INTERNATIONAL STANDARD ISO 10303-42:2003
TECHNICAL CORRIGENDUM 1

Published 2007-07-15

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

Part 42:
Integrated generic resource: Geometric and topological representation

TECHNICAL CORRIGENDUM 1

Systèmes d'automatisation industrielle et intégration — Représentation et échange de données de produits — Partie 42: Ressource générique intégrée: Représentation géométrique et topologique

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to ISO 10303-42:2003 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data.*

ICS 25.040.40 **Ref. No. ISO 10303-42:2003/Cor.1:2007(E)**

Introduction

This Technical Corrigendum applies to ISO 10303-42:2003.

The purpose of the modifications to the text of ISO 10303-42:2003 is to correct errors in EXPRESS entity and function definitions and interface specification, to add further explanatory text to the definition of transformations, to clarify the interpretation of angular measures, and to update the normative references.

NOTE 1 All the additions and amendments contained in this document are also applicable to ISO 10303-42:1994 as amended by TC3.

Modifications to the text of ISO 10303-42:2003

Clause 2 p. 3

The first normative reference has been withdrawn and replaced by a later edition. Delete the first normative reference (ISO/IEC 8824-1) and replace with:

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

Clause 4.2.3, p. 15

The interpretation of parameters that are related to angular measures requires further clarification. Add the following paragraph at the end of clause 4.2.3.

Where the curve or surface parameterisation uses trigonometric functions, the parameter for the function behaves like an angle and can be considered to be an angular parameter. Numerical values for such angular parameters shall use the current units for **plane_angle_measure**.

Clause 4.4.20 **cartesian_transformation_operator,** *p. 42*

The use of transformation operators that introduce mirroring or reflection requires further explanatory text. Replace the following paragraph
For those entities whose attributes include an **axis2_placement**, *the transformation is applied, after the derivation, to the derived attributes* **p** *defining the placement coordinate* **directions***. For a transformed* **surface**, *the direction of the surface normal at any point is obtained by transforming the normal, at the corresponding point, to the original* **surface***. For geometric entities with attributes (such as the radius of a circle) which have the dimensionality of length, the values will be multiplied by* S.
with:

For those entities whose attributes include an **axis2_placement**, the transformation is applied, after the derivation, to the derived attributes **p** defining the placement coordinate **directions**. For a transformed **surface**, the direction of the surface normal at any point is obtained by transforming the normal, at the corresponding point, to the original **surface**. The parametrisation of the transformed surface is defined using the transformed value of **p** as defined above. For geometric entities with attributes (such as the radius of a circle) which have the dimensionality of length, the values will be multiplied by S.

For transformations involving reflection or mirroring, with $|\mathbf{T}| = \ 1$, the relationship between the sense of the boundary and the interior of a **curve_bounded_surface** or **face_surface** is affected.
For a **curve_bounded_surface** if **n** is the direction of the surface normal and **t** is the direction of the tangent vector at a point on the boundary after transformation, then the interior is in the direction $|\mathbf{T}|\mathbf{n}\times\mathbf{t}$.
For a **face** or **face_surface**, if $|\mathbf{T}| = \ 1$ the interior of the transformed face will lie to the right when traversing the bounding loops in the positive sense.

Clause 4.4.26 circle, p. 49

To clarify the use of angular measures as parameters replace:
The parametrisation range is 0 $\leq u \leq$ 360 degrees
with:

The parametrisation range is 0 $\leq u \leq$ 360 degrees, u is an angular parameter and when a numerical value is specified it shall use the current units for plane_angle_measure.

Clause 4.4.27 ellipse, p 50.

To clarify the use of angular measures as parameters replace:
The parametrisation range is 0 $\leq u \leq$ 360 degrees
with:

The parametrisation range is 0 $\leq u \leq$ 360 degrees, u is an angular parameter and when a numerical value is specified it shall use the current units for plane_angle_measure.

Clause 4.4.57 cylindrical_surface, p 87.

To clarify the use of angular measures as parameters add the following sentence after
In the above parameterisation, the length unit for the unit vector **z** *is equal to that of the* **radius***.*

u is an angular parameter and when a numerical value is specified it shall use the current units for **plane_angle_measure**.

Clause 4.4.58 conical_surface, p 88.

To clarify the use of angular measures as parameters add the following sentence after
In the above parameterisation, the length unit for the unit vector **z** *is equal to that of the* **radius***.*

u is an angular parameter and when a numerical value is specified it shall use the current units for **plane_angle_measure**.

Clause 4.4.59 spherical_surface, p 90.

To clarify the use of angular measures as parameters add the following sentence after

where the parametrisation range is 0 $\leq u \leq$ 360 degrees and -90 $\leq v \leq$ 90 degrees.

u and v are angular parameters and when numerical values are specified they shall use the current units for **plane_angle_measure**.

Clause 4.4.60 toroidal_surface, p 91.

To clarify the use of angular measures as parameters add the following sentence after

where the parametrisation range is 0 $\leq u, v \leq$ 360 degrees.

u and v are angular parameters and when numerical values are specified they shall use the current units for **plane_angle_measure**.

Clause 4.4.61 degenerate_toroidal_surface, p 93.
To clarify the use of angular measures as parameters, after the statement
where ϕ *degrees is the angle given by* $rcos(\phi) = \quad R$
Add:

u and v are angular parameters and when numerical values are specified they shall use the current units for **plane_angle_measure**.

