INTERNATIONAL STANDARD

ISO 13584-31

First edition
1999-12-15

Industrial automation systems and integration — Parts library —

Part 31: Implementation resources: Geometric programming interface

Systèmes d'automatisation industrielle et intégration — Bibliothèque de composants —

Partie 31: Ressources de mise en application: Interface de programmation géométrique

Reference number
ISO 13584-31:1999(E)

Contents

Page

Figures

Tables

Foreword

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

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.

International Standard ISO 13584-31 was prepared by Technical Committee ISO/TC 184, *Industrial automation system and integration,* Subcommittee SC4, *Industrial data and global manufacturing programming languages.*

ISO 13584 consists of the following parts under the general title *Industrial automation systems and integration - Parts library:*

— Part 1, Overview and fundamental principles;

— Part 10, Conceptual description: Conceptual model of parts library;

— Part 20, Logical resource: Logical model of expressions;

— Part 24, Logical resource: Logical model of supplier library;

— Part 26, Logical resource: Supplier identification;

— Part 31, Implementation resource: Geometric programming interface;

— Part 42, Description methodology: Methodology for structuring part families;

— Part 101, View exchange protocol: Geometric view exchange protocol by parametric program;

— Part 102, View exchange protocol: View exchange protocol by ISO 10303 conforming specification.

The structure of this International Standard is described in ISO 13584-1. The numbering of the parts of this International Standard reflects its structure:

— Parts 10 to 19 specify the conceptual descriptions,

— Parts 20 to 29 specify the logical resources,

— Parts 30 to 39 specfy the implementation resources,

— Parts 40 to 49 specify the description methodology,

— Parts 50 to 59 specify the conformance testing,

— Parts 100 to 199 specify the view exchange protocol,

— Parts 500 to 599 specify the standardised content.

Should further parts of ISO 13584 be published, they will follow the same numbering pattern.

Annexes A and B form an integral part of this part of ISO 13584.

Annex B is for information only.

Introduction

ISO 13584 is an International Standard for the computer-interpretable representation and exchange of part library data. The objective is to provide a neutral mechanism capable of transferring parts library data, independent of any application that is using a parts library data system. The nature of this description makes it suitable not only for the exchange of files containing parts, but also as a basis for implementing and sharing databases of parts library data.

This International Standard is organized as a series of parts, each published separately. The parts of ISO 13854 fall into one of the following series: conceptual descriptions, logical resources, implementation resources, description methodology, conformance testing, view exchange protocol, and standardised content. The series are described in ISO 13584-1. This part of ISO 13584 is a member of the (implementation resources) series .

This part of ISO 13584 specifies an interface to enable the creation of product model data inside an user system from an application program that is independent of the target user system.

This interface may be used, outside the context of standardized parts library data, to permit the development of application programs that are independent of the target CAD system. In the context of ISO 10303, this interface may be implemented on the top of the SDAI interface to provide constrained geometry construction facilities.

In the context of parts library data, conforming to the ISO 13584 Standard series, the product model data creation process is an application program provided by parts library suppliers, that creates geometric model inside the user system. The interface ensures its independancy from the target user system.

Industrial automation systems and integration - Parts Library - Part 31: Implementation resources: Geometric programming interface

1 Scope and field of application

This part of ISO 13584 specifies an application programming interface that enables an application program to generate geometric models that are independent of the target user system. The interface allows portability of programs that describe parametric shape representations of parts families held in an ISO 13584 parts library.

The following are within the scope of this International Standard:

— programs to generate geometric representations within a modelling system that are independent of the target system,

— programs that specify geometric representations that are created through constraint-based geometric definitions,

— programs that structure geometric representations created independently of the target system,

— programs that specify presentation style attributes for symbolic visualisation of representations created,

— programs that support technical drawing standard conventions for shape representation, including a 2D hidden line mechanism.

The following are outside the scope of this International Standard:

— The precise control of the image to be displayed on the receiving system devices,

— The precise definition of the data that shall be created on the receiving system,

— The storage of a parametric model on the receiving system.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO 13584. 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 13584 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 publication referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO 128: 1982	*Technical drawings - General principles of presentation.*
ISO 1539: 1991	*Information technology - Programming languages - FORTRAN.*
ISO/IEC 8824-1 [1]	*Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

[1] To be published

ISO 10303-11: 1994	*Industrial automation systems - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual.*
ISO 10303-41: 1994	*Industrial automation systems - Product data representation and exchange - Part 41: Fundamentals of product description and support.*
ISO 10303-42: 1994	*Industrial automation systems - Product data representation and exchange - Part 42: Integrated resources: Geometric and topological representation.*
ISO 10303-43: 1994	*Industrial automation systems - Product data representation and exchange - Part 43: Integrated resources: Representation structures.*
ISO 10303-46: 1994	*Industrial automation systems - Product data representation and exchange - Part 46: Integrated generic resources: Visual presentation.*
ISO 13584-10	*Industrial automation systems - Parts library - Part 10: Conceptual Model of Parts Library.*

3 Terms, definitions and abbreviations

3.1 Terms defined in ISO 13584-10

For the purpose of this part of ISO 13584 , the following terms defined in ISO 13584-10 apply:

Abstract part ;

CAD system ;

EXPRESS ;

Functional view ;

Library Management System (LMS) ;

Library supplier ;

Part ;

Parts library ;

Parts supplier ;

Product ;

Product data ;

Program ;

Representation of a part ;

Supplier ;

Supplier part ;

Supplier library ;

Structure ;

User ;

User library ;

View ;

View control variable ;

View exchange protocol .

3.2 Other terms and definitions

For the purposes of this part of ISO 13584, the following terms and definitions apply.

3.2.1 application programming interface (API)
set of functions that may be triggered from one program by using the concrete syntax defined in one binding.

3.2.2 binding
description of the concrete syntax that shall be used in a particular programming language to trigger the different functions that constitute an application programming interface.

3.2.3 local coordinate system (LCS)
orthogonal right-handed coordinate system used to orientate and to locate geometrical entities in space. Local Coordinate Systems are modeled by an **axis2_placement** entity.

3.2.4 parameter
variable whose name and type of values are specified.

3.2.5 parametric (shape) model
expression of a parametric shape by means of a data model.

3.2.6 parametric (shape) program
expression of a parametric shape by means of a program referring to an API.

3.2.7 parametric shape
common description of a family of cognate shapes and a set of parameters. A parametric shape specifies a partial function from the domain of the parameters onto the set of shapes.

3.2.8 temporary database
temporary database is a mechanism that permits to store construction or temporary data before transferring to the CAD system.

3.3 Abbreviations

For the purposes of this part of ISO 13584, the following abbreviations apply.

— 2D: Two Dimensional ;

— 3D: Three Dimensional ;

— API: Application Programming Interface ;

— CAD: Computer Aided Design ;

— EPS: Epsilon, see 4.6 ;

— HLI: Hidden Line Involved, see 5.3.5 ;

— LCS: Local Coordinate System ;

— LMS: Library Management System ;

— MAX : Maximal value, see 4.6 ;

— OVC: Object View Coordinate system, see 5.3.1 ;

— SDAI: Standard Data Access Interface ;

— TDB: Temporary Data Base, see 5.3.4.

4 Fundamental concepts

4.1 Requirement for parametrics capabilities

1) The ISO 13584 International Standard shall provide a mechanism that enables the global description of the shapes of all the different parts that belong to the same parts family within an ISO 13584 Library.

EXAMPLE 1 - The ISO 4014 hexagon head bolts Standard [1] specifies thousands of different bolts. Describing separately the shape of each bolt is infeasible.

2) Each global shape description shall be associated with a set of numeric-typed, string-typed or Boolean-typed parameters whose set of values characterise each part of the part family. The mechanism that generates each specific shape out of the global description and from a specific set of values v of the parameters shall be deterministic, i.e., it shall define a partial function f from the domain of the set of parameters D onto the set of shapes S.

$$f : D \mapsto S \,;\, s = f(v).$$

Such a description is called a parametric shape.

EXAMPLE 2 - The global description of the 2D top views of the different bolts of the ISO 4014 Standard may be specified as dependent on two real parameters L and D. For each pair of permitted values (l,d) of L and D, the mechanism shall be able to generate in a deterministic way an unique shape.

3) It is a requirement that a parametric shape shall be specified through graphical user interactions. This implies that the mechanism shall provide for the description of constraint-based geometry, the solver of these constraints being part of the mechanism.

4.2 Exchange format for parametric shape description

1) A program that references an application programming interface (API) may be used for exchanging a global shape description that fulfils the requirements of 4.1. The API will specify the constraint-based geometric functions. The program control structure shall specify the function composition that constitutes the global function. The implementation of the API on a receiving system constitutes the solver of the constraint-based geometric functions. Such a program is called a parametric program.

2) It is assumed that the present technology in the field of Computer Aided Design (CAD) allows for the generation of parametric shapes in terms of a parametric program based on a standardised constraint-based API from an interactively-defined system-specific description of this family of shapes.

NOTE - This assumption clarifies the difference between the exchange format specified in this International Standard (a FORTRAN program that makes some call to a standardised API), and the environment that may be used to create such a description (e.g., an interactive graphic system such as a parametric CAD system).

4.3 Internal representation of the data created in the receiving CAD system

The interface specification shall :

— be precise enough to enable a part supplier to describe the shapes of parts

— avoid any implementation specification to enable portability on any CAD modeller.

In this International Standard, these two goals are achieved by describing a logical model of the target modelling system. This logical model is defined as an information model in the EXPRESS language. Each interface function is specified by reference to this logical model.

4.4 Library supplier and LMS user responsibility.

1) When a part is used in a product, both the representation of the part shape, and the presentation of this shape shall be created by the LMS and sent to the geometric modelling system.

EXAMPLE - If a screw is selected from the LMS by an user, during the insertion into a drawing on the CAD system, the screw shall be presented on the screen in a given colour and a given line width according to the representation selected

2) A library contains geometric descriptions that originate from different suppliers. This library may be used for various application contexts. This interface shall allow the library supplier to specify the shapes of the parts, and the library user to ensure a homogeneous level between the presentations of the parts. In this International Standard, this goal is achieved by allowing logical control of the part supplier on the shape presentation aspect (e.g., choice of a named curve style) and by assuming that:

— through some non-standardised initialisation process of the interface, the LMS user may specify the complete presentation aspect (e.g., width, font and colour values) that corresponds to each logically defined style;

— the shape generated by the LMS is displayed according to the current visualisation of the modelling system.

4.5 Compatibility

1) The representation items created inside a product data model through the interface described in this International Standard shall be exchangeable though an ISO 10303 AP conforming exchange file. All the entities attributes that may not be specified by the library supplier shall be bounded with the entity by the interface with consideration of the initialisation of the interface performed by the library user.

2) If the geometric modelling system supports an ISO 10303-22 SDAI interface [2], the interface specified in this International Standard shall be implemented as an applicative layer on the top of the SDAI interface. This applicative layer must contain :

— the solver for the constraint-based geometric entities definition,

— the default value tables for the attributes, possibly set by the LMS user, should be limited for each entity created through the SDAI.

EXAMPLE - If an arc with a given radius is to be created tangent to two lines, with a style "**plain_solid_line**", the applicative layer contains the solver that computes the tangent circle, its trimming parameters and the table that contains the precise width and colour corresponding to a '**plain_solid_line**'.

4.6 Geometry representation accuracy

Despite the fact that different modelling systems have different numeric accuracy, it shall be possible :

1) to ensure that a supplier program will work correctly on any "correct" interface implementation ;

2) to ensure that an interface implementation will correctly process a "correct" supplier program.

In this International Standard these two goals are achieved by defining reference numeric bounds for different measures involved in geometric entity definitions.

Three reference numeric bounds are defined :

1) **EPS** is the minimal value allowed for certain measures involved in some geometric entity measure.

   EXAMPLE 1 - The application program is not allowed to define a line segment whose length is smaller than EPS.

2) **MAX** is the maximal value allowed for certain measures involved in some geometric entity measure.

   EXAMPLE 2 - The application program is not allowed to define a circular arc whose radius is greater than MAX.

3) **ZERO_VALUE** is the maximal value allowed for the (mathematically computed) distance between two points that are asserted to be identical.

   EXAMPLE - The application program is not allowed to define a contour (i.e., a closed **composite_curve**) such that the distance between the end point of one **composite_curve_segment**, and the beginning point of the next **composite_curve_segment** is greater than **ZERO_VALUE**.

All these reference numeric bounds are defined in the scaled unit specified for the created geometric representation:

1) **view_length_unit** scaled by the **view_length_scale_factor** for **length_measure**.

2) **view_angle_unit** for **plane_angle_measure**.

In this International Standard, the following values are defined for the reference numeric bounds:

1) **EPS** = 10^{-3}

2) **MAX** = 10^{+4}

3) **ZERO_VALUE** = 10^{-6}

A program conforming to this International Standard shall fulfil the constraints defined, for each geometric entity, by reference to these numeric bounds. An interface conforming to this International Standard shall be able to process a program conforming to this International Standard.

5 Interface presentation

5.1 Specification and conformance

5.1.1 Allowed levels of implementation

This standard specifies 3 levels of implementation, according to the *geometrical power* of the interface whose values are : 2D, 3D curve, solid. These values are numbered 1, 2 and 3[2)]. Any interface of power level i must contain all the functions of power level j for j < i, hence, it may create views of *geometrical_power_level* j when such a power level is set through view initialisation. Whatever the geometrical power of an interface, a view may also be created with a *geometrical_power_level* equal to 0.

Three levels of interface implementations (1 to 3) have been defined. All the functions have been classified in accordance with these levels and a conforming implementation shall provide, for the selected usage, all the functions of the level it belongs to. The level of an interface may be accessed through an inquire function.

5.1.2 Simulation of missing entities

All the entities defined, for each interface level, in this International Standard shall be (conceptually) implemented in the temporary database. If some entities do not exist in the target product modelling system, they must be simulated by using other available entities. This simulation process is specified, for each entity, in this part of ISO 13584.

5.2 Interface tables

The current characteristics of the interface are stored in interface tables. All the values of these table entries may be interrogated by the application program through inquire functions contained in this part of the ISO 13584 Standard. Two tables are defined:

1) The *interface description table* contains all the persistent characteristics of the interface (e.g. *interface_level*, *hidden_line_capability*...). These values may be interrogated but may not be changed by the application program. They are implementation dependent.

2) The *interface status table* contains values of modal variables (e.g. visualisation attributes). The initial value of such a variable is defined in this International Standard either as being view dependent by set during interface initialisation process, or as having specific values. The interface status table values may be interrogated and, for all but the view dependent (e.g. *view_length_unit*, *hidden_line*, ...), changed by the application program.

2) Note : In the first version of this standard, level 3 corresponds to the creation of solids with implicit topology (solid primitives, sweeps and Boolean operations). In a later version the creation of explicit topological elements (vertex, edge, face...) may be introduced as a level 4. The former is known as CSG (Constructive Solid Geometry), the later is known as B-Rep (Boundary Representation)

The content of these interface tables are described in clause **8** of this International Standard.

5.3 Creation of product model data

This subclause introduces the concept of geometric model creation within a geometric modelling system from an application program. In the context of parts libraries conforming to the ISO 13584 Standard series it is mainly intended for parts library program developers that develop the programs that create the part model data within a CAD system.

When a functional view is selected in a Library Management System (LMS) a part supplier program is called. This functional view is related to the occurrence of the part called object occurrence, that the part supplier has described.

The role of the part supplier program is to populate this functional view by using the interface functions.

5.3.1 Reference coordinate system of a view (OVC)

A functional view created by using the interface functions consists of **geometric_representation_item**s. Each view is composed by the application program within its own **geometric_representation_context**, called the *Object_view_modelling_coordinate_system* (OVC). The application program is independent of the relative positioning of the OVC in any CAD coordinate system. The Library Management System (LMS) is in charge of view initialisation, and it is assumed that after the view initialisation function has been performed, all the **geometric_representation_item**s sent to the CAD system in their own OVC will be accurately positioned and/or converted.

The *"View_initialisation "* function should therefore activate some unspecified positioning process. In practice, and according to the particular CAD system philosophy, the positioning process may, e.g.:

1) define a new local coordinate system if the CAD system uses an instancing mechanism;

2) attach the OVC to the cursor for later positioning ;

3) accomplish some interactions with the CAD user to interrogate the position and then initialise a transformation matrix in the interface;

4) compute the correct position if it results from the object occurrence positioning and then initialise a transformation matrix in the interface;

5) do nothing, the view is created at the origin of the global coordinate system and is positioned afterwards by the CAD user.

When a view is initialised as 2D, the 2D space is assumed to be the x, y plane, hence the z coordinate used in the creation function is meaningless for geometric entities. For such entities the z coordinate shall be equal to zero.

In the context of part library data conforming to the ISO 13584 Standard series, when several part supplier programs refer to different functional views of the same part, the OVC used in these different programs are dependent upon each other. An absolute coordinate system shall be associated with the part by the part supplier. All the part supplier programs that produce 3D functional views of this part shall use this absolute coordinate system as their own OVC. All the part supplier programs that produce 2D functional views of this part shall:

1) Specify the 2D functional view that is produced by each part supplier program in conformity with ISO 128, (see **Figure 1**) .

2) Select as OVC for each part supplier program the coordinate system that results from the part absolute coordinate system and from the specification of the produced 2D functional view (see **Figure 1**).

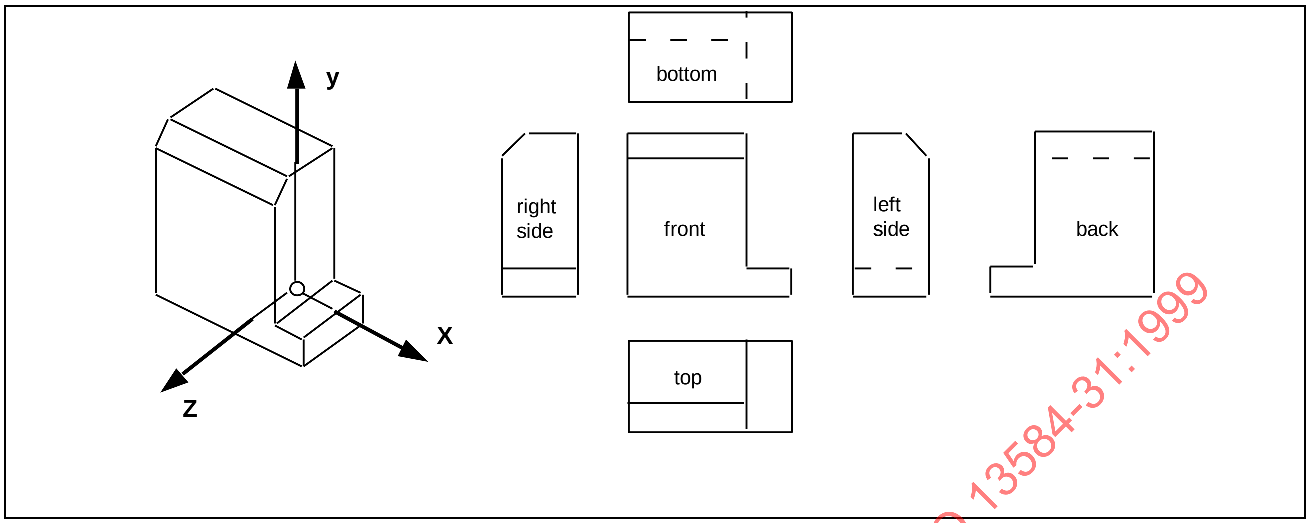


Figure 1 — Absolute coordinate system of a part (parts supplier defined)

5.3.2 Geometrical units in the OVC

The length and plane angle units used in OVC are defined by three interface status table entries: *view_length_unit*, *view_length_scale_factor*, and *view_angle_unit*. The *view_length_unit* entry defines the base length unit used in the view. It may be metre (METRE) or inch (INCH). The *view_length_scale_factor* entry defines the multiplicative scale factor to be applied to the base length unit. The *view_angle_unit* defines the plane angle unit used in the view. It may be radian (RAD), degree (DEG), or grad (GRAD). In this International Standard, the word "OVC unit" refers to either the *view_length_unit* scaled by the *view_length_scale_factor* (*OVC_length_unit*) or to the *view_angle_unit* (*OVC_angle_unit*).

The default values of the OVC unit are specified in clause **8.2** of this International Standard. These default values may be redefined by the part supplier, externally to the program, as part of the functional model the part supplier program belongs to. The default values are set during view initialisation. They may be interrogated, but not changed by the application program. All the geometrical dimensions defined by or returned to the application program are specified in the current OVC units.

The interface ensures correct scaling from OVC units to the CAD system modelling space units. This scaling, called the *OVC-CAD transformation* applies to all the geometric representation items created through interface functions.

5.3.3 Content of a view

The role of the interface functions is to create data inside product modelling system databases. Since product modelling system databases are different from each other, the exact effect of each function may not be described at the physical level. To allow a precise specification of the effect of an interface function, this standard defines the target CAD system database through a logical model defined as an EXPRESS information model (see clause **6**). This logical model is assumed to be implemented in some physical way in the target CAD system.

5.3.4 Temporary database

To provide for the creation of intermediate geometry, a *temporary database* (TDB) has to be created. The interface functions allow the creation of geometrical entities either in the Temporary database or as CAD system data. The Temporary data entities may be referenced, modified, used in a geometric construction, or sent to the CAD system. Entities inside the CAD system shall not be referenced. When a temporary database entity is sent to the CAD system, it is no longer referenced as temporary data.

For geometrical entity types having the same visualisation types, attributes may be created both as temporary database entities and in the CAD system.

The visualisation attributes are attached to these entities when they are created on a modal basis, whether this creation occurs in the TDB or in the CAD system. In the TDB, entities attributes may be changed. When an entity is sent from the TDB to the CAD system, visualisation attributes of this entity retain their current values in the TDB.

Temporary data entities may be geometrically moved or duplicated. These geometric manipulations do not change the visualisation attributes: the modified entity, or the duplicated one, preserves the visualisation attributes of the initial one.

The structural relationships available are different in the TDB and in the CAD system. The entities sent to the CAD system are structured by sets. Sets are objects permanently existing in the CAD system database. For this structure a hierarchical set structure is used. For the structure of elements in the TDB, a temporary group structure is used. The group structure may be used to facilitate the creation process of geometric elements. This group structure is also hierarchical. The maximum number of entities that shall be allowed in the TDB by the interface implementation is equal to or greater than that specified in clause **10** of this International Standard.

This document does not specify any implementation form of this temporary database. Only the functionality required from the part supplier functions shall be supported.

5.3.5 Hidden line removal process

For 2D views created by the interface, a hidden line concept is provided.

1) In addition to the curve entities, 2D interfaces may create *fill area* that may be "opaque".

2) An interface status table entry, called *hidden_line_involved* (HLI) specifies whether or not the curve or fill area entities created through the interface shall be involved in the hidden line removal process. When HLI is equal to *true*, each curve and fill area entity created through an interface function shall be involved in the hidden line removal process and shall have attached an **api_predefined_occlusion_style** visual appearance style. This **api_predefined_occlusion_style** style shall contain a *view_level* attribute which represents the "height" of the entity in some virtual 3D space as a real value, and a *name* attribute that specifies how each entity must be changed if it is hidden. An opaque fill area hides all or part of curve entities that are inside its borders and have a strictly lower value of view level. It shall not hide curve entities with the same value of view level.

3) Only the curve and fill area entities created when HLI is equal to *false* shall be sent to the CAD system when the application program requests this transmission with the *Fix_Ent* (fix entities into CAD system) function, or requests their direct creation in the CAD system during view construction. The curve and fill area entities created when HLI is equal to *true* remain in the interface until the hidden line removal process is performed.

4) When the application program requests the transmission to the CAD system of an entity that is involved in the hidden line removal process, either with the *Fix_Ent* function, or by requiring its direct creation in the CAD system, this entity is attached to an **api_predefined_virtually_sent_style** and is said to be virtually sent. The **api_predefined_virtually_sent_style** style shall contains an *api_set_name* attribute that shall contain, in the format of a string, the (unique) name of the current open set when the entity is virtually sent.

5) At the end of each view construction, the removal process is performed. Only the virtually sent entities are involved in this process. Temporary database entities are not involved.

6) When a virtually sent curve entity is partially hidden by a fill area, the visible part shall be presented with the current style of the curve. The parts of the virtually sent curve entity that are hidden are processed according to the *name* attribute of the **api_predefined_occlusion_style** visual appearance style.

If the value of this attribute is :	they are :
no_change	sent without any change ;
dashed	sent as invisible if this capability exists in the CAD system, or else curve entities and fill area borders and hatching are sent as dashed lines ;
invisible	sent as invisible if this capability exists in the CAD system, or else they are not sent.

When a fill area is hidden by another surface, a hidden part removal process is only required by this International Standard on the lines that belong to the fill areas : borders and/or hatching.

The involvement of points is not specified in this International Standard.

The capability of hidden line removal is not mandatory. The *hidden_line_capability* entry of the interface description table states if or if not it is available. The *hidden_line* entry of the interface status table states if or if not the elimination process shall be activated for the next view. The default value of the *hidden_line* entry is set to the value of *hidden_line_capability* (if the capability is available, it is always activated unless it is changed by the application program).

The hidden line removal process may only be activated for views initialised as 2D.

The hidden line removal process shall not change the set structure defined by the application program at the (virtual) sending time. This set structure is recorded in the *api_set_name* attribute of the **api_predefined_virtualy_sent_style** style.

5.3.6 The representation process

The role of the interface functions is to create data inside product modelling system databases. The viewing process of such data is assumed to be controlled by the product modelling system and the system user. However, it can be seen that the application program must have some control over the geometrical aspect of entities (for instance to meet the requirements of the applicable technical drafting standard, or to underscore some semantic difference between entities)because the user wishes some similarity between views obtained from various supplier libraries.

These two goals are achieved in the following way (see clause **6.2.4** and clause **6.2.5**):

1) All the presentation styles are defined either as pre-defined styles or as externally defined styles

2) The pre-defined styles are described by this part of ISO 13584. Externally defined styles may be defined either by this part of ISO 13584 or by any part belonging to the exchange series of parts.

3) Pre-defined styles or externally defined styles only partially describe the visual appearance of the corresponding style. According to their requirements, they may leave e.g., the colour as implementation dependent

4) The interface shall provide a tool to allow the CAD user to set the exact value of all the visual appearance attributes left as implementation dependent each pre-defined or externally defined style.

5) When a view exchange protocol referenced in an application program is not supported by some interface implementation, then the first style defined for the current representation item in this part of ISO 13584 shall be used in place of the unknown style and no error shall be reported.

5.4 Entities structure

5.4.1 Group structure in the TDB

In the TDB, entities are gathered into a group defined using the entity, *Entity_structured*. When a function operates on a group, it operates repetitively (and recursively) on each relevant entity of the group. When this function is a duplication function, its result is also a group. This group belongs to the current open group and has the same group structure as the initial one. In this duplication process, the duplicate of an open group shall be a closed group. When the function is a modification function, it shall preserve the existing group structure of the existing entities that are inside the modified group.

Entities shall not be modified and no error shall be reported when a function is triggered on a group that contains geometrical entities not allowed as function input parameters. For instance, if the *Chg_Curve_Style* (Change Presentation Style for Curves) function is triggered on a group that contains points, solids and curve entities, the *curve_style* of the curve entities shall be modified but points and solid entities shall remain unmodified in the same group structure.

The whole TDB is itself a group. It is called the *root group*. This root group is open when the interface is initialised and shall not be closed. Hence, there shall always exist an open group.

With the exception of the root group each entity, geometrical or structured shall belong to exactly 1 group (that may be the root group). Groups are structured according to a hierarchical tree structure. The root of the tree is the root group.

Groups may be :

— created: they belong to the current open group and becomes the current open group;

— reopened: all the entities created in the TDB after the group reopening will belong to this group until its closure;

— closed.

Entities sent to (or created in the) CAD system are removed from the group structure.

To ensure a hierarchical group structure, the open groups are managed through a stack. The top of the stack is the current open group.

When the interface is initialised, the root group is put into the stack. No function shall be allowed to close this group, hence it shall always remain in the stack.

When a group is created (1) it belongs to the current open group and, (2) it is put on the top of the stack. Hence, it becomes the current open group.

Only the group that is at the top of the stack may be closed. In this case, this group is removed from the stack and the new top of the stack become the current open group.

When a group is reopened, it is put on the top of the stack. This does not change the group the reopened group belongs to.

Three functions provide for direct modifications of the group structure. None of these functions change the stack content.

The **remove_ent_grp** (Remove Entity from Group) function allows the removal of an entity (geometrical or structured) from a group. After this function, the entity shall belong to the root group.

The *Gather_Ent_Grp* (Gathering entities into new group) function allows a list of entities (geometrical or structured) to be gathered into a new group. None of these entities shall contain the current open group. All these entities are removed from the group to which they belonged and are put into this new group. This new group shall belong to the current open group.

The *Add_Ent_Grp* (Adding Entity into Group) function allows the addition of an entity (geometrical or structured) to an existing group. This entity shall not contain the group to which it is being added. This entity is first removed from its group and then added to the given group.

Groups are local to the TDB. Their intent is to facilitate geometrical construction. The maximum number of groups that shall be allowed by the interface implementation is equal to or greater than that specified in clause **10** of this International Standard.

5.4.2 Structure of the entities sent to the CAD system

It is assumed that the data stored in the CAD system database are distributed into subsets. Conceptually all the data belong to views. Inside a view, the geometrical data are structured into sets and subsets according to a hierarchical tree structure. The structure to be given to the data sent by the application program is specified to the CAD system in the following manner.

1) Before sending any data, a view initialisation must be performed by the LMS. All the data sent to the CAD system between this initialisation and the end of the application program shall belong to the view. A view shall not contain another view.

2) The *Open_Set* function opens a set. The set name shall be placed on the top of a set stack, and all the geometrical entities sent to the CAD system shall belong to this set. The set itself is a subset of the previous top of the set stack, or, if the set stack is empty, a subset of the view.

The *Close_Set* function shall only be allowed for a set name that is on the top of the set stack. When the *Close_Set* function is called, this set is closed and its name shall be removed from the stack. If the stack is not empty the top of the stack is the current open set. If the stack is empty there is no open set. A closed set shall never be reopened. The name of each set shall be unique inside a view. The maximum size of the set stack that shall be allowed in the TDB by the interface implementation is equal to or greater than that specified in clause **10** of this International Standard.

The mapping between this conceptual structure and a depth-limited tree structure that may be available on a target CAD system is made as follows. The top level of the CAD tree structure, if any, maps the view structure. The following levels, if they exist, map the first levels of the set-subset tree structure. When a set of the CAD tree structure is terminal (i.e. when it may no longer be subdivided into subsets) all the entities belonging to subsets of the conceptual corresponding set are put into this terminal set.

5.5 Geometrical or structured entity name

To be able to refer to any entity created in the TDB, all the entities created by the interface function are named by a value belonging to some abstract data type called *entity_name_type*. The value of this abstract data type can be either 0 or unknown. When an interface function fails and does not succeed in creating some entity, this function returns 0. When an entity is sent to the CAD system, access to the entity is no longer available and the name of this entity becomes unknown. The unknown value is, in particular, returned by the interface functions when an entity is created directly in the CAD system database. When an interface function is called with zero or unknown entities as arguments, it shall return 0.

All the non zero and non unknown values of entity names returned by interface functions during one session, (i.e. between the time where the LMS triggers an application program and the time where this program returns) shall be unique. The name of an entity shall not be reused, even when the first entity has been sent to the CAD system.

NOTE - In the FORTRAN binding, in FORTRAN, *entity_name_type* is mapped onto INTEGER. The zero value is mapped onto 0. The unknown value is mapped onto a negative value. Hence, only positive integer are accessible entity names.

5.6 Coordinate system and transformation

The interface provides functions to change the reference coordinate system of the OVC modelling space. Four functions may be used by the application program: *Ref_Sys_3_Pnt*, *Ref_Sys_2_Dir*, *Ref_Sys_Position_Relative* and *Ref_Sys_A2p*. All the entities created after such a change, either in the TDB or in the CAD system database, shall be defined with respect to the new reference coordinate system.

To allow the application programmer to preserve the previous reference coordinate system, the *Ref_Sys_A2p* (Reference System by Axis2_placement) function allows the creation of an LCS entity from the current OVC reference coordinate system. Subsequent change of the reference coordinate system to this LCS allows the resetting of the coordinate system of the OVC to its previous value. The coordinate system of the OVC can be reset to its previous value if the reference coordinate system is changed to this LCS.

5.7 Interface error state

A global *error_variable* shall be set *w*hen an error condition is detected during execution of an interface function. It is an integer entry from the interface status table, corresponding to the integer error number defined in the function specification. It shall also write in the *error_origin* entry of the interface status table the name of the function that identified the error, and in the *error_text* entry the message associated with the error number. The name of the function shall be the syntactical name in the currently used language (e.g. FORTRAN). The message shall be a translation of the error description given in **5.8.1**. These error variables may be interrogated and reset by the application program.

As long as the *error_variable* is set, the interface shall remain in an error state (*error_state* = true). In this error state, the only interface functions that are allowed and behave as specified in **Annex A** of this International Standard are:

1) The inquire functions, and

2) the Reset_Error_State function,

All the other interface functions are permitted but they do not change anything. They return to the calling application program. When the application program returns while the interface is in an error state, the LMS shall:

1) Close all the possible open **set**s,

2) Close the open view with *error_state* = true,

3) Write to the error file, the *error_variable*, *error_origin* and *error_text* entry values,

4) Close the interface.

5.8 Error handling

5.8.1 Error handling methodology

For each interface function, a finite number of error situations is specified that will cause the error variables to be set. Every interface implementation shall support this error checking. The error variables provide an interface between the application program and the standardised interface. The application program may interrogate the error value, interpret the information about the error and reset the *error_variable* to zero, to restore the interface to a non error state (*error_state* = false).The interface error handling strategy is derived from the following classification of errors :

Class	I	errors resulting in a precisely defined reaction ;
Class	II	errors resulting in an attempt to save the results or previous operations ;
Class	III	errors that cause unpredictable results including the crash of the CAD system.

The interface recognises three situations in which errors are detected :

Situation A error detected in interface functions ;

Situation B error detected in functions called from interface (CAD system functions, operating system functions) ;

Situation C error detected outside the interface.

If errors are detected outside the interface (situation C), either the application program may regain control over the execution, or program execution will be terminated abnormally. In the latter case, results are unpredictable (class **III**) and may cause the CAD system may crash. If, however, the application program obtains control, it may attempt to return to the LMS to try and close the interface properly (see **5.7**). The operations defined in **5.7** may also be performed by the interface itself as a standard error reaction to class **II** errors.

All errors that are listed explicitly as part of the definition of the interface functions belong to class **I**. Either they are detected within the interface itself (situation A) or when a function called from the interface has returned control to the corresponding interface function with the appropriate error information (situation B). In all class **I** cases, the interface sets the *error_variable*, *error_origin*, and *error_text*, to the error values. If the failure occurs during an entity creation function and the entity cannot be created, then the entity name returned by the function is set to zero. If an interface function is invoked with more than one error condition, any one of the relevant error number is set in the error variables.

The *Inq_Error_State* function allows the application program to handle the error. The *Reset_Error_State* function allows the removal of the interface from the error state. To close the open view with *error_state* = true allows the LMS to forewarn the CAD system that a view is wrong .

While in the error-state, the inquire functions behave as specified in their functional description (see annex **A**), and an action shall not generate a new error. Hence, for inquire functions no errors are specified. An output parameter, called *error_indicator*, is used to report possible difficulty during function performance.

The following error numbers are reserved:

1) Unused error numbers less than 1001 are reserved for future standardisation,

2) Error numbers 1000 to 2000 are reserved for language binding.

5.8.2 Error messages

Table 1 — Input error messages

error number	**error description**
1	entity name not defined (zero, or unknown)
2	entity type out of permitted range
3	value for length measure out of permitted range
4	value for plane angle measure out of permitted range
5	integer value out of permitted range
6	string value out of permitted range
7	real value out of permitted range

Table 2 — Geometry error messages

error number	error description
101	attempt to create a degenerated entity
102	magnitude of direction vector out of range [EPS,MAX]
103	distance between two points out of range [EPS,MAX]
104	distance between two contours less than EPS
105	attempt to create a degenerated direction during entity creation
106	attempt to create a degenerated axis2_placement during entity creation
107	attempt to create a degenerated axis1_placement during entity creation
108	attempt to create a degenerated basic curve during entity creation
109	attempt to create a degenerated solid during entity creation
110	attempt to create a point outside the parametric range of curves entity
111	attempt to create a line whose segment length is out of range [EPS,MAX]
112	attempt to create an arc whose segment length is less than EPS
113	attempt to create a self-intersected contour entity
114	attempt to create an overlapping solid
115	given entities are identical
116	given points are linear dependent
117	given directions are parallel
118	given curves entities are parallel/concentric
119	given entities are not in the same plane
120	given cut length too long
121	radius too big/small
122	no intersection of given curves entities
123	an intersection of given contours detected
124	an intersection between axis an plane of surface detected
125	an overlapping of given contours detected
126	axis of revolution not in plane of surface
127	geometrical design is not feasible
128	calculation process for creating a conical arc numerical not stable
129	Approximation process to ensure contour closure failed
130	Boolean operation failed

Table 3 — System error messages

error number	error description
201	temporary database overflow
202	error while sending entity to CAD system
203	function not compatible with implemented interface level
204	function not compatible with current power level
205	maximal number of points per polyline exceeded
206	maximal number of entities per contour exceeded
207	maximal number of inner boundaries exceeded
208	maximal number of groups exceeded
209	maximal number of character per string exceeded
210	group stack overflow
211	set stack overflow
212	usage of entity only allowed for usage within the TDB

Table 4 — Entity structure error messages

error number	error description
301	attempt to close the root group
302	attempt to reopen an already open group
303	entity is member of root group
304	entity contains the current open group
305	attempt to create cyclical group structure
306	name of set not unique
307	attempt to close the root set

Table 5 — Presentation style error messages

error number	error description
401	source of exchange protocol unknown
402	identifier of external style unknown
403	assignment of hatch style failed
404	hidden line occlusion style not attached

Table 6 — Language binding error messages

error number	error description
1001	enumerated value out of range
1002	mismatch of number and list length
1003	mismatch of string length

6 Logical model of the target modelling system

6.1 Geometric representation item

The role of the interface functions is to create representation items either in the TDB or in the CAD system database.

There exists three kind of representation items.

— geometric representation items are geometric or annotation entities used to describe the shape that is created through the interface;

— styles are entities used to describe the visual appearance of the geometric representation items;

— structured entities are used to structure the geometric representation items either in the TDB or in the CAD system database.

The geometric representation items that may be created through the interface functions are classified according the following tree (see **Figure 2**).

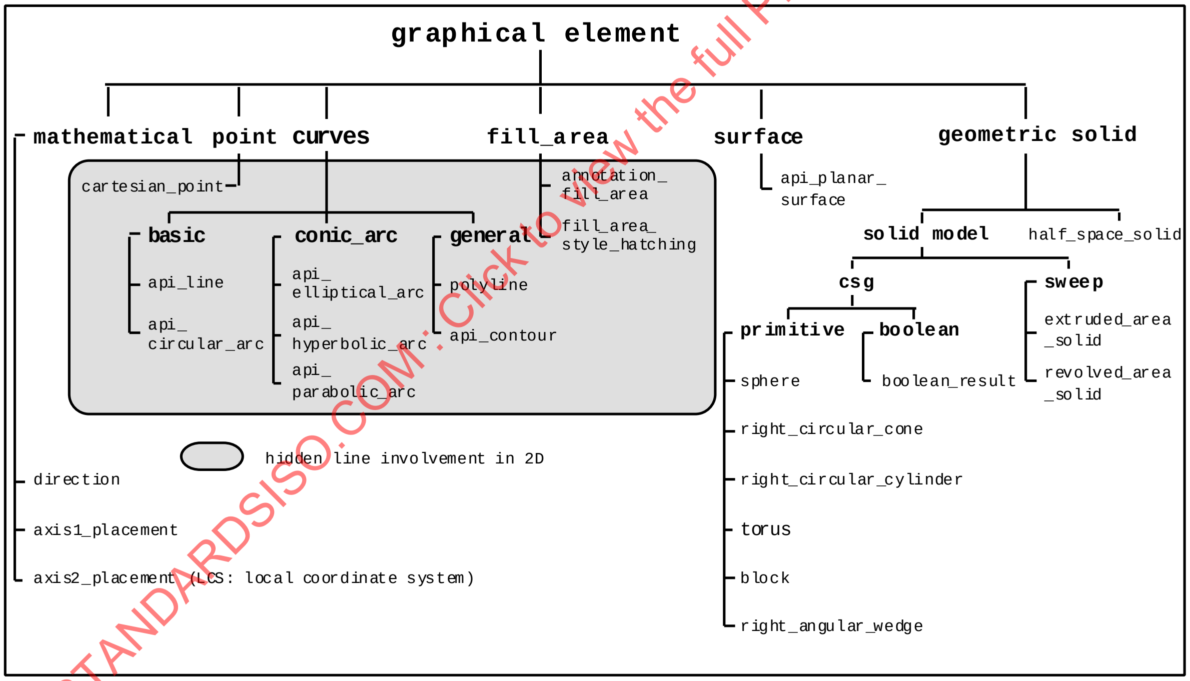


Figure 2 — Geometric representation items defined in the interface

This structure is used both to describe the styles of the various entities and the range of some interface functions.

The implementation of the geometric representation items inside the TDB or in the CAD system is not standardised. Nevertheless, a model of this implementation is defined in this Standard to specify the geometric behaviour of each entity during entity manipulation. This model is defined through an abstract data model specified in EXPRESS. This abstract data model, called **api_abstract_schema**, uses a subset of the generic resources defined in the integrated resource series (Part 41, 42, 43 and 46) of part of ISO 10303 to specify product model data. These resources are referred to as "ISO 10303 generic resources". This abstract data model is not necessarily implemented in the TDB, nor in the CAD system. All of the entities created through the interface shall behave as if their implementations conform to this abstract data model.

If an interface implementation is intended to create entities conforming to some ISO 10303 application protocol that specialises the ISO 10303 generic resources, the same specialisation shall be applied to the subset of these resources used in the **api_abstract_schema**. Any additional information shall be generated by the interface.

In the definition of the **api_abstract_schema**, the types and entities defined in the ISO 10303 generic resources preserve the names they have in ISO 10303, even if some additional constraints or restriction of allowed subtypes, are added in their definition. These constraints shall be checked by the interface and, if they are satisfied, the created entities shall conform to the definition given in ISO 10303. The additional WHERE RULES that express the constraints specific to **api_abstract_schema** are identified by a name prefixed by the string "**api_**".

Some entities are also defined by explicitly subtyping entities defined in the ISO 10303 integrated resources. This subtyping is used to specify the range of some interface functions. The name of such entities is prefixed by the string "**api_**". The subtyping generally consists of restricting the entities defined in the ISO 10303 integrated resources. Such entities may be implemented as an instance of their supertype, or as instances of the specialisation of these supertypes defined in some ISO 10303 Application Protocol.

Finally, some entities are defined by generalisation of entities defined in the generic resources of ISO 10303 by adding new attributes. Such entities are mainly used for defining structure and visual appearance. When the target CAD system is a repository conforming to some ISO 10303 Application Protocol, the interface shall ensure the mapping of these entities onto the resources available within this application protocol. The mapping is textually described in the definition of the api-specific entity.

When some additional constraints are added to the EXPRESS specification of an ISO 10303 generic resource, a Note documents the nature of this restriction. When no restrictions are documented, the resource definition conforms to the definition of the ISO 10303 generic resource.

In the integrated resource series of parts of ISO 10303, some generic resources explicitly refer to other generic resources that are neither used nor referenced in **api_abstract_schema** and whose instance may not appear in a population conforming to the **api_abstract_schema**. These resources are REFERENCEd from the relevant EXPRESS schema in the ISO 10303 integrated resources in order to preserve the structure of the generic resources (particularly the existing WHERE RULES) while insuring the formal correctness of the schema. These entities are only referenced in the WHERE RULES duplicated from the ISO 10303 generic resources, not from entities belonging to the **api_abstract_schema**. Therefore, this REFERENCE is only formal.

The entities created by the interface shall not be degenerated. The concept of degeneration shall be independent of any particular interface implementation. In this International Standard, constraints are defined for each entity that may be created by the interface. An entity that does not fulfil these constraints is called a degenerated entity. When a function tries to create a degenerated entity, an error shall occur, the entity shall not be created, and an error message shall be generated.

The degeneration of entities is defined by reference to an absolute minimal allowed value called EPS that is expressed in the (current) *view_length_unit* scaled by the *view_scale_factor*. The value defined for:

EPS is 10^{-3} *view_length_unit* x *view_scale_factor*.

When a particular interface implementation does not have the capability to create entities whose extent is as small as EPS for a particular choice of the *view_length_unit* by the application program, an error shall occur during the *"set_ovc_length_unit"* function, that is a function triggered by the LMS.

The constant, ZERO_VALUE, defines the real values that shall be identified with zero by the interface on any interface implementation. When the distance between two points is less than ZERO_VALUE, these two points shall be considered as identical by the interface.

For instance, when the distance between two trimming points of two **composite_curve_segment**s that belong to the same **composite_curve** is less than ZERO_VALUE, the interface shall ensure the continuity of the **composite_curve**, regardless of the required precision of the target CAD system. The ZERO_VALUE is expressed in the (current) *view_length_unit* scaled by the *view_scale_factor*. The value defined for:

ZERO_VALUE is 10^{-6} *view_length_unit* x *view_length_scale_factor*.

NOTE - The definition of these two values acknowledges the impracticability of exact real calculus that lead to ambiguities. The solution retained, that is often used in practice, consists of defining a range [ZERO_VALUE, EPS] of forbidden real values.

6.1.1 api_abstract_schema

This subclause defines requirements for the **api_abstract_schema**. The following EXPRESS declaration introduces the **api_abstract_schema** block and reference the external resources required for formal consistency with the ISO 10303 generic resource definition. Instances of such referenced entities shall not appear in a population created using interface functions.

EXPRESS specification:

```
*)
SCHEMA api_abstract_schema;
 REFERENCE FROM geometry_schema
  (pcurve);
 REFERENCE FROM measure_schema
  (measure_with_unit,
descriptive_measure);
 REFERENCE FROM presentation_appearance_schema
  (surface_style_usage,
   presentation_style_by_context,
   fill_area_style_colour,
   fill_area_style_tiles,
   pre_defined_hatch_style,
   pre_defined_presentation_style,
   pre_defined_tile_style,
   externally_defined_hatch_style,
   externally_defined_curve_font,
   externally_defined_tile_style,
   curve_style_font,
   curve_style_font_and_scaling,
   text_style,
   point_style,
   symbol_style,
   approximation_tolerance);
(*
```

NOTE - The schema referenced above can be found in the following part of ISO 10303:

geometry_schema	ISO 10303-42
measure_schema	ISO 10303-41
presentation_appearance_schema	ISO 10303-46

6.1.1.1 API_ABSTRACT_SCHEMA constant definition: Geometry representation accuracy

This subclause declares the named constants used in **api_abstract_schema** as reference numeric bounds for geometry representation accuracy.

EXPRESS specification:

```
*)
CONSTANT
 EPS    : REAL := 1.E-3;
 ZERO_VALUE : REAL := 1.E-6;
 MAX    : REAL := 1.E+4;
END_CONSTANT;
(*
```

NOTE - In the context of the **api_abstract_schema**, **EPS**, **ZERO_VALUE** and **MAX** shall be expressed in *view_length_unit* scaled by *view_length_scale_factor*. for **length_measure** and in *view_angle_unit* for **plane_angle_measure**.

6.1.2 API_ABSTRACT_SCHEMA type definition : Fundamentals of product description and support

This subclause declares the generic type resources defined in ISO 10303-41 that are part of ISO 13584 **api_abstract_schema**.

6.1.2.1 Identifier

An **identifier** is an alphanumeric string that allows an individual thing to be identified. It may not provide natural language meaning.

EXAMPLE - A part number would be an identifier.

EXPRESS specification:

```
*)
TYPE identifier = STRING;
END_TYPE;
(*
```

6.1.2.2 Label

A **label** is the term by which something may be referred to. It is a string that represents the human-interpretable name of something and shall have a natural language meaning.

EXAMPLE - "Smith", "Widget Inc.", and "Materials Test Laboratory" are examples of labels.

EXPRESS specification:

```
*)
TYPE label = STRING;
END_TYPE;
(*
```

6.1.2.3 Text

A **text** is an alphanumeric string of characters that is intended to be read and understood by a human being. It is for information purposes only.

EXPRESS specification:

```
*)
TYPE text = STRING;
END_TYPE;
(*
```

6.1.2.4 Length_measure

A **length_measure** is the value of a distance.

EXPRESS specification:

```
*)
TYPE length_measure = REAL;
END_TYPE;
(*
```

NOTE - In the context of the **api_abstract_schema**, **length_measure** shall be expressed in *view_length_unit* scaled by *view_length_scale_factor*.

6.1.2.5 Plane_angle_measure

A **plane_angle_measure** is the value of an angle in a plane.

EXPRESS specification:

```
*)
TYPE plane_angle_measure = REAL;
END_TYPE;
(*
```

NOTE - In the context of the **api_abstract_schema**, **plane_angle_measure** shall be expressed in *view_angle_unit*.

6.1.2.6 Positive_length_measure

A **positive_length_measure** is a **length_measure** that is greater than zero.

EXPRESS specification:

```
*)
TYPE positive_length_measure = length_measure;
WHERE
 WR1: SELF > 0;
END_TYPE;
(*
```

Formal propositions:

WR1: The value shall be positive.

6.1.2.7 Positive_plane_angle_measure

A **positive_plane_angle_measure** is a **plane_angle_measure** that is greater than zero.

EXPRESS specification:

```
*)
TYPE positive_plane_angle_measure = plane_angle_measure;
WHERE
 WR1: SELF > 0;
END_TYPE;
(*
```

Formal propositions:

WR1: The value shall be positive.

6.1.2.8 Parameter_value

A **parameter_value** is the value that specifies the amount of a parameter in some parameter space.

EXPRESS specification:

```
*)
TYPE parameter_value = REAL;
END_TYPE;
(*
```

6.1.2.9 Message

A **message** is a communication that is addressed to a system in order to trigger some action. The result of such an action is an **externally_defined_item**.

NOTE - The legal values for the message are specified within an application interpreted model.

EXPRESS specification:

```
*)
TYPE message = STRING;
END_TYPE;
(*
```

6.1.2.10 Reference

A **reference** is a means of identifying and retrieving an **externally_defined_item**.

EXPRESS specification:

```
*)
TYPE source_item = SELECT (identifier, message);
END_TYPE;
(*
```

6.1.3 API_ABSTRACT_SCHEMA type definition : Geometric and topological representations

This subclause declares the generic type resources defined in ISO 10303-42 that are part of the **api_abstract_schema**.

6.1.3.1 Dimension_count

A **dimension_count** is a positive integer used to define the coordinate space dimensionality of a **geometric_representation_context**.

EXPRESS specification:

```
*)
TYPE dimension_count = INTEGER;
WHERE
 WR1: SELF > 0;
END_TYPE;
(*
```

Formal propositions:

WR1: A **dimension_count** shall be positive.

6.1.3.2 Transition_code

This type conveys the continuity properties of a composite curve or surface. The continuity referred to is geometric, not parametric continuity.

EXPRESS specification:

```
*)
TYPE transition_code = ENUMERATION OF
 (discontinuous,
  continuous,
  cont_same_gradient,
  cont_same_gradient_same_curvature);
END_TYPE;
(*
```

Enumerated item definitions:

discontinuous: The segments, or patches, do not join. This is permitted only at the boundary of the curve or surface indicating that it is not closed.

continuous: The segments, or patches, join but no condition on their tangents is implied.

cont_same_gradient: The segments, or patches, join, and their tangent vectors, or tangent planes, are parallel and have the same direction at the joint; equality of derivatives is not required.

cont_same_gradient_same_curvature: For a curve, the segments join, their tangent vectors are parallel and in the same direction and their curvatures are equal at the joint; equality of derivatives is not required. For a surface, this implies that the principal curvatures are the same and the principal directions are coincident along the common boundary.

NOTE - In the context of the **api_abstract_schema** there are only composite curves.

6.1.3.3 Preferred_surface_curve_representation

This type is used to indicate the preferred form of representation for a surface curve, that is either a curve in geometric space or in the parametric space of the underlying surfaces.

EXPRESS specification:

```
*)
TYPE preferred_surface_curve_representation = ENUMERATION OF
 (curve_3d,
  pcurve_s1,
  pcurve_s2);
END_TYPE;
(*
```

Enumerated item definitions:

curve_3d: The curve in three-dimensional space is preferred.

pcurve_s1: The first pcurve is preferred.

pcurve_s2: The second pcurve is preferred.

6.1.3.4 Trimming_preference

This type is used to indicate the preferred way of trimming a parametric curve where the trimming is multiply defined.

EXPRESS specification:

```
*)
TYPE trimming_preference = ENUMERATION OF
 (cartesian, parameter,
  unspecified);
END_TYPE;
(*
```

NOTE - In the context of **api_abstract_schema**, the **trimming_preference** is implementation dependent.

Enumerated item definitions:

cartesian: Indicates that trimming by cartesian point is preferred.

parameter: Indicates a preference for the parameter value.

unspecified: Indicates that no preference is communicated.

6.1.3.5 Axis2_placement

This select type collects together both versions of axis2 placement as used in two-dimensional or in three-dimensional Cartesian space. This enables entities requiring this information to reference them without specifying the space dimensionality.

EXPRESS specification:

```
*)
TYPE axis2_placement = SELECT
 (axis2_placement_2d,
  axis2_placement_3d);
END_TYPE;
(*
```

6.1.3.6 Curve_on_surface

A **curve_on_surface** is a curve on a parametric surface. It may be any of the following

— a **pcurve** or

— a **surface_curve**, including the specialised subtypes of **intersection_curve** and **seam_curve**, or

— a **composite_curve_on_surface**.

The **curve_on_surface** select type collects these curves together for reference purposes.

EXPRESS specification:

```
*)
TYPE curve_on_surface = SELECT
 (pcurve,
  surface_curve,
  composite_curve_on_surface);
END_TYPE;
(*
```

6.1.3.7 Pcurve_or_surface

This select type enables a surface curve to identify as an attribute the associated surface or pcurve.

EXPRESS specification:

```
*)
TYPE pcurve_or_surface = SELECT
 (pcurve,
  surface);
END_TYPE;
(*
```

6.1.3.8 Trimming_select

This select type identifies the two possible ways of trimming a parametric curve, by a cartesian point on the curve, or by a REAL number defining a parameter value within the parametric range of the curve.

EXPRESS specification:

```
*)
TYPE trimming_select = SELECT
 (cartesian_point,
  parameter_value);
END_TYPE;
(*
```

6.1.3.9 Vector_or_direction

This type is used to identify the type of entity that can participate in vector computations.

EXPRESS specification:

```
*)
TYPE vector_or_direction = SELECT
 (vector,
  direction);
END_TYPE;
(*
```

6.1.4 API_ABSTRACT_SCHEMA type definition: Geometry models

This subclause declares the generic type resources for geometry models defined in ISO 10303-42 that are part of the **api_abstract_schema**.

6.1.4.1 Boolean_operand

This select type identifies all those types of entities that may participate in a Boolean operation to form a CSG solid.

EXPRESS specification:

```
*)
TYPE boolean_operand = SELECT
 (solid_model,
  half_space_solid,
  csg_primitive,
  boolean_result);
```

```
END_TYPE;
(*
```

6.1.4.2 Boolean_operator

This type defines the three Boolean operators used in the definition of CSG solids.

EXPRESS specification:

```
*)
TYPE boolean_operator = ENUMERATION OF
 (union,
  intersection,
  difference);
END_TYPE;
(*
```

Enumerated item definitions:

union: The operation of constructing the regularised set theoretic union of the volumes defined by two solids.

intersection: The operation of constructing the regularised set theoretic intersection of the volumes defined by two solids.

difference: The regularised set theoretic difference between the volumes defined by two solids.

6.1.4.3 Csg_primitive

This select type defines the set of CSG primitives that may participate in Boolean operations. The CSG primitives are **sphere**, **right_circular_cone**, **right_circular_cylinder**, **torus**, **block** and **right_angular_wedge.**

EXPRESS specification:

```
*)
TYPE csg_primitive = SELECT
 (sphere,
  block,
  right_angular_wedge,
  torus,
  right_circular_cone,
  right_circular_cylinder);
END_TYPE;
(*
```

6.1.4.4 Csg_select

This type identifies the types of entity that may be selected as the root of a CSG tree including a single CSG primitive as a special case.

