloop, FALSE);

(* face *)
  g_1_face  : advanced_face :=
                   advanced_face (’g_1_face’, [fbleft, fbright],
                                   bspsurf, TRUE );
END_PARAMETER;

SCHEMA_DATA gen1_fac_ctxt;

END_SCHEMA_DATA;

END_CONTEXT ;

(*
```

C.4.10 degenerate_torus context

This context provides the faces needed to define a **closed_shell** with degenerate **toroidal_surface** face geometry. Two curved faces are defined each of which is half of one of the instances (’apple’ or ’lemon’ of **degenerate_toroidal_surface**. The faces meet in a common central plane. The closed shell is completed by planar faces with circular arc edge geometry.

Parameter orc can be varied to re-locate origin and dimensions rad1, rad2 re-scaled to change scale of figure.

```
*)
CONTEXT degenerate_torus;
```

```
  WITH aic_topology_bounded_surface;
PARAMETER
  orc      : length_measure := 0.0;
  rad1     : length_measure := 30.0;
  rad2     : length_measure := 50.0;
  d1       : length_measure := 40.0;

 origin : cartesian_point := cartesian_point('origin',
                                             [orc, orc, orc]);
 p1  : cartesian_point := cartesian_point('p1', [orc, orc + d1, orc] );
 p2  : cartesian_point := cartesian_point('p2', [orc, orc - d1, orc] );
 p3  : cartesian_point := cartesian_point('p3',
                                             [rad1 + orc,  orc, orc]);
 p4  : cartesian_point := cartesian_point('p4',
                                           [orc - rad1, orc, 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_y,  pos_z) ;
 a3 : axis2_placement_3d := axis2_placement_3d('a3', p3, pos_z, pos_x);
 a4 : axis2_placement_3d := axis2_placement_3d('a4', p4, pos_z, pos_x);

(* degenerate toroidal surfaces + plane *)
   apple : degenerate_toroidal_surface :=
         degenerate_toroidal_surface ('apple', a1, rad1, rad2, TRUE) ;
   lemon : degenerate_toroidal_surface :=
        degenerate_toroidal_surface ('lemon', a1, rad1, rad2, FALSE) ;
   midplane : plane := plane ('midplane', a3) ;

 (* circles for bounding edges *)
  circ3     : circle := circle ('circ3', a3, rad2) ;
  circ4     : circle := circle ('circ4', a4, rad2) ;


 (* vertices and edges *)
   v1       : vertex_point := vertex_point ('v1', p1);
   v2       : vertex_point := vertex_point ('v2', p2);

  elong3  : edge_curve := edge_curve ('elong3', v2, v1, circ3, TRUE);
  eshort3 : edge_curve := edge_curve ('eshort3', v1, v2, circ3, TRUE);
  elong4  : edge_curve := edge_curve ('elong4', v1, v2, circ4, TRUE);
  eshort4 : edge_curve := edge_curve ('eshort4', v2, v1, circ4, TRUE);
```

```
(* oriented_edges *)
  el3_t   : oriented_edge := oriented_edge ('el3_t', elong3, TRUE) ;
  el3_f   : oriented_edge := oriented_edge ('el3_f', elong3, FALSE) ;
  es3_t   : oriented_edge := oriented_edge ('es3_t', eshort3, TRUE) ;
  es3_f   : oriented_edge := oriented_edge ('es3_f', eshort3, FALSE);
  el4_t   : oriented_edge := oriented_edge ('el4_t', elong4, TRUE) ;
  el4_f   : oriented_edge := oriented_edge ('el4_f', elong4, FALSE) ;
  es4_t   : oriented_edge := oriented_edge ('es4_t', eshort4, TRUE) ;
  es4_f   : oriented_edge := oriented_edge ('es4_f', eshort4, FALSE);

(* face bounding loops  *)
  elapple     : edge_loop := edge_loop ('elapple', [el3_f, el4_f]) ;
  ellemon     : edge_loop := edge_loop ('ellemon', [es3_t, es4_t]) ;
  elright     : edge_loop := edge_loop ('elright', [el3_t, es4_f]) ;
  elleft      : edge_loop := edge_loop ('elleft', [es3_f, el4_t]) ;

  fbapple      : face_outer_bound :=
                       face_outer_bound ('fbapple', elapple, TRUE) ;
  fblemon      : face_outer_bound :=
                       face_outer_bound ('fblemon', ellemon, TRUE) ;
  fbright      : face_outer_bound :=
                       face_outer_bound ('fbright', elright, TRUE) ;
  fbleft       : face_outer_bound :=
                       face_outer_bound ('fbleft', elleft, TRUE) ;

(* faces *)
  afapple   : advanced_face :=
                advanced_face ('afapple', [fbapple],  apple, TRUE );
  aflemon  : advanced_face :=
                advanced_face ('aflemon', [fblemon], lemon, TRUE );
  afright  : advanced_face :=
                advanced_face ('afright', [fbright], midplane, TRUE );
  afleft   : advanced_face :=
                 advanced_face ('afleft',[fbleft], midplane, TRUE );
END_PARAMETER;

SCHEMA_DATA deg_tor_ctxt;

END_SCHEMA_DATA;

END_CONTEXT ;

(*
```

C.4.11 Cylinder_sphere_pcurve context

This context provides the faces needed to define a simple closed shell of cylindrical shape with hemi-spherical base centred at (orc,orc,orc), radius rad and axial height h to oblique face.

All bounds are defined by **edge_loops** and which reference pcurves which have the geometric form of conics. Each pcurve is created in a special 2D **parametric_representation_context**.

```
*)
CONTEXT cylinder_sphere_pcurve ;
  WITH aic_topology_bounded_surface;
PARAMETER
  unit      : length_measure := 1.0;
  zero      : length_measure := 0.0;
  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);

  plctxt  : representation_context :=
                parametric_representation_context(’plctxt’,
                        ’parameter_space’ ) ||
                geometric_representation_context(2) ;
  cylctxt : representation_context :=
                parametric_representation_context(’cylctxt’,
                           ’parameter_space’  ) ||
                geometric_representation_context(2) ;
```

```
spctxt  : representation_context :=
              parametric_representation_context('spctxt',
                        'parameter_space' ) ||
              geometric_representation_context(2) ;

ppos_u  : direction := direction ('ppos_u', [1, 0]);
pvec_u  : vector := vector ('pvec_u', ppos_u, unit );
ppt1  : cartesian_point := cartesian_point ( 'ppt1', [zero, zero] );
ppt2  : cartesian_point := cartesian_point ( 'ppt1', [zero, h] );
pap2  : axis2_placement_2d := axis2_placement_2d ('pap2',
                                              ppt1, ppos_u );

plin1 : line := line ('plin1', ppt1, pvec_u );
plin2 : line := line ('plin2', ppt2, pvec_u );
pelli : ellipse := ellipse('pelli', pap2, majrad, rad);

dr1   : definitional_representation :=
             definitional_representation ('dr1', [plin1], spctxt);
dr2   : definitional_representation :=
            definitional_representation ('dr2', [plin2], cylctxt);
dr3   : definitional_representation :=
             definitional_representation ('dr3', [pelli], plctxt);

pcrv1 : pcurve := pcurve ('pcrv1', sphere, dr1 );
pcrv2 : pcurve := pcurve ('pcrv2', cyl, dr2 );
pcrv3 : pcurve := pcurve ('pcrv1', pl, dr3 );

circ  : circle := circle ('circ', a1 , rad);
elli  : ellipse := ellipse('elli', a2 , majrad ,rad);
scrv_top  : surface_curve := surface_curve ('scrv_top', circ,
                                      [pcrv1, pcrv2], !curve_3d);
scrv_bot  : surface_curve := surface_curve ('scrv_bot', elli,
                                          [pcrv3], !pcurve_s1);