Clause 4.4.62 dupin_cyclide_surface, p 95.
To clarify the use of angular measures as parameters, after the sentence
where the domain of parametrisation is $0° \leq u, v \leq 360°$, *and* $\sqrt{}$ *denotes the positive square root.*
add the sentence:

u and v are angular parameters and when numerical values are specified they shall use the current units for **plane_angle_measure**.

Clause 4.4.65 surface_of_revolution, p 95.
To clarify the use of angular measures as parameters, after the sentence
For a **surface_of_revolution** *the parametric range is 0* $\leq u \leq$ *360 degrees*

add the sentence

u is an angular parameter and when a numerical value is specified it shall use the current units for **plane_angle_measure**.

Clause 4.6.5 associated_surface, p 148.
The attribute name **basis_surface** *is not unique in this schema. In order to remove any possible ambiguity a qualified name is now used in the EXPRESS definition of this function. Remove completely the existing EXPRESS definition and replace with:*

EXPRESS specification:

```
*)
FUNCTION associated_surface(arg : pcurve_or_surface) : surface;
  LOCAL
    surf : surface;
  END_LOCAL;

  IF 'GEOMETRY_SCHEMA.PCURVE' IN TYPEOF(arg) THEN
    surf := arg\pcurve.basis_surface;
  ELSE
    surf := arg;
  END_IF;
  RETURN(surf);
END_FUNCTION;
(*
```

Clause 4.6.8 build_axes, p 151.

The attribute name **orientation** *is not unique in this schema. In order to remove any possible ambiguity a qualified name is now used in the EXPRESS definition of this function. Remove completely the existing EXPRESS definition and replace with:*

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))\vector.orientation, d1]);
END_FUNCTION;
(*
```

Clause 4.6.14 normalise, p 157.

The attribute name **orientation** *is not unique in this schema. In order to remove any possible ambiguity a qualified name is now used in the EXPRESS definition of this function. Remove completely the existing EXPRESS definition and replace with:*

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\vector.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 scalar_times_vector, p 158.

The attribute name **orientation** *is not unique in this schema. In order to remove any possible ambiguity a qualified name is now used in the EXPRESS definition of this function. Remove completely the existing EXPRESS definition and replace with:*

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\vector.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 vector_sum, p 160.

The attribute name **orientation** *is not unique in this schema. In order to remove any possible ambiguity a qualified name is now used in the EXPRESS definition of this function. Remove completely the existing EXPRESS definition and replace with:*

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\vector.orientation;
      ELSE
        mag1 := 1.0;
        vec1 := arg1;
      END_IF;
      IF 'GEOMETRY_SCHEMA.VECTOR' IN TYPEOF(arg2) THEN
        mag2 := arg2.magnitude;
        vec2 := arg2\vector.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 vector_difference, p 161.

The attribute name **orientation** *is not unique in this schema. In order to remove any possible ambiguity a qualified name is now used in the EXPRESS definition of this function. There was also a sign error in the formulation of this function, which was not present in earlier editions of this part of ISO 10303. Remove completely the existing EXPRESS definition and replace with:*

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\vector.orientation;
        ELSE
          mag1 := 1.0;
          vec1 := arg1;
        END_IF;
        IF 'GEOMETRY_SCHEMA.VECTOR' IN TYPEOF(arg2) THEN
          mag2 := arg2.magnitude;
          vec2 := arg2\vector.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 5.5.21 edge_curve_pcurves, p 229.

In the return statement for this function RESULT is incorrectly typed in upper case. Remove the EXPRESS specification of this function and replace with:

EXPRESS specification:

```
 *)
 FUNCTION edge_curve_pcurves (an_edge  : edge_curve;
                        the_surface_curves : SET OF surface_curve)
       : SET OF pcurve;
 LOCAL
   a_curve      : curve;
   result       : SET OF pcurve;
   the_geometry : LIST[1:2] OF pcurve_or_surface;
 END_LOCAL;
   a_curve := an_edge.edge_geometry;
   result := [];
   IF 'GEOMETRY_SCHEMA.PCURVE' IN TYPEOF(a_curve) THEN
     result := result + a_curve;
   ELSE
     IF 'GEOMETRY_SCHEMA.SURFACE_CURVE' IN TYPEOF(a_curve) THEN
       the_geometry := a_curve\surface_curve.associated_geometry;
       REPEAT k := 1 TO SIZEOF(the_geometry);
          IF 'GEOMETRY_SCHEMA.PCURVE' IN TYPEOF (the_geometry[k])
          THEN
             result := result + the_geometry[k];
          END_IF;
       END_REPEAT;
     ELSE
       REPEAT j := 1 TO SIZEOF(the_surface_curves);
         the_geometry := the_surface_curves[j].associated_geometry;
         IF the_surface_curves[j].curve_3d :=: a_curve
         THEN
           REPEAT k := 1 TO SIZEOF(the_geometry);
             IF 'GEOMETRY_SCHEMA.PCURVE' IN TYPEOF (the_geometry[k])
             THEN
               result := result + the_geometry[k];
             END_IF;
           END_REPEAT;
         END_IF;
       END_REPEAT;
     END_IF;
   END_IF;

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

Clause 5.5.22 vertex_point_pcurves, p 231.

In the return statement for this function RESULT is incorrectly typed in upper case. Remove the EXPRESS specification of this function and replace with: