INTERNATIONAL STANDARD ISO 10303-42:1994
TECHNICAL CORRIGENDUM 2

Published 1999-12-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION

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

Part 42:
Integrated generic resources: Geometric and topological representation

TECHNICAL CORRIGENDUM 2

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

Partie 42: Ressources génériques intégrées: Représentation géométrique et topologique

RECTIFICATIF TECHNIQUE 2

Technical Corrigendum 2 to International Standard ISO 10303-42:1994 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data*.

Introduction

This document corrects ISO 10303-42:1994, Product data representation and exchange - Part 42: Integrated generic resources: Geometric and topological representation, as amended by ISO 10303-42:1994/Cor. 1:1999. The corrected document replaces ISO 10303-42:1994, as amended by Technical Corrigendum 1.

The purpose of the modifications to the text of ISO 10303-42:1994 is to correct errors in the EXPRESS definitions likely to cause compilation problems, to include additional EXPRESS constructs required for usage with the corrected EXPRESS definitions, to replace the URL in the annex for the computer-interpretable EXPRESS, and to replace the object identifier for the document and the applicable schema.

ICS 25.040.40 **Ref. No. ISO 10303-42:1994/Cor.2:1999(E)**

Modifications to the text of ISO 10303-42:1994

Clause 4.6.6, p. 99

The EXPRESS specification does not assign correct values to the **u** *variable. Remove the EXPRESS specification and replace with the following:*

EXPRESS specification:

```
*)
FUNCTION base_axis(dim : INTEGER; axis1, axis2, axis3 : direction) :
                                                    LIST [2:3] OF
direction;
  LOCAL
    u      : LIST [2:3] OF direction;
    factor : REAL;
    d1, d2 : direction;
  END_LOCAL;

  IF (dim = 3) THEN
    d1 := NVL(normalise(axis3),  dummy_gri || direction([0.0,0.0,1.0]));
    d2 := first_proj_axis(d1,axis1);
    u := [d2, second_proj_axis(d1,d2,axis2), d1];
  ELSE
     IF EXISTS(axis1) THEN
      d1 := normalise(axis1);
      u := [d1, orthogonal_complement(d1)];
      IF EXISTS(axis2) THEN
        factor := dot_product(axis2,u[2]);
        IF (factor < 0.0) THEN
          u[2].direction_ratios[1] := -u[2].direction_ratios[1];
          u[2].direction_ratios[2] := -u[2].direction_ratios[2];
        END_IF;
      END_IF;
    ELSE
      IF EXISTS(axis2) THEN
        d1 := normalise(axis2);
        u := [orthogonal_complement(d1), d1];
        u[1].direction_ratios[1] := -u[1].direction_ratios[1];
        u[1].direction_ratios[2] := -u[1].direction_ratios[2];
      ELSE
u := [dummy_gri || direction([1.0, 0.0]), dummy_gri ||
direction([0.0, 1.0])];
      END_IF;
    END_IF;
  END_IF;
  RETURN(u);
END_FUNCTION;
(*
```

Clause 4.6.7, p. 100

The EXPRESS specification does not assign correct values to the return variable. Remove the EXPRESS specification and replace with the following:

EXPRESS specification:

```
*)
 FUNCTION build_2axes(ref_direction : direction) : LIST [2:2] OF direction;
   LOCAL
     d : direction := NVL(normalise(ref_direction),
                          dummy_gri || direction([1.0,0.0]));
   END_LOCAL;

   RETURN([d, orthogonal_complement(d)]);
 END_FUNCTION;
(*
```

Clause 4.6.8, p. 100

The EXPRESS specification does not assign correct values to the return variable. Remove the EXPRESS specification and replace with the following:

EXPRESS specification:

```
*)
FUNCTION build_axes(axis, ref_direction : direction) :
                                          LIST [3:3] OF direction;
   LOCAL
     d1, d2 : direction;
   END_LOCAL;
   d1 := NVL(normalise(axis), dummy_gri || direction([0.0,0.0,1.0]));
   d2 := first_proj_axis(d1, ref_direction);
   RETURN([d2, normalise(cross_product(d1,d2)).orientation, d1]);
END_FUNCTION;
(*
```

Clause 4.6.9, p. 101

The EXPRESS specification does not assign correct values to the **result** *variable. Remove the EXPRESS specification and replace with the following:*

EXPRESS specification:

```
*)
 FUNCTION orthogonal_complement(vec : direction) : direction;
   LOCAL
```

```
    result :  direction ;
  END_LOCAL;

  IF (vec.dim <> 2) OR NOT EXISTS (vec) THEN
    RETURN(?);
  ELSE
    result := dummy_gri || direction([-vec.direction_ratios[2], vec.direction_ratios[1]]);
    RETURN(result);
  END_IF;
END_FUNCTION;
(*
```

Clause 4.6.12, p. 103

The EXPRESS specification does not assign correct values to the **res** *and* **result** *variables. Remove the EXPRESS specification and replace with the following:*

EXPRESS specification:

```
*)
FUNCTION cross_product (arg1, arg2 : direction) : vector;
  LOCAL
    mag    : REAL;
    res    : direction;
    v1,v2  : LIST[3:3] OF REAL;
    result : vector;
  END_LOCAL;

  IF ( NOT EXISTS (arg1) OR (arg1.dim = 2)) OR
     ( NOT EXISTS (arg2) OR (arg2.dim = 2)) THEN
    RETURN(?);
  ELSE
    BEGIN
      v1  := normalise(arg1).direction_ratios;
      v2  := normalise(arg2).direction_ratios;
      res := dummy_gri || direction([(v1[2]*v2[3] - v1[3]*v2[2]),
    (v1[3]*v2[1] - v1[1]*v2[3]), (v1[1]*v2[2] - v1[2]*v2[1])]);
      mag := 0.0;
      REPEAT i := 1 TO 3;
        mag := mag + res.direction_ratios[i]*res.direction_ratios[i];
      END_REPEAT;
      IF (mag > 0.0) THEN
        result := dummy_gri || vector(res, SQRT(mag));
      ELSE
        result := dummy_gri || vector(arg1, 0.0);
      END_IF;
      RETURN(result);
    END;
  END_IF;
END_FUNCTION;
```

(*

Clause 4.6.14, p. 105

The EXPRESS specification does not assign correct values to the **v** *variable and may corrupt the input parameter* **arg**. *Remove the EXPRESS specification and replace with the following:*

EXPRESS specification:

```
*)
FUNCTION normalise (arg : vector_or_direction) : vector_or_direction;
   LOCAL
     ndim   : INTEGER;
     v      : direction;
     result : vector_or_direction;
     vec    : vector;
     mag    : REAL;
   END_LOCAL;

   IF NOT EXISTS (arg) THEN
     result := ?;
     (* When function is called with invalid data a NULL result is returned *)
   ELSE
     ndim := arg.dim;
     IF 'GEOMETRY_SCHEMA.VECTOR' IN TYPEOF(arg) THEN
       BEGIN
             v := dummy_gri || direction(arg.orientation.direction_ratios);
         IF arg.magnitude = 0.0 THEN
           RETURN(?);
         ELSE
          vec := dummy_gri || vector (v, 1.0);
         END_IF;
       END;
     ELSE
       v := dummy_gri || direction (arg.direction_ratios);
     END_IF;
     mag := 0.0;
     REPEAT  i := 1 TO ndim;
       mag := mag + v.direction_ratios[i]*v.direction_ratios[i];
     END_REPEAT;
     IF mag > 0.0 THEN
       mag := SQRT(mag);
       REPEAT  i := 1 TO ndim;
         v.direction_ratios[i] := v.direction_ratios[i]/mag;
       END_REPEAT;
       IF 'GEOMETRY_SCHEMA.VECTOR' IN TYPEOF(arg) THEN
         vec.orientation := v;
         result := vec;
       ELSE
         result := v;
```

```
      END_IF;
    ELSE
      RETURN(?);
    END_IF;
  END_IF;
  RETURN (result);
END_FUNCTION;
(*
```

Clause 4.6.15, p. 107

The EXPRESS specification does not assign correct values to the **result** *variable. Remove the EXPRESS specification and replace with the following:*

EXPRESS specification:

```
*)
FUNCTION scalar_times_vector (scalar : REAL; vec : vector_or_direction)
                                     : vector;
  LOCAL
    v      : direction;
    mag    : REAL;
    result : vector;
  END_LOCAL;

  IF NOT EXISTS (scalar) OR NOT EXISTS (vec) THEN
    RETURN (?) ;
   ELSE
    IF 'GEOMETRY_SCHEMA.VECTOR' IN TYPEOF (vec) THEN
      v   := dummy_gri || direction(vec.orientation.direction_ratios);
      mag := scalar * vec.magnitude;
    ELSE
      v   := dummy_gri || direction(vec.direction_ratios);
      mag := scalar;
    END_IF;
    IF (mag < 0.0 ) THEN
      REPEAT i := 1 TO SIZEOF(v.direction_ratios);
        v.direction_ratios[i] := -v.direction_ratios[i];
      END_REPEAT;
      mag := -mag;
    END_IF;
    result := dummy_gri || vector(normalise(v), mag);
  END_IF;
  RETURN (result);
END_FUNCTION;
(*
```

Clause 4.6.16, p. 108

The EXPRESS specification does not assign correct values to the **res** *and* **result** *variables. Remove the EXPRESS specification and replace with the following:*

EXPRESS specification:

```
*)
FUNCTION vector_sum(arg1, arg2 : vector_or_direction) : vector;
  LOCAL
    result          : vector;
    res, vec1, vec2 : direction;
    mag, mag1, mag2 : REAL;
    ndim            : INTEGER;
  END_LOCAL;

  IF ((NOT EXISTS (arg1)) OR (NOT EXISTS (arg2))) OR (arg1.dim <> arg2.dim)
      THEN
    RETURN (?) ;

  ELSE
    BEGIN
      IF 'GEOMETRY_SCHEMA.VECTOR' IN TYPEOF(arg1) THEN
        mag1 := arg1.magnitude;
        vec1 := arg1.orientation;
      ELSE
        mag1 := 1.0;
        vec1 := arg1;
      END_IF;
      IF 'GEOMETRY_SCHEMA.VECTOR' IN TYPEOF(arg2) THEN
        mag2 := arg2.magnitude;
        vec2 := arg2.orientation;
      ELSE
        mag2 := 1.0;
        vec2 := arg2;
      END_IF;
      vec1 := normalise (vec1);
      vec2 := normalise (vec2);
      ndim := SIZEOF(vec1.direction_ratios);
      mag := 0.0;
      res := dummy_gri || direction(vec1.direction_ratios);
      REPEAT i := 1 TO ndim;
        res.direction_ratios[i] := mag1*vec1.direction_ratios[i] +
                                   mag2*vec2.direction_ratios[i];
        mag := mag + (res.direction_ratios[i]*res.direction_ratios[i]);
      END_REPEAT;
      IF (mag > 0.0 ) THEN
      result := dummy_gri || vector( res, SQRT(mag));
      ELSE
        result := dummy_gri || vector( vec1,  0.0);
      END_IF;
    END;
  END_IF;
  RETURN (result);
END_FUNCTION;
(*
```

Clause 4.6.17, p. 109

The EXPRESS specification does not assign correct values to the **res** *variable. Remove the EXPRESS specification and replace with the following:*

EXPRESS specification:

```
*)
FUNCTION vector_difference(arg1, arg2 : vector_or_direction) : vector;
  LOCAL
    result          : vector;
    res, vec1, vec2 : direction;
    mag, mag1, mag2 : REAL;
    ndim            : INTEGER;
  END_LOCAL;

  IF ((NOT EXISTS (arg1)) OR (NOT EXISTS (arg2))) OR (arg1.dim <> arg2.dim)
      THEN
    RETURN (?) ;
   ELSE
    BEGIN
      IF 'GEOMETRY_SCHEMA.VECTOR' IN TYPEOF(arg1) THEN
        mag1 := arg1.magnitude;
        vec1 := arg1.orientation;
      ELSE
        mag1 := 1.0;
        vec1 := arg1;
      END_IF;
      IF 'GEOMETRY_SCHEMA.VECTOR' IN TYPEOF(arg2) THEN
        mag2 := arg2.magnitude;
        vec2 := arg2.orientation;
      ELSE
        mag2 := 1.0;
        vec2 := arg2;
      END_IF;
      vec1 := normalise (vec1);
      vec2 := normalise (vec2);
      ndim := SIZEOF(vec1.direction_ratios);
      mag := 0.0;
      res := dummy_gri || direction(vec1.direction_ratios);
      REPEAT i := 1 TO ndim;
        res.direction_ratios[i] := mag1*vec1.direction_ratios[i] +
                                   mag2*vec2.direction_ratios[i];
        mag := mag + (res.direction_ratios[i]*res.direction_ratios[i]);
      END_REPEAT;
      IF (mag > 0.0 ) THEN
      result := dummy_gri || vector( res, SQRT(mag));
      ELSE
        result := dummy_gri || vector( vec1,  0.0);
      END_IF;
    END;
```

```
  END_IF;
  RETURN (result);
END_FUNCTION;
(*
```

Clause 4.6.18, p. 110

The EXPRESS specification does not assign correct values to the **knot_mult** *variable. Remove the EXPRESS specification and replace with the following:*

EXPRESS specification:

```
*)
 FUNCTION default_b_spline_knot_mult(degree, up_knots : INTEGER;
                                     uniform : knot_type)
                                             : LIST [2:?] OF INTEGER;
   LOCAL
     knot_mult : LIST [1:up_knots] OF INTEGER;
   END_LOCAL;

   IF uniform = uniform_knots THEN
     knot_mult := [1:up_knots];
   ELSE
     IF uniform = quasi_uniform_knots THEN
       knot_mult := [1:up_knots];
       knot_mult[1] := degree + 1;
       knot_mult[up_knots] := degree + 1;
     ELSE
       IF uniform = piecewise_bezier_knots THEN
         knot_mult := [degree:up_knots];
         knot_mult[1] := degree + 1;
         knot_mult[up_knots] := degree + 1;
       ELSE
         knot_mult := [0:up_knots];
       END_IF;
     END_IF;
   END_IF;
   RETURN(knot_mult);
 END_FUNCTION;
(*
```

Clause 4.6.19, p. 111

The EXPRESS specification does not assign correct values to the **knots** *variable. Remove the EXPRESS specification and replace with the following:*

EXPRESS specification:

```
*)
```

```
FUNCTION default_b_spline_knots(degree,up_knots : INTEGER;
                                uniform : knot_type)
                                        : LIST [2:?] OF parameter_value;
  LOCAL
    knots  : LIST [1:up_knots] OF parameter_value := [0:up_knots];
    ishift : INTEGER := 1;
  END_LOCAL;

  IF (uniform = uniform_knots) THEN
     ishift := degree + 1;
  END_if;
  IF (uniform = uniform_knots) OR
     (uniform = quasi_uniform_knots) OR
     (uniform = piecewise_bezier_knots) THEN

    REPEAT i := 1 TO up_knots;
      knots[i] := i - ishift;
    END_REPEAT;
  END_IF;
  RETURN(knots);
END_FUNCTION;
(*
```

Clause 4.6.20, p. 112

The EXPRESS specification does not assign correct values to the return variable. Remove the EXPRESS specification and replace with the following:

EXPRESS specification:

```
*)
 FUNCTION default_b_spline_curve_weights(up_cp : INTEGER)
                                         : ARRAY [0:up_cp] OF REAL;
   RETURN([1:up_cp + 1]);
 END_FUNCTION;
(*
```

Clause 4.6.21, p. 113

The EXPRESS specification does not assign correct values to the return variable. Remove the EXPRESS specification and replace with the following:

EXPRESS specification:

```
*)
FUNCTION default_b_spline_surface_weights(u_upper, v_upper: INTEGER)
                                       : ARRAY [0:u_upper] OF
                                         ARRAY [0:v_upper] OF REAL;
   RETURN([[1:v_upper + 1]:u_upper +1]);
```

```
END_FUNCTION;
(*
```

Clause 4.6.22, p. 114

The EXPRESS specification contained some redundant variable declarations. Remove the EXPRESS specification and replace with the following:

EXPRESS specification:

```
*)
FUNCTION constraints_param_b_spline(degree, up_knots, up_cp : INTEGER;
                                                    knot_mult : LIST OF INTEGER;
                          knots : LIST OF parameter_value) : BOOLEAN;
   LOCAL
     result  : BOOLEAN := TRUE;
     k, sum  : INTEGER;
   END_LOCAL;

   (* Find sum of knot multiplicities. *)
   sum := knot_mult[1];

   REPEAT i := 2 TO up_knots;
     sum := sum + knot_mult[i];
   END_REPEAT;

   (* Check limits holding for all B-spline parametrisations *)
   IF (degree < 1) OR (up_knots < 2) OR (up_cp < degree) OR
         (sum <> (degree + up_cp + 2)) THEN
     result := FALSE;
     RETURN(result);
   END_IF;

   k := knot_mult[1];

   IF (k < 1) OR (k > degree + 1) THEN
     result := FALSE;
     RETURN(result);
   END_IF;

   REPEAT i := 2 TO up_knots;
     IF (knot_mult[i] < 1) OR (knots[i] <= knots[i-1]) THEN
       result := FALSE;
       RETURN(result);
     END_IF;

     k := knot_mult[i];

     IF (i < up_knots) AND (k > degree) THEN
       result := FALSE;
```

```
      RETURN(result);
    END_IF;

    IF (i = up_knots) AND (k > degree + 1) THEN
      result := FALSE;
      RETURN(result);
    END_IF;
  END_REPEAT;
  RETURN(result);
END_FUNCTION;
(*
```

Clause 4.6.28, p. 119

The EXPRESS specification does not correctly initialise the **res** *variable. Remove the EXPRESS specification and replace with the following:*

EXPRESS specification:

```
*)
 FUNCTION list_to_array(lis : LIST [0:?] OF GENERIC : T;
                        low,u : INTEGER) : ARRAY OF GENERIC : T;
   LOCAL
     n   : INTEGER;
     res : ARRAY [low:u] OF GENERIC : T;
   END_LOCAL;

   n := SIZEOF(lis);
   IF (n <> (u-low +1)) THEN
     RETURN(?);
   ELSE
     res := [lis[1] : n];
     REPEAT i := 2 TO n;
       res[low+i-1] := lis[i];
     END_REPEAT;
     RETURN(res);
   END_IF;
 END_FUNCTION;
(*
```

Clause 4.6.29, p. 120

The EXPRESS specification does not assign correct values to the **res** *variable. Remove the EXPRESS specification and replace with the following:*

EXPRESS specification:

```
*)
 FUNCTION make_array_of_array(lis : LIST[1:?] OF LIST [1:?] OF GENERIC : T;
```

```
                              low1, u1, low2, u2 : INTEGER):
                  ARRAY OF ARRAY OF GENERIC : T;
   LOCAL
     res   : ARRAY[low1:u1] OF ARRAY [low2:u2] OF GENERIC : T;
   END_LOCAL;

(* Check input dimensions for consistency *)
   IF (u1-low1+1) <> SIZEOF(lis) THEN
     RETURN (?);
   END_IF;
   IF (u2 - low2 + 1 ) <> SIZEOF(lis[1]) THEN
     RETURN (?) ;
   END_IF;
(* Initialise res with values from lis[1] *)
   res := [list_to_array(lis[1], low2, u2) : (u1-low1 + 1)];
   REPEAT i := 2 TO HIINDEX(lis);
     IF (u2-low2+1) <> SIZEOF(lis[i]) THEN
       RETURN (?);
     END_IF;
     res[low1+i-1] := list_to_array(lis[i], low2, u2);
   END_REPEAT;

   RETURN (res);
 END_FUNCTION;
(*
```

Clause 5.3, p. 127

A constant is required to initialise ENTITYs constructed within ENTITY definitions or FUNCTIONs. The CONSTANT **dummy_gri** *is a partial definition to be used when types of* **geometric_representation_item** *are constructed. Add the following new clause 5.3.1 and renumber existing clauses 5.3.1 through 5.3.5 accordingly.*

5.3.1 Constant definition

The constant **dummy_tri** is a partial entity definition to be used when types of **topological_representation_item** are constructed. It provides the correct supertypes and the **name** attribute as an empty string.

EXPRESS specification:

```
*)
 CONSTANT
    dummy_tri : topological_representation_item := representation_item('')||
                  topological_representation_item();
 END_CONSTANT;
(*
```

Clause 5.5.3, p. 154

The EXPRESS specification does not give a correct initialisation of the **the_reverse** *variable. Additional clarification to values of the attribute are given. The declared types for the return variable and the local variable are changed to the subtype actually returned.*

Remove:
This function returns an **edge** equivalent to the input **edge** except that the orientation is reversed.

Replace with:
This function returns an **oriented_edge** equivalent to the input **edge** except that the orientation is reversed.

Remove the EXPRESS specification and replace with the following:

EXPRESS specification:

```
*)
FUNCTION edge_reversed (an_edge : edge) : oriented_edge;
  LOCAL
    the_reverse : oriented_edge;
  END_LOCAL;

  IF ('TOPOLOGY_SCHEMA.ORIENTED_EDGE' IN TYPEOF (an_edge) ) THEN
    the_reverse := dummy_tri ||
            edge(an_edge.edge_end, an_edge.edge_start) ||
            oriented_edge(an_edge\oriented_edge.edge_element,
                       NOT (an_edge\oriented_edge.orientation)) ;
  ELSE
    the_reverse := dummy_tri ||
            edge(an_edge.edge_end, an_edge.edge_start) ||
            oriented_edge(an_edge, FALSE);
  END_IF;
  RETURN (the_reverse);
END_FUNCTION;
(*
```

Remove the argument definition and replace with the following:

the_reverse: (output) The **oriented_edge** that is the result of the orientation reversal.

Clause 5.5.4, p. 155
The EXPRESS specification does not give a correct initialisation of the **the_reverse** *variable. The declared types for the return variable and the local variable are changed to the subtype actually returned.*

Remove:
This function returns a **path** equivalent to the input **path** except that the orientation is reversed.

Replace with:
This function returns an **oriented_path** equivalent to the input **path** except that the orientation is reversed.

Remove the EXPRESS specification and replace with the following:

EXPRESS specification:

```
*)
FUNCTION path_reversed (a_path : path) : oriented_path;
  LOCAL
    the_reverse : oriented_path ;
  END_LOCAL;
  IF ('TOPOLOGY_SCHEMA.ORIENTED_PATH' IN TYPEOF (a_path) ) THEN
    the_reverse := dummy_tri ||
       path(list_of_topology_reversed (a_path.edge_list)) ||
          oriented_path(a_path\oriented_path.path_element,
                        NOT(a_path\oriented_path.orientation)) ;
  ELSE
    the_reverse := dummy_tri ||
                   path(list_of_topology_reversed (a_path.edge_list)) ||
                       oriented_path(a_path, FALSE);
  END_IF;

  RETURN (the_reverse);
END_FUNCTION;
(*
```

Remove the argument definition and replace with the following:

the_reverse: (output) The **oriented_path** which is the result of the orientation reversal.

Clause 5.5.5, p. 155
The EXPRESS specification does not give a correct initialisation of the **the_reverse** *variable, the return value is of type* **face_outer_bound** *when* **a_face_bound** *is of this type.*

Remove the EXPRESS specification and replace with the following:

EXPRESS specification:

```
*)
FUNCTION face_bound_reversed (a_face_bound : face_bound) : face_bound;
  LOCAL
    the_reverse : face_bound ;
  END_LOCAL;
  IF ('TOPOLOGY_SCHEMA.FACE_OUTER_BOUND' IN TYPEOF (a_face_bound) ) THEN
```

```
    the_reverse := dummy_tri ||
                     face_bound(a_face_bound\face_bound.bound,
                          NOT (a_face_bound\face_bound.orientation))
                           || face_outer_bound() ;
  ELSE
    the_reverse := dummy_tri ||
               face_bound(a_face_bound.bound, NOT(a_face_bound.orientation));
  END_IF;
 RETURN (the_reverse);
END_FUNCTION;
(*
```

Clause 5.5.6, p. 156

The EXPRESS specification does not give a correct initialisation of the **the_reverse** *variable. The declared types for the return variable and the local variable are changed to the subtype actually returned.*

Remove:

This function returns a **face** equivalent to input **face** except that the orientation is reversed.

Replace with:

This function returns an **oriented_face** equivalent to input **face** except that the orientation is reversed.