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, scrv_top,
                                                              TRUE);
edge2  : edge_curve := edge_curve ('edge2', verte, verte, scrv_bot,
```

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

SCHEMA_DATA cyl_sph_pc_ctxt;

CONSTANT
  rt2 == sqrt(2.0);
  rt3 == sqrt(3.0);
END_CONSTANT;

END_SCHEMA_DATA;

END_CONTEXT ;
(*
```

C.4.12 Trimmed_cylinder context

This context provides the **advanced_faces** needed to define a simple closed shell in the form of a vertical cylinder trimmed with two parallel flat faces. The purpose is to illusrate the use of surface and curve entities to define multiple faces or edges.

```
*)
CONTEXT trimmed_cylinder ;
  WITH aic_topology_bounded_surface;
PARAMETER
  orc       : length_measure := 0.0;
  h         : length_measure := 100.0;
  rad       : length_measure := 25.0;
  distx     : length_measure := 20.0;
  disty     : length_measure := 15.0;
```

```
 unit      : length_measure := 1.0;

 origin    : cartesian_point := cartesian_point ('origin',
                                                 [orc, orc, orc]);
 ptleftb   : cartesian_point :=
    cartesian_point ('ptleftb', [orc - distx, orc + disty, orc + h]);
 ptleftf   : cartesian_point :=
     cartesian_point('ptleftf', [orc - distx, orc - disty, orc + h]);
 ptrightb  : cartesian_point :=
    cartesian_point('ptrightb', [orc + distx, orc + disty, orc + h]);
 ptrightb  : cartesian_point :=
   cartesian_point ('ptrightf', [orc + distx, orc - disty, orc + h]);
 pbleftb   : cartesian_point :=
        cartesian_point ('pbleftb', [orc - distx, orc + disty, orc]);
 pbleftf   : cartesian_point :=
        cartesian_point ('pbleftf', [orc - distx, orc - disty, orc]);
 pbrightb  : cartesian_point :=
        cartesian_point('pbrightb', [orc + distx, orc + disty, orc]);
 pbrightf  : cartesian_point :=
        cartesian_point('pbrightf', [orc + distx, orc - disty, orc]);
 ptop      : cartesian_point :=
              cartesian_point ('ptop', [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]);

 vec_z     : vector := vector('vec_z', pos_z, unit) ;
 vec_y     : vector := vector('vec_y', pos_y, unit) ;

 a1  : axis2_placement_3d := axis2_placement_3d ('a1', origin,
                                                  pos_z, pos_x) ;
 a2  : axis2_placement_3d := axis2_placement_3d ('a2', pleft,
                                        pos_y, pos_z) ;
 a3  : axis2_placement_3d := axis2_placement_3d ('a3', pright,
                                      pos_y, pos_z) ;
 a4  : axis2_placement_3d := axis2_placement_3d ('a4', ptop,
                                                  pos_z, pos_x) ;

pltop    : plane := plane ('pltop', a4) ;
plbase   : plane := plane('plbase', a1) ;
plleft   : plane := plane ('plleft', a2) ;
plright  : plane := plane('plright', a3) ;
cyl   : cylindrical_surface := cylindrical_surface ('cyl', a1, rad);

 circbas   : circle := circle ('circbas', a1 , rad);
```

```
  circtop   : circle := circle ('circtop', a4 , rad);

 (* lines for vertical edges *)
  linrf      : line := line ('linrf',  pbrightf, vec_z) ;
  linrb      : line := line ('linrf',  pbrightb, vec_z) ;
  linlf      : line := line ('linrf',  pbleftf, vec_z) ;
  linlb      : line := line ('linrf',  pbleftb, vec_z) ;
(* lines for horizontal edges *)
  lintl      : line := line ('lintl',  ptleftf, vec_y) ;
  lintr      : line := line ('lintr',  ptrightf, vec_y) ;
  linbl      : line := line ('linbl',  pbleftf, vec_y) ;
  linbr      : line := line ('linbr',  pbrightf, vec_y) ;

(* vertices vt : on top, vb : on base *)
  vtlb      : vertex_point := vertex_point ('vtlb', ptleftb);
  vtlf      : vertex_point := vertex_point ('vtlf', ptleftf);
  vtrb      : vertex_point := vertex_point ('vtrb', ptrightb);
  vtrf      : vertex_point := vertex_point ('vtrf', ptrightf);
  vblb      : vertex_point := vertex_point ('vblb', pbleftb);
  vblf      : vertex_point := vertex_point ('vblf', pbleftf);
  vbrb      : vertex_point := vertex_point ('vbrb', pbrightb);
  vbrf      : vertex_point := vertex_point ('vbrf', pbrightf);

(* vertical edges *)

 edgerb : edge_curve := edge_curve('edgerb', vbrb, vtrb, linrb, TRUE);
 edgerf : edge_curve := edge_curve('edgerf', vbrf, vtrf, linrf, TRUE);
 edgelb : edge_curve := edge_curve('edgerb', vblb, vtlb, linlb, TRUE);
 edgelf : edge_curve := edge_curve('edgerf', vblf, vtlf, linlf, TRUE);
 edgerb_t : oriented_edge := oriented_edge ('edgerb_t',edgerb , TRUE);
 edgerf_t : oriented_edge := oriented_edge ('edgerf_t',edgerf , TRUE);
 edgelb_t : oriented_edge := oriented_edge ('edgelb_t',edgelb , TRUE);
 edgelf_t : oriented_edge := oriented_edge ('edgelf_t',edgelf , TRUE);
 edgerb_f : oriented_edge := oriented_edge ('edgerb_f',edgerb ,FALSE);
 edgerf_f : oriented_edge := oriented_edge ('edgerf_f',edgerf ,FALSE);
 edgelb_f : oriented_edge := oriented_edge ('edgelb_f',edgelb ,FALSE);
 edgelf_f : oriented_edge := oriented_edge ('edgelf_f',edgelf ,FALSE);

(* horizontal edges *)
 edtopl : edge_curve := edge_curve ('edtopl', vtlf, vtlb, lintl, TRUE);
 edtopr : edge_curve := edge_curve ('edtopr', vtrf, vtrb, lintr, TRUE);
 edbasl : edge_curve := edge_curve ('edbasl', vblb, vtlb, linbl, TRUE);
 edbasr : edge_curve := edge_curve ('edbasr', vblf, vtlf, linbr, TRUE);
 edtopl_t  : oriented_edge := oriented_edge('edtopl_t',edgerb , TRUE);
 edtopr_t  : oriented_edge := oriented_edge('edtopr_t',edgerf , TRUE);
 edbasl_t  : oriented_edge := oriented_edge('edbasl_t',edgelb , TRUE);
```

```
 edbasr_t  : oriented_edge := oriented_edge('edbasr_t',edgelf , TRUE);
 edtopl_f  : oriented_edge := oriented_edge('edtopl_f',edgerb ,FALSE);
 edtopr_f  : oriented_edge := oriented_edge('edtopr_f',edgerf ,FALSE);
 edbasl_f  : oriented_edge := oriented_edge('edbasl_f',edgelb ,FALSE);
 edbasr_f  : oriented_edge := oriented_edge('edbasr_f',edgelf ,FALSE);

(* curved edges *)
 edtopf  : edge_curve := edge_curve ('edtopf', vtlf, vtrf, circtop,
                                                              TRUE);
 edtopb  : edge_curve := edge_curve ('edtopb', vtrb, vtlb, circtop,
                                                               TRUE);
 edbasf  : edge_curve := edge_curve ('edbasf', vblf, vbrf, circbas,
                                                               TRUE);
 edbasb  : edge_curve := edge_curve ('edbasb', vbrb, vblb, circbas,
                                                               TRUE);
 edtopf_t : oriented_edge := oriented_edge ('edtopf_t', edtopf, TRUE);
 edtopb_t : oriented_edge := oriented_edge ('edtopb_t', edtopb, TRUE);
 edbasf_t : oriented_edge := oriented_edge ('edbasf_t', edbasf, TRUE);
 edbasb_t : oriented_edge := oriented_edge ('edbasb_t', edbasb, TRUE);
 edtopf_f : oriented_edge := oriented_edge ('edtopf_f', edtopf,FALSE);
 edtopb_f : oriented_edge := oriented_edge ('edtopb_f', edtopb,FALSE);
 edbasf_f : oriented_edge := oriented_edge ('edbasf_f', edbasf,FALSE);
 edbasb_f : oriented_edge := oriented_edge ('edbasb_f', edbasb,FALSE);

(* bounding loops *)
 looptop  : edge_loop := edge_loop ('looptop', [edtopf_t, edtopr_t,
                                               edtopb_t, edtopl_f ]);
 loopbase : edge_loop := edge_loop ('loopbase', [edbasf_f, edbasl_t,
                                                 edbasb_f, edbasr_f ]);
 loopleft : edge_loop := edge_loop ('loopleft', [edgelf_t, edtopl_t,
                                                 edgelb_f, edbasl_f ]);
 loopright: edge_loop := edge_loop ('loopright', [edgerf_f, edbasr_t,
                                                 edgerb_t, edtopr_f ]);
 loopfront: edge_loop := edge_loop ('loopfront', [edbasf_t, edgerf_t,
                                                 edtopf_f, edgelf_f ]);
 loopeback: edge_loop := edge_loop ('loopeback', [edbasb_t, edgelb_t,
                                                 edtopb_f, edgerb_f ]);

 btop  : face_outer_bound := face_outer_bound('btop', looptop, TRUE);
 bbase : face_outer_bound := face_outer_bound('bbase', loopbase, TRUE);
 bleft : face_outer_bound := face_outer_bound('bleft', loopleft, TRUE);
bright : face_outer_bound := face_outer_bound('bright', loopright,
                                                              TRUE);
bfront : face_outer_bound := face_outer_bound('bfront', loopfront,
                                                               TRUE);
bback : face_outer_bound := face_outer_bound('bback', loopback, TRUE);
```

```
 front_face : advanced_face := advanced_face ('front_face',
                                              [bfront], cyl, TRUE );
 back_face : advanced_face := advanced_face ('back_face',
                                              [bback], cyl, TRUE );
 top_face : advanced_face :=
                 advanced_face ('top_face', [btop], pltop, TRUE );
 base_face : advanced_face := advanced_face ('base_face',
                                            [bbase], plbase, FALSE );
 left_face : advanced_face := advanced_face ('left_face',
                                             [bleft], plleft, FALSE );
 right_face : advanced_face := advanced_face ('right_face',
                                             [bright], plright, TRUE );

END_PARAMETER;

SCHEMA_DATA trim_cyl_ctxt;

END_SCHEMA_DATA;

END_CONTEXT ;
(*
```

C.5 Contexts defined for presentation-related abstract test cases

C.5.1 Set_ambience context

This context provides the facility to set the ambient lighting level.

```
CONTEXT set_ambience ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component, green_component,
                                                blue_component ) ;
  END_PARAMETER ;

  SCHEMA_DATA set_ambience ;
    als = light_source_ambient {SUBOF (@ambient_light_source); } ;
```

```
    ambient_light_source = light_source {name -> 'ambient1';
                        light_colour -> @c; SUPOF (@als); } ;
  END_SCHEMA_DATA ;
END_CONTEXT ;
```

C.5.2 Set_positional_light context

This context provides the facility to set a positional light.

```
CONTEXT set_positional_light ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
                       green_component, blue_component ) ;

    x : length_measure := 30 ;
    y : length_measure := 40 ;
    z : length_measure := 50 ;
    at : cartesian_point := cartesian_point ( 'at', [x, y, z] ) ;

    constant_attenuation : real := 0.4 ;
    distance_attenuation : real := 0.7 ;
  END_PARAMETER ;

  SCHEMA_DATA set_positional_light ;
    pls = light_source_positional {
            SUBOF (@positional_light_source);
            position -> @at;
            constant_attenuation -> @constant_attenuation;
            distance_attenuation -> @distance_attenuation;
          } ;
    positional_light_source = light_source {name -> 'pos1';
                              light_colour -> @c; SUPOF (@pls); } ;
  END_SCHEMA_DATA ;
END_CONTEXT ;
```

C.5.3 Set_directional_light context

This context provides the facility to set a directional light.

```
CONTEXT set_directional_light ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
                        green_component, blue_component ) ;

    orx : real := 0.7 ;
    ory : real := 0.7 ;
    orz : real := 0.7 ;
    orientation : direction := direction ( 'or', [orx, ory, orz] ) ;
  END_PARAMETER ;

  SCHEMA_DATA set_directional_light ;
    dls = light_source_directional {
            SUBOF (@directional_light_source);
            orientation -> @orientation;
          } ;
    directional_light_source = light_source {name -> 'dir1';
                               light_colour -> @c; SUPOF (@dls); };
  END_SCHEMA_DATA ;
END_CONTEXT ;
```

C.5.4 Set_spot_light context

This context provides the facility to set a spotlight.

```
CONTEXT set_spotlight ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
                    green_component, blue_component ) ;

    constant_attenuation : real := 0.4 ;
    distance_attenuation : real := 0.7 ;

    concentration : real := 0.8 ;
```

```
    spread        : postive_plane_angle_measure := 45 ;

    x : length_measure := 30 ;
    y : length_measure := 40 ;
    z : length_measure := 50 ;
    at : cartesian_point := cartesian_point ( 'at', [x, y, z] ) ;

    orx : real := 0.7 ;
    ory : real := 0.7 ;
    orz : real := 0.7 ;
    orientation : direction := direction ( 'or', [orx, ory, orz] ) ;
  END_PARAMETER ;

  SCHEMA_DATA set_spotlight ;
    sls = light_source_spot {
            SUBOF (@spot_light_source);
            position -> @at;
            orientation -> @orientation;
            constant_attenuation -> @constant_attenuation;
            distance_attenuation -> @distance_attenuation;
            concentration_exponent -> @concentration;
            spread_angle -> @spread;
          } ;
    spot_light_source = light_source {name ->
                  'spot1'; light_colour -> @c; SUPOF (@sls); } ;
  END_SCHEMA_DATA ;
END_CONTEXT ;
```

C.5.5 Set_ambient_style_rendering context

This context provides the facility to set ambient style rendering with completely non-transparent surfaces.

```
CONTEXT set_ambient_style_rendering ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
                               green_component, blue_component );

    surface_method : shading_surface_method := normal_shading ;

    ambience : real := 0.7 ;
```

```
    transparency : real := 0.0 ;

    sst : surface_style_transparent
        := surface_style_transparent (@transparency) ;
    ssa : surface_style_reflectance_ambient
        := surface_style_reflectance_ambient (@ambience) ;

    ssrwp : surface_style_rendering_with_properties
          := surface_style_rendering_with_properties ([sst, ssa]) ;
  END_PARAMETER ;

  SCHEMA_DATA set_ambient _style ;
    surface_style = surface_style_rendering {
                      rendering_method -> @method ;
                      surface_colour -> @c ;
                      SUPOF(@ssrwp) ;
                    } ;
  END_SCHEMA_DATA ;
END_CONTEXT ;
```

C.5.6 Set_diffuse_style_rendering context

This context provides the facility to set diffuse style rendering with completely non-transparent surfaces.

```
CONTEXT set_diffuse_style_rendering ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
                   green_component, blue_component ) ;

    surface_method : shading_surface_method := normal_shading ;

    ambience     : real := 0.7 ;
    diffuseness  : real := 0.7 ;
    transparency : real := 0.0 ;

    sst  : surface_style_transparent
         := surface_style_transparent (@transparency) ;
    ssad : surface_style_reflectance_ambient_diffuse
         := surface_style_reflectance_ambient_diffuse
                              (@ambience, @diffuseness) ;
```

```
    ssrwp : surface_style_rendering_with_properties
          := surface_style_rendering_with_properties ([sst, ssad]) ;
  END_PARAMETER ;

  SCHEMA_DATA set_diffuse_style ;
    surface_style = surface_style_rendering {
                      rendering_method -> @method ;
                      surface_colour -> @c ;
                      SUPOF(@ssrwp) ;
                    } ;
  END_SCHEMA_DATA ;
END_CONTEXT ;
```

C.5.7 Set_specular_style_rendering context

This context provides the facility to set specular style rendering with completely non-transparent surfaces.

```
CONTEXT set_specular_style_rendering ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
                      green_component, blue_component ) ;

    specular_colour_name     : label := @colour_name ;
    specular_red_component   : real := @specular_red_component ;
    specular_green_component : real := @specular_green_component ;
    specular_blue_component  : real := @specular_blue_component ;
    specular_colour    : colour := colour (specular_colour_name,
                                           specular_red_component,
                                           specular_green_component,
                                           specular_blue_component ) ;

    surface_method : shading_surface_method := normal_shading ;

    ambience     : real := 0.7 ;
    diffuseness  : real := 0.7 ;
    transparency : real := 0.0 ;
    specularity  : real := 0.7 ;
    specular_exponent : real := 0.3 ;
```

```
    sst   : surface_style_transparent
          := surface_style_transparent (@transparency) ;
    ssads : surface_style_reflectance_ambient_diffuse_specular
          := surface_style_reflectance_ambient_diffuse_specular
              (@ambience, @diffuseness, @specularity,
               @specular_exponent, @specular_colour) ;

    ssrwp : surface_style_rendering_with_properties
          := surface_style_rendering_with_properties ([sst, ssads]) ;
  END_PARAMETER ;

  SCHEMA_DATA set_specular_style ;
    surface_style = surface_style_rendering {
                      rendering_method -> @method ;
                      surface_colour -> @c ;
                      SUPOF(@ssrwp) ;
                    } ;
  END_SCHEMA_DATA ;
END_CONTEXT ;
```

C.5.8 Set_curve_style_rendering context

This context provides the facility to set curve style rendering, with linear style as the default.

```
CONTEXT set_curve_style_rendering ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
                   green_component, blue_component ) ;

    curve_method : shading_curve_method := linear_colour ;
  END_PARAMETER ;

  SCHEMA_DATA set_curve_rendering ;
    srp = surface_rendering_properties { rendered_colour -> @c; } ;
    csr = curve_style_rendering { rendering_method -> @method;
                                  rendering_properties -> @srp; } ;

    curve_style = curve_or_render { @csr; } ;
  END_SCHEMA_DATA ;
```

```
END_CONTEXT ;
```

C.5.9 Set_clipped_camera_model_d3 context

This context provides the facility to set up a three-dimensional camera model required for non-rendered presentation.

```
CONTEXT set_clipped_camera_model_d3 ;

  PARAMETER
    camera_model_name : label := 'camera_model_name' ;
    projection : central_or_parallel := parallel ;

    ppx : length_measure := 0.0 ;
    ppy : length_measure := 0.0 ;
    ppz : length_measure := 0.0 ;
    projection_point : cartesian_point := cartesian_point ('pp',
                                                [ppx, ppy, ppz]) ;

    view_plane_distance : length_measure := 1.0 ;
    front_plane_distance : length_measure := 1.0 ;
    back_plane_distance : length_measure := 1.0 ;

    front_clipping : boolean := true ;
    back_clipping : boolean := true ;
    sides_clipping : boolean := true ;

    zero : length_measure := 0.0;
    xl   : length_measure := 100.0 ;
    yl   : length_measure := 100.0 ;
    origin : cartesian_point := cartesian_point ('origin',
                                          [zero, zero, zero]) ;
    neg_x  : direction := direction ('neg_x', [-1, 0, 0]) ;
    neg_y  : direction := direction ('neg_y', [0, -1, 0]) ;
    neg_z  : direction := direction ('neg_z', [0, 0, -1]) ;

    a2 : axis2_placement := axis2_placement_2d ('a2',origin, ?);
    box : planar_box ('box', xl, yl, a2) ;

    view : view_volume
         := view_volume (projection, projection_point,
                         view_plane_distance,
                         front_plane_distance, front_clipping,
                         back_plane_distance, back_clipping,
                         sides_clipping,
                         box) ;
```

```
    view_reference_system : axis2_placement_3d :=
                              axis2_placement_3d('vrs',origin, ?, ? );
  END_PARAMETER ;

  SCHEMA_DATA set_clipped_camera_model ;
    camera_model = camera_model {
                     name -> @camera_model_name ;
                     view_reference_system -> @view_reference_system ;
                     perspective_of_volume -> @view ; } ;
  END_SCHEMA_DATA ;
END_CONTEXT ;
```

C.5.10 Set_camera_model_d3_shading context

This context provides the facility to set up a three-dimensional camera model required for rendering, which includes the facility to set hidden line and surface elimination, and to use lighting values which have been set in other contexts.

```
CONTEXT set_camera_model_d3_shading ;

  PARAMETER
    camera_model_name : label := 'camera_model_name' ;
    hlhsr : boolean := true ;
    lights : SET [1:?] OF light_source ;
    with_hlhsr  : camera_model_d3_with_hlhsr
                 := camera_model_d3_with_hlhsr (@hlhsr) ;
    with_lights : camera_model_d3_with_light_sources
                 := camera_model_d3_with_light_sources (@lights) ;

    projection : central_or_parallel := parallel ;

    ppx : length_measure := 0.0 ;
    ppy : length_measure := 0.0 ;
    ppz : length_measure := 0.0 ;
    projection_point : cartesian_point := cartesian_point ('pp',
                                                [ppx, ppy, ppz]) ;

    view_plane_distance : length_measure := 1.0 ;
    front_plane_distance : length_measure := 1.0 ;
    back_plane_distance : length_measure := 1.0 ;

    front_clipping : boolean := true ;
    back_clipping : boolean := true ;
    sides_clipping : boolean := true ;
```

```
    zero : length_measure := 0;
    xl   : length_measure := 100.0 ;
    yl   : length_measure := 100.0 ;
   origin : cartesian_point := cartesian_point ('origin',
                    [zero, zero, zero]) ;
    neg_x  : direction := direction ('neg_x', [-1, 0, 0]) ;
    neg_y  : direction := direction ('neg_y', [0, -1, 0]) ;
    neg_z  : direction := direction ('neg_z', [0, 0, -1]) ;

    a2 : axis2_placement := axis2_placement_2d ('a2',origin, ? ) ;
    box : planar_box ('box', xl, yl, a2) ;

    view : view_volume
         := view_volume (projection, projection_point,
                         view_plane_distance,
                         front_plane_distance, front_clipping,
                         back_plane_distance, back_clipping,
                         sides_clipping,
                         box) ;

    view_reference_system : axis2_placement_3d :=
                                 axis2_placement_3d ( ? ) ;
  END_PARAMETER ;

  SCHEMA_DATA set_camera_model ;
     camera_model = camera_model {
                    name -> @camera_model_name ;
                    view_reference_system -> @view_reference_system ;
                    perspective_of_volume -> @view ;
                    SUPOF (@with_hlhsr, @with_lights); } ;
  END_SCHEMA_DATA ;
END_CONTEXT ;
```

C.5.11 Set_point_style context

This context provides the facility to set the point style to one of the predefined types.

```
CONTEXT set_point_style ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
```

```
                    green_component, blue_component ) ;

    predefined_type : marker_select := (!x;) ;
    size : positive_length_measure := 0.01 ;
    marker_name : label := 'x' ;
  END_PARAMETER ;

  SCHEMA_DATA set_predefined_point_style ;
    point_style = point_style {name -> @marker_name;
                               marker -> @predefined_type;
                               marker_size -> @size;
                               marker_colour -> @colour;} ;
  END_SCHEMA_DATA ;

END_CONTEXT ;
```

C.5.12 Set_curve_style context

This context provides the facility to set the curve style to one of the predefined types.

```
CONTEXT set_curve_style ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
                   green_component, blue_component ) ;

    predefined_type : curve_font_or_scaled_curve_font_select :=
                                                  (!continuous;) ;
    width : positive_length_measure := 0.01 ;
    curve_name : label := 'continuous' ;
  END_PARAMETER ;

  SCHEMA_DATA set_curve_style ;
    curve_style = curve_style {name -> @curve_name;
                               curve_font -> @predefined_type;
                               curve_width -> @width;
                               curve_colour -> @colour;} ;
  END_SCHEMA_DATA ;

END_CONTEXT ;
```

C.5.13 Set_user_defined_curve_style context

This context provides the facility to set a user-defined curve style.

```
CONTEXT set_user_defined_curve_style ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
                           green_component, blue_component ) ;

    width : positive_length_measure := 0.01 ;
    curve_name : label := 'user defined' ;
    pattern : LIST [1:?] OF curve_style_font_pattern ;

    user_defined_type : curve_font_or_scaled_curve_font_select
                      := (@pattern;) ;
  END_PARAMETER ;

  SCHEMA_DATA set_user_defined_curve_style ;
    curve_style = curve_style {name -> @curve_name;
                               curve_font -> @user_defined_type;
                               curve_width -> @width;
                               curve_colour -> @colour;} ;
  END_SCHEMA_DATA ;

END_CONTEXT ;
```

C.5.14 Set_surface_style context

This context provides the facility to set the surface styles.

```
CONTEXT set_surface_style ; % 517

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
                   green_component, blue_component ) ;
```

```
  side : surface_side := !both ;

  u_iso_lines : u_direction_count := 5 ;
  v_iso_lines : v_direction_count := 5 ;
END_PARAMETER ;

SCHEMA_DATA set_surface_style ;
  LOCAL
    parameter_line_curve, control_grid_curve, silhouette_curve,
      segmentation_curve, boundary_curve : curve_style ;

    parameter_line : surface_style_parameter_line ;
    control_grid   : surface_style_control_grid ;
    silhouette     : surface_style_silhouette ;
    segmentation   : surface_style_segmentation_curve ;
    boundary       : surface_style_boundary ;

    styles        : surface_side_style ;
    surface_style : surface_style_usage ;
  END_LOCAL ;

  CALL set_predefined_curve_style ;
    IMPORT (parameter_line_curve := @curve_style ;) ;
  END_CALL ;

  CALL set_predefined_curve_style ;
    IMPORT (control_grid_curve := @curve_style ;) ;
  END_CALL ;

  CALL set_predefined_curve_style ;
    IMPORT (silhouette_curve := @curve_style ;) ;
  END_CALL ;

  CALL set_predefined_curve_style ;
    IMPORT (segmentation_curve := @curve_style ;) ;
  END_CALL ;

  CALL set_predefined_curve_style ;
    IMPORT (boundary_curve := @curve_style ;) ;
  END_CALL ;

  parameter_line = surface_style_parameter_line {
    style_of_parameter_lines -> @parameter_line_curve ;
    direction_counts -> [@u_iso_lines; @v_iso_lines]; } ;
```

```
    control_grid = surface_style_control_grid {
      style_of_control_grid -> @control_grid_curve ; } ;

    silhouette = surface_style_silhouette {
      style_of_silhouette -> @silhouette_curve ; } ;

    segmentation_curve = surface_style_segmentation_curve {
      style_of_segmentation_curve -> @segmentation_curve ; } ;

    boundary = surface_style_boundary
               {style_of_boundary -> boundary_curve ; } ;

    styles = surface_side_style {
               name -> @name;
               styles -> [@parameter_line, @control_grid, @silhouette,
                          @segmentation_curve, @boundary];
                                 } ;

    surface_style = surface_style_usage {side -> @which_side;
                                         style -> @styles;} ;
  END_SCHEMA_DATA ;

END_CONTEXT ;
```

C.5.15 Set_rendered_surface_style context

This context provides the facility to set the surface styles.

```
CONTEXT set_rendered_surface_style ;

  PARAMETER
    colour_name     : label := 'white' ;
    red_component   : real := 0.2 ;
    green_component : real := 0.2 ;
    blue_component  : real := 0.2 ;
    c : colour
        := colour (colour_name, red_component,
               green_component, blue_component ) ;

    side : surface_side := !both ;

    u_iso_lines : u_direction_count := 5 ;
    v_iso_lines : v_direction_count := 5 ;

    fill_area_colour_name     : label := colour_name ;
    fill_area_red_component   : real := red_component ;
```

```
  fill_area_green_component : real := green_component ;
  fill_area_blue_component  : real := blue_component ;
  fill_area_colour : colour
      := colour (fill_area_colour_name, fill_area_red_component,
              fill_area_green_component, fill_area_blue_component );

  fill_area_name : label := 'complete' ;
  fill_area : surface_style_fill_area
            := surface_style_fill_area (@fill_area_colour_name,
                                        @fill_area_colour) ;

  rendering : surface_style_rendering ;
END_PARAMETER ;

SCHEMA_DATA set_rendered_surface_style ;
  LOCAL
    parameter_line_curve, control_grid_curve, silhouette_curve,
      segmentation_curve, boundary_curve : curve_style ;

    parameter_line : surface_style_parameter_line ;
    control_grid   : surface_style_control_grid ;
    silhouette     : surface_style_silhouette ;
    segmentation   : surface_style_segmentation_curve ;
    boundary       : surface_style_boundary ;
    fill_area      : surface_style_fill_area ;

    styles        : surface_side_style ;
    surface_style : surface_style_usage ;
  END_LOCAL ;

  CALL set_predefined_curve_style ;
    IMPORT (parameter_line_curve := @curve_style ;) ;
  END_CALL ;

  CALL set_predefined_curve_style ;
    IMPORT (control_grid_curve := @curve_style ;) ;
  END_CALL ;

  CALL set_predefined_curve_style ;
    IMPORT (silhouette_curve := @curve_style ;) ;
  END_CALL ;

  CALL set_predefined_curve_style ;
    IMPORT (segmentation_curve := @curve_style ;) ;
  END_CALL ;
```

```
    CALL set_predefined_curve_style ;
      IMPORT (boundary_curve := @curve_style ;) ;
    END_CALL ;

    parameter_line = surface_style_parameter_line {
      style_of_parameter_lines -> @parameter_line_curve ;
      direction_counts -> [@u_iso_lines; @v_iso_lines]; } ;

    control_grid = surface_style_control_grid {
      style_of_control_grid -> @control_grid_curve ; } ;

    silhouette = surface_style_silhouette {
      style_of_silhouette -> @silhouette_curve ; } ;

    segmentation_curve = surface_style_segmentation_curve {
      style_of_segmentation_curve -> @segmentation_curve ; } ;

    boundary = surface_style_boundary {style_of_boundary ->
                                          boundary_curve ; } ;

    styles = surface_side_style {
               name -> @name;
               styles -> [@parameter_line, @control_grid, @silhouette,
                          @segmentation_curve, @boundary,
                                 @fill_area, @rendering];
                             } ;

    surface_style = surface_style_usage {side -> @which_side;
                            style -> @styles;} ;
  END_SCHEMA_DATA ;

END_CONTEXT ;
```

C.5.16 Set_presentation_styles context

This context provides the facility to set the presentation styles for 517 and/or 518, using values initialised in other contexts.

```
CONTEXT set_presentation_styles ;

  PARAMETER
    point_style : point_style ;
    curve_style : curve_style ;
    surface_style : surface_style_usage ;
  END_PARAMETER ;
```

```
  SCHEMA_DATA presentation_styles ;
   presentation_styles = [@point_style, @curve_style, @surface_style];
  END_SCHEMA_DATA ;
END_CONTEXT ;
```

Table C.1 – Use of contexts in test cases

Context	Test Cases
basic_product_structure	ab1 – ab15, eb1 – eb6, fb1 – fb6, ps1 – ps4, np1, vp1 – vp7
b_spline_pyramid_faces	ab9
cone_faces	ab6
cone_shell	eb5
cylinder_sphere_faces	ab1, ab2, ab14, ps1, ps4, np1, vp7
cylinder_sphere_pcurve	ab13
cylinder_sphere_shell	eb1, eb2, vp6
cylinder_union_b_spline	ab5
cylinder_union_polyline	eb4, eb6
cylinder_union_polyline_advanced	ab4, ab7
degenerate_torus	ab12
genus_1_face_surface	ab11
rational_b_spline_pyramid_faces	ab10
set_ambience	vp1 – vp7
set_ambient_style_rendering	vp1, vp2
set_camera_model_d3_shading	vp1, vp2, vp3, vp4, vp5, vp6, vp7
set_clipped_camera_model_d3	
set_curve_style	vp2, vp3, vp4, vp6, vp7
set_curve_style_rendering	vp1
set_diffuse_style_rendering	vp3, vp5
set_directional_light	vp2, vp4, vp6, vp7
set_point_style	vp2 – vp7
set_positional_light	vp1, vp4, vp6, vp7
set_presentation_styles	vp1 – vp7
set_specular_style_rendering	vp4, vp6, vp7
set_spotlight	vp3 – vp7
set_surface_style	
set_rendered_surface_style	vp2 – vp7
set_user_defined_curve_style	vp5
swept_faces	ab8
tetrashell_instance	fb1, fb2, fb4, fb5, vp1, vp2, vp3, vp4, vp5
toroidal_segment	eb3
toroidal_segment_advanced	ab3
trimmed_cylinder	ab15

Annex D
(informative)

Test purposes without verdict criteria

The following test purposes are not provided with test cases or verdict criteria.

D.1 Test purposes which excluded by Express constraints

D.1.1 bounded_curve

aim442: bounded_curve as **composite_curve**

D.1.2 composite_curve

aim443: composite_curve with **self_intersect** = true

aim444: composite_curve with **self_intersect** = false

aim445: composite_curve with **self_intersect** = unknown

aim446: composite_curve with **segments** having at least one element as composite_curve_segment

D.1.3 composite_curve_segment

aim447: composite_curve_segment with **parent_curve** as curve

aim448: composite_curve_segment with **same_sense** = true

aim449: composite_curve_segment with **same_sense** = false

aim450: composite_curve_segment with **transition** = cont_same_gradient_same_curvature

aim451: composite_curve_segment with **transition** = cont_same_gradient

aim452: composite_curve_segment with **transition** = continuous

aim453: composite_curve_segment with **transition** = discontinuous

D.1.4 connected_face_set

aim454: connected_face_set as **open_shell**

D.1.5 face

aim455: face as **oriented_face**

D.1.6 open_shell

aim456: open_shell as **oriented_open_shell**

D.1.7 oriented_face

aim457: oriented_face with **orientation** = true

aim458: oriented_face with **orientation** = false

aim459: oriented_face with **face_element** as face

D.1.8 oriented_open_shell

aim460: oriented_open_shell with **orientation** = true

aim461: oriented_open_shell with **orientation** = false

aim462: oriented_open_shell with **open_shell_element** as open_shell

D.1.9 oriented_path

aim463: oriented_path with **orientation** = true

aim464: oriented_path with **orientation** = false

aim465: oriented_path with **path_element** as path

D.1.10 path

aim466: path as **oriented_path**

D.1.11 property_definition

aim467: property_definition with **definition** as product_definition_shape

D.1.12 surface_curve

aim468: surface_curve with **associated_geometry** having at least one element as surface