Remove the EXPRESS definition and replace with the following:

EXPRESS specification:

```
*)
FUNCTION face_reversed (a_face : face) : oriented_face;
  LOCAL
    the_reverse : oriented_face ;
  END_LOCAL;
  IF ('TOPOLOGY_SCHEMA.ORIENTED_FACE' IN TYPEOF (a_face) ) THEN
    the_reverse := dummy_tri ||
      face(set_of_topology_reversed(a_face.bounds)) ||
       oriented_face(a_face\oriented_face.face_element,
                          NOT (a_face\oriented_face.orientation)) ;
  ELSE
    the_reverse := dummy_tri ||
      face(set_of_topology_reversed(a_face.bounds)) ||
                              oriented_face(a_face, FALSE) ;
  END_IF;
     RETURN (the_reverse);
END_FUNCTION;
(*
```

Remove the argument definition and replace with the following:

the_reverse: (output) The **oriented_face** which is the result of the orientation reversal.

Clause 5.5.7, p. 156

The EXPRESS specification does not give a correct initialisation of the **the_reverse** *variable and intermediate values of the variable. The structure of the function is simplified by making use of two new simpler functions.*

Remove:

This function returns a **shell** equivalent to the input **shell** except that the orientation is reversed.

Replace with:

This function returns an **oriented_open_shell** or **oriented_closed_shell** equivalent to the input **shell** except that the orientation is reversed.

Remove the EXPRESS specification and replace with the following:

EXPRESS specification:

```
*)
 FUNCTION shell_reversed (a_shell : shell) : shell;
   IF ('TOPOLOGY_SCHEMA.OPEN_SHELL' IN TYPEOF (a_shell) ) THEN
     RETURN (open_shell_reversed (a_shell));
   ELSE
     IF ('TOPOLOGY_SCHEMA.CLOSED_SHELL' IN TYPEOF (a_shell) ) THEN
       RETURN (closed_shell_reversed (a_shell));
     ELSE
       RETURN (?);
     END_IF;
   END_IF;
 END_FUNCTION;
(*
```

Clause 5.5.8, p. 157

With the changes identified in this Technical Corrigendum additional topology_schema functions are required. Re-number existing clauses 5.5.8 to 5.5.20 as 5.5.10 to 5.5.22 respectively. Add the following functions as clauses 5.5.8 and 5.5.9.

5.5.8 closed_shell_reversed

This function returns an **oriented_closed_shell** or equivalent to the input **closed_shell** except that the orientation is reversed.

EXPRESS specification:

```
*)
```

```
FUNCTION closed_shell_reversed (a_shell : closed_shell) :
                                      oriented_closed_shell;
 LOCAL
   the_reverse : oriented_closed_shell;
 END_LOCAL;
  IF ('TOPOLOGY_SCHEMA.ORIENTED_CLOSED_SHELL' IN TYPEOF (a_shell) ) THEN
     the_reverse := dummy_tri ||
                    connected_face_set (
                       a_shell\connected_face_set.cfs_faces) ||
                    closed_shell () || oriented_closed_shell(
                       a_shell\oriented_closed_shell.closed_shell_element,
                       NOT(a_shell\oriented_closed_shell.orientation));
  ELSE
     the_reverse := dummy_tri ||
                   connected_face_set (
                     a_shell\connected_face_set.cfs_faces) ||
                   closed_shell () || oriented_closed_shell (a_shell, FALSE);
  END_IF;
  RETURN (the_reverse);
END_FUNCTION;
(*
```

Argument definitions:

a_shell: (input) The **closed_shell** which is to have its orientation reversed.

the_reverse: (output) The result of the orientation reversal.

5.5.9 open_shell_reversed

This function returns an **oriented_open_shell** or equivalent to the input **open_shell** except that the orientation is reversed.

EXPRESS specification:

```
*)
 FUNCTION open_shell_reversed ( a_shell : open_shell) :
                                        oriented_open_shell;
   LOCAL
     the_reverse : oriented_open_shell;
   END_LOCAL;
   IF ('TOPOLOGY_SCHEMA.ORIENTED_OPEN_SHELL' IN TYPEOF (a_shell) ) THEN
     the_reverse := dummy_tri ||
                    connected_face_set (
                        a_shell\connected_face_set.cfs_faces) ||
                    open_shell () || oriented_open_shell(
                      a_shell\oriented_open_shell.open_shell_element,
```