D.2 Test purposes which are excluded by application protocol requirements

D.2.1 b_spline_curve

aim469: b_spline_curve with **self_intersect** = true

aim470: b_spline_curve with **curve_form** = hyperbolic_arc

aim471: b_spline_curve with **curve_form** = parabolic_arc

aim472: b_spline_curve with **curve_form** = elliptic_arc

aim473: b_spline_curve with **curve_form** = circular_arc

aim474: b_spline_curve with **curve_form** = polyline_form

D.2.2 b_spline_surface

aim475: b_spline_surface with **self_intersect** = true

aim476: b_spline_surface with **surface_form** = surf_of_linear_extrusion

aim477: b_spline_surface with **surface_form** = quadric_surf

aim478: b_spline_surface with **surface_form** = generalised_cone

aim479: b_spline_surface with **surface_form** = ruled_surf

aim480: b_spline_surface with **surface_form** = surf_of_revolution

aim481: b_spline_surface with **surface_form** = toroidal_surf

aim482: b_spline_surface with **surface_form** = spherical_surf

aim483: b_spline_surface with **surface_form** = conical_surf

aim484: b_spline_surface with **surface_form** = cylindrical_surf

aim485: b_spline_surface with **surface_form** = plane_surf

D.3 Test purposes of no practical importance

The test purposes in the next six subclauses would require instantiation of certain B-spline subtypes which are rarely used in practice. In each instance the identical geometry may be created by using the more common Bezier subtype.

D.3.1 b_spline_curve

aim486: b_spline_curve as **quasi_uniform_curve**

aim487: b_spline_curve as **uniform_curve**

D.3.2 b_spline_surface

aim488: b_spline_surface as **quasi_uniform_surface**

aim489: b_spline_surface as **uniform_surface**

D.3.3 quasi_uniform_curve

aim490: quasi_uniform_curve (no specific test purposes)

D.3.4 quasi_uniform_surface

aim491: quasi_uniform_surface (no specific test purposes)

D.3.5 uniform_curve

aim492: uniform_curve (no specific test purposes)

D.3.6 uniform_surface

aim493: uniform_surface (no specific test purposes)

D.3.7 si_unit

aim494: si_unit with **prefix** absent

aim495: si_unit with **prefix** present as atto

aim496: si_unit with **prefix** present as femto

aim497: si_unit with **prefix** present as pico

aim498: si_unit with **prefix** present as nano

aim499: si_unit with **prefix** present as micro

aim500: si_unit with **prefix** present as centi

aim501: si_unit with **prefix** present as deci

aim502: si_unit with **prefix** present as deca

aim503: si_unit with **prefix** present as hecto

aim504: si_unit with **prefix** present as kilo

aim505: si_unit with **prefix** present as mega

aim506: si_unit with **prefix** present as giga

aim507: si_unit with **prefix** present as tera

aim508: si_unit with **prefix** present as peta

aim509: si_unit with **prefix** present as exa

D.3.8 product_category

aim510: product_category with **description** present as text

aim511: product_category with **description** absent

aim512: product_category with **name** as label

aim513: product_category as **product_related_product_category**

D.3.9 product_related_product_category

aim514: product_related_product_category with **products** having at least one element as product

D.3.10 Test purposes involving complex assembly structures

ae414: Assembly containing more than one assembly in roles of sub-assembly.

ae428: Assembly with sub-assembly located by transformation.

D.3.11 Test purposes based upon zero instances

ae422: Product with zero parts.

ae424: Part in zero products.

D.3.12 Presentation layout

ae507: screen_image as presentation_area.

ae516: screen_image containing one 3D_projection.

ae517: screen_image containing two or more 3D_projections.

D.3.13 Presentation of specific geometry or topology

ae525: Curve_appearance associated with one curve.

ae529: Curve_appearance associated with one edge.

ae533: Point_appearance associated with one point.

ae541: Presentation_appearance associated with one shell.

ae546: Surface_appearance associated with one face.

ae550: Surface_appearance associated with one surface.

ae554: Topological_representation_item associated with one presentation attribute.

D.3.14 Many-to-one presentation relationships

ae520: 3D_projection presenting many B-rep models.

ae523: B-rep model associated with many 3D_projections.

ae538: Presentation_appearance associated with many B-reps.

ae539b: B-rep model presented as many presentation_appearances.

Annex E
(informative)

Error tests

E.1 Error tests for faceted B-rep AIC

E.1.1 Test case fb6

Test case fb6 is designed to test the definition of a faceted B-rep containing one or more voids and to test the detection of illegal intersections and nesting of the voids. The **tetrashell_instance** context is used with different parameters to define the outer shell and the void shells. The result is an illegal hollow tetrahedral solid with void(s) of a similar shape. This test is defined as a post-processor test only.

NOTE 1 If required this test can easily be modified to test geometric precision by varying the parameters to define voids which are very close to each other or to the outer shell. As defined in the current version of this test case interference should be detected in all examples.

NOTE 2 **tetrashell_instance** context is re-used with different parameters to define void shells, and with default parameters to define outer shell.

E.1.1.1 Postprocessor input specification

```
*)
TEST_CASE example_204_6; WITH faceted_brep_aic;

  REALIZATION

  LOCAL
    prod1_name : STRING ;
    shape_1_def : product_definition_shape ;
    shape1_def_rep3 : shape_definition_representation ;
    shell_object, hollow1, hollow2, hollow3, hollow4 : closed_shell ;
    void1, void2, void3, void4 : oriented_closed_shell ;
    tetra_with_void : manifold_solid_brep ;
    tetra_with_voids1 : manifold_solid_brep ;
    tetra_with_voids2 : manifold_solid_brep ;
    fbsr1, fbsr2, fbsr3 : faceted_brep_shape_representation ;
    its_context : representation_context ;
    its_units : named_unit ;
    len_exp   : dimensional_exponents :=
                    dimensional_exponent(1, 0, 0, 0, 0, 0, 0) ;
  END_LOCAL;

    CALL basic_product_structure ;
```

```
      IMPORT (shape_1_def := @prod_def_shape; );
      WITH   (prod_name := @prod1_name; );
    END_CALL;

    CALL tetrashell_instance ; -- uses default values, so no WITH
      IMPORT (shell_object := @tetrashell; );
    END_CALL;

    CALL tetrashell_instance ;
(*     params. re-set for dimensions (oversize void)  *)
      IMPORT (hollow1 := @tetrashell; );
      WITH (orc := 20; lx := 80.001; ly := 80.001; lz := 80.001;);
    END_CALL;

    void1 := oriented_closed_shell('void1', hollow1, FALSE);

  tetra_with_void := faceted_brep('tetra_with_void', shell_object) ||
          brep_with_voids ([void1]) ; -- void intersects outer shell

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


    its_context := geometric_representation_context
                   ('context', 'context_for_tetrahedron', 3) ||
                   global_unit_assigned_context ( [its_units]);


   CALL tetrashell_instance ;
(*    params. re-set for dimensions (large void)  *)
      IMPORT (hollow2 := @tetrashell; );
      WITH (orc := 20; lx := 50; ly := 50; lz := 50;);
    END_CALL;

    void2 := oriented_closed_shell('void2', hollow2, FALSE);

    CALL tetrashell_instance ;
(*  parameters re-set for dimensions (nested void)  *)
      IMPORT (hollow3 := @tetrashell; );
      WITH (orc := 25; lx := 20; ly := 20; lz := 20;);
    END_CALL;

   void3 := oriented_closed_shell('void3', hollow3, FALSE) ;

    tetra_with_voids1 := faceted_brep (shell_object) ||
                         brep_with_voids ([void2, void3]) ;
```

```
    CALL tetrashell_instance ;
(*   parameters re-set for dimensions (small void)  *)
      IMPORT (hollow4 := @tetrashell; );
      WITH (orc := 15; lx := 20; ly := 20; lz := 20;);
    END_CALL;

   void4 := oriented_closed_shell('void4', hollow4, FALSE) ;

(* tetra_with_voids2 defined with interference between inner voids *)

    tetra_with_voids2 := faceted_brep (shell_object) ||
                         brep_with_voids ([void2, void4]) ;

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


    fbsr1 := faceted_brep_shape_representation
             ('fbsr1', [tetra_with_void], its_context ) ;

    fbsr2 := faceted_brep_shape_representation
             ('fbsr2', [tetra_with_voids1], its_context ) ;

    fbsr3 := faceted_brep_shape_representation
             ('fbsr3',  [tetra_with_voids2], its_context ) ;

    shape1_def_rep3 := shape_definition_representation
                         (shape_1_def, fbsr1 );


  END_REALIZATION;
END_TEST_CASE;
(*
```

E.1.1.2 Postprocessor verdict criteria

AIM206: The complex subtype faceted_brep and brep_with_voids should be correctly interpreted with a single void, void shell shall not intersect outer, this intersection should be detected in fbsr1.

AIM207: faceted_brep with more than one void shall be correctly interpreted, void shells shall not intersect each other, or outer,
intersection of void shells should be detected in fbsr2,
nesting of void shells to create an illegal instance should be detected in fbsr3.

AIM208: Normal to each void shell shall point into void.

Annex F
(informative)

Example ISO 10303-21 file

The physical file documented in this annex corresponds to Abstract test case FB1, and has been generated from the EXPRESS-I definitions.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION ( ('example ATC for STEP-305 CD ballot'), '1') ;
FILE_NAME ('ATC.S21', '1994-04-01T00:00:01',
          ('JO', 'RJG'),
   ('CAE Group', 'Dept of Mech Eng', 'The University_of_Leeds', 'UK'),
   'Lupine++', 'Lupine++', 'authorisation for sending file: unknown');
FILE_SCHEMA (('part_204_brep_product_schema')) ;
ENDSEC;
DATA;
#51 = SHAPE_DEFINITION_REPRESENTATION ( #42, #50 ) ;
#50 = FACETED_BREP_SHAPE_REPRESENTATION ( 'fbsr1', ( #49 ), #48 ) ;
#49 = FACETED_BREP ( 'tetrahedron', #46 ) ;
#48 = ( GEOMETRIC_REPRESENTATION_CONTEXT ( 'context_1',
      'context_for_tetrahedron',  3 )
        GLOBAL_UNIT_ASSIGNED_CONTEXT ( ( #47) ) ) ;
#47 = ( LENGTH_UNIT ( ) SI_UNIT ( 'milli', 'metre' ) ) ;
#46 = CLOSED_SHELL ( 'tetrashell', ( #26, #27, #28, #29 ) ) ;
#42 = PRODUCT_DEFINITION_SHAPE ( 'test_shape_property',
                           'shape property of test part 1', #40 ) ;
#40 = PRODUCT_DEFINITION ( 'pd01', 'test product definition 1',
                                     #39, #37 ) ;
#39 = PRODUCT_DEFINITION_FORMATION ( 'pdf01', 'test version 1', #38 );
#38 = PRODUCT ( 'p01', 'prod1_name', 'simple test product', ( #36 ) );
#37 = DESIGN_CONTEXT ( 'pt_204_prod_def', #31, 'design' ) ;
#36 = MECHANICAL_CONTEXT ( 'pt_204_product', #31, 'mechanical' ) ;
#31 = APPLICATION_CONTEXT ( 'ap 204 test' ) ;
#30 = APPLICATION_PROTOCOL_DEFINITION ( 'dis',
          'part_204_brep_product_shema', 2000, #31) ;
#29 = FACE_SURFACE ( 'fs4', ( #25 ), #21, .TRUE. ) ;
#28 = FACE_SURFACE ( 'fs3', ( #24 ), #20, .TRUE. ) ;
#27 = FACE_SURFACE ( 'fs2', ( #23 ), #19, .TRUE. ) ;
#26 = FACE_SURFACE ( 'fs1', ( #22 ), #18, .TRUE. ) ;
#25 = FACE_OUTER_BOUND ( 'b4', #13, .TRUE. ) ;
#24 = FACE_OUTER_BOUND ( 'b3', #12, .TRUE. ) ;
#23 = FACE_OUTER_BOUND ( 'b2', #11, .TRUE. ) ;
#22 = FACE_OUTER_BOUND ( 'b1', #10, .TRUE. ) ;
```

```
#21 = PLANE ( 'p4', #17 ) ;
#20 = PLANE ( 'p3', #16 ) ;
#19 = PLANE ( 'p2', #15 ) ;
#18 = PLANE ( 'p1', #14 ) ;
#17 = AXIS2_PLACEMENT_3D ( 'a4', #2, #8, #9 ) ;
#16 = AXIS2_PLACEMENT_3D ( 'a3', #1, #7, #6 ) ;
#15 = AXIS2_PLACEMENT_3D ( 'a2', #1, #6, #5 ) ;
#14 = AXIS2_PLACEMENT_3D ( 'a1', #1, #5, #6 ) ;
#13 = POLY_LOOP ( 'loop_slope', ( #4, #2, #3 ) ) ;
#12 = POLY_LOOP ( 'loop_z', ( #1, #3, #2 ) ) ;
#11 = POLY_LOOP ( 'loop_y', ( #1, #2, #4 ) ) ;
#10 = POLY_LOOP ( 'loop_x', ( #1, #4, #3 ) ) ;
#9 = DIRECTION ( 'dperp', (  1, -1,  0 ) ) ;
#8 = DIRECTION ( 'dslope', (  1,  1,  1 ) ) ;
#7 = DIRECTION ( 'neg_z', (  0,  0, -1 ) ) ;
#6 = DIRECTION ( 'neg_y', (  0, -1,  0 ) ) ;
#5 = DIRECTION ( 'neg_x', ( -1,  0,  0 ) ) ;
#4 = CARTESIAN_POINT ( 'p_z', (  0,  0,  100 ) ) ;
#3 = CARTESIAN_POINT ( 'p_y', (  0,  100,  0 ) ) ;
#2 = CARTESIAN_POINT ( 'p_x', (  100,  0,  0 ) ) ;
#1 = CARTESIAN_POINT ( 'origin', (  0,  0,  0 ) ) ;
ENDSEC;
END-ISO-10303-21;
```

Index

ICS 25.040.40

Price based on 309 pages