EXPRESS specification:

```
*)
TYPE csg_select = SELECT
 (boolean_result,
  csg_primitive);
END_TYPE;
(*
```

6.1.4.5 Geometric_set_select

This set select identifies the types of entities that can occur in a **geometric_set**.

EXPRESS specification:

```
*)
TYPE geometric_set_select = SELECT
 (point,
  curve,
  surface);
END_TYPE;
(*
```

6.1.5 API_ABSTRACT_SCHEMA type definition: api specific types for structuring

This subclause declares the api-specific type resources defined for structuring the geometric representation items created by the interface functions.

6.1.5.1 Api_grouped_item

The **api_grouped_item** type specifies those objects that can be part of a **group**

EXPRESS specification:

```
*)
TYPE api_grouped_item = SELECT
 (direction,
  vector,
  placement,
  annotation_fill_area,
  fill_area_style_hatching,
  geometric_set_select,
  solid_model,
  half_space_solid,
  csg_select,
  api_group);
END_TYPE;
(*
```

6.1.5.2 Api_set_item

The **api_set_item** type specifies those objects that can be part of a **api_set**

EXPRESS specification:

```
*)
TYPE api_set_item = SELECT
 (direction,
  vector,
  placement,
  annotation_fill_area,
  geometric_set_select,
  solid_model,
  half_space_solid,
  csg_select,
  api_set);
END_TYPE;
(*
```

6.1.6 API_ABSTRACT_SCHEMA entities definition : Fundamentals of product description and support

This subclause declares the generic entities resources defined in ISO 10303-41 that are part of **api_abstract_schema**.

6.1.6.1 Shape_representation

A **shape_representation** is a specific kind of **representation** that represents a shape.

NOTE 1 - In the context of the **api_abstract_schema,** only one **shape_representation** shall exist. This **shape_representation** corresponds to the shape of the product that is created through the interface in the CAD system database. This is documented by the api-specific associated GLOBAL RULE.

NOTE 2 - In the context of the **api_abstract_schema,** the **context_of_items** shall be a **geometric-_representation_context** . This is documented by the api-specific WHERE RULE.

EXPRESS specification:

```
*)
ENTITY shape_representation
 SUBTYPE OF (representation);
WHERE
api_WR1: 'API_ABSTRACT_SCHEMA.GEOMETRIC_REPRESENTATION_CONTEXT' IN
     TYPEOF (SELF\representation.context_of_items);
END_ENTITY;
(*
```

Attribute definitions:

SELF\representation.items: A set of **representation_item**s that represent the shape of the product

SELF\representation.context_of_items: The OVC **representation_context** in which the **items** are related to form the shape of the product.

Formal propositions:

api_WR1: The **context_of_items** of the **shape_representation** shall be a **geometric_representation_context.**

Associated global rule:

The following global rule is associated with this entity and restrict its use or its relationships with other entities:

unique_shape_representation: The **unique_shape_representation** rule requires there exists an unique **shape_representation** entity in the population of the **api_abstract_schema.** This **shape_representation** corresponds to the shape of the product that is created through the interface in the CAD system database

6.1.6.2 Group

A **group** is an identification of a collection of elements.

EXPRESS specification:

```
*)
ENTITY group;
 name    : label;
 description : text;
```

```
END_ENTITY;
(*
```

Attribute definitions:

name: The word, or group of words, by which the **group** is referred to.

description: Text that relates the nature of the **group**.

6.1.6.3 Group_assignment

A **group_assignment** is an association of a **group** with product data**.**

EXPRESS Specification:

```
*)
ENTITY group_assignment
 ABSTRACT SUPERTYPE;
 assigned_group : group;
END_ENTITY;
(*
```

Attribute definitions:

assigned_group: The **group** that is to be associated with the product data.

6.1.6.4 External_source

An **external_source** is the identification of a source of product data that is not the application protocol to which the exchange conforms.

NOTE - In the context of the **api_abstract_schema, external_source**s are the locations for **externally_defined_style**s of geometric representation items.

EXPRESS specification:

```
*)
ENTITY external_source;
 source_id : source_item;
END_ENTITY;
(*
```

Attribute definitions:

source_id: The identification of the **external_source**.

6.1.6.5 Pre_defined_item

A **pre_defined_item** is the identification of information that is not explicitly represented in a given exchange but that is defined in the application protocol to which the exchange conforms.

NOTE - In the context of the **api_abstract_schema** some specific entity styles are defined as **pre_defined_item**s.

EXPRESS specification:

```
*)
ENTITY pre_defined_item;
```

```
 name : label;
END_ENTITY;
(*
```

Attribute definitions:

name: The words or group of words by which the **pre_defined_item** is referred to.

6.1.6.6 Externally_defined_item

An **externally_defined_item** is the identification of information that is not explicitly represented in a given exchange and that is not defined in the application protocol to which the exchange conforms.

NOTE - In the context of the **api_abstract_schema** entity styles are defined as **externally_defined_items**.

EXPRESS specification:

```
*)
ENTITY externally_defined_item;
 item_id : source_item;
 source : external_source;
END_ENTITY;
(*
```

Attribute definitions:

item_id: The identification of the referent item.

source: An **external_source** that contains the referent item.

6.1.7 API_ABSTRACT_SCHEMA entity definition: Representation structures

This subclause declares the generic entity resources defined in ISO 10303-43 that are part of the **api_abstract_schema**.

6.1.7.1 Representation_context

A **representation_context** is a context in which a collection of **representation_items** are related.

Two **representation_contexts** are separate and have no relationship unless a relationship is separately specified.

EXPRESS specification:

```
*)
ENTITY representation_context;
 context_identifier : identifier;
 context_type    : text;
INVERSE
 representations_in_context : SET [1:?] OF representation
  FOR context_of_items;
END_ENTITY;
(*
```

Attribute definitions:

context_identifier: An identifier of the **representation_context**.

context_type: A description of the type of a **representation_context**.

representations_in_context: At least one **representation** shall be associated with each **representation_context**.

6.1.7.2 Representation_item

A **representation_item** is an element of product data that participates in one or more **representation**s or contributes to the definition of another **representation_item**.

A **representation_item** contributes to the definition of another **representation_item** when it is referenced by that **representation_item.**

EXPRESS specification:

```
*)
ENTITY representation_item;
 name : label;
WHERE
 WR1   : SIZEOF(using_representations(SELF)) > 0;
 api_WR2: SIZEOF(using_representations(SELF)) = 1;
END_ENTITY;
(*
```

NOTE - In the context of the **api_abstract_schema**, every **representation_item** shall be associated with only one **representation_context**. It is documented by **api_WR2**.

Attribute definitions:

name: An identifier of the **representation_item**.

Formal propositions:

WR1: Every **representation_item** shall be associated with at least one **representation_context.**

api_WR2: A **representation_item** shall be referenced by one **representation_context**.

6.1.7.3 Representation

A **representation** is a collection of one or more **representation_item**s that are related in a specified **representation_context**. The use of a **representation,** i.e., that is being represented, is not specified in the part of ISO 10303.

This relationship of **representation_item** to **representation_context** is the basis for distinguishing which entities from the set of all **representation_item**s are related.

NOTE 1 - Consider the context in which a set of **geometric_representation_item**s is related to represent the shape of something. The **items** related in this context are used in this **representation**. All other **geometric_representation_item**s are specifically not included. This is the basis for distinguishing which **geometric_representation_item**s in the set of **representation_item**s are related. This distinction is not otherwise included in the specification of **representation_item**s.

The members of the set of **items** plus all **representation_item**s indirectly referenced by that set are related to the **context_of_items** by a **representation**. Indirect reference to a **representation_item** occurs when it is referenced through any number of intervening entities, each of type **representation_item**.

NOTE 2 - **representation** relates a **representation_context** to trees of **representation_item**s with each tree rooted on one member of the set of **items**. A **representation_item** is one node in the tree, and the reference of one **representation_item** to another is a branch.

The set of **representation**s directly referenced as **items**, related to the **context_of_items**, is the **representation**. The **representation_item**s indirectly referenced as described above support the definition of the **items** and are all related in the same **representation_context**.

NOTE 3 - In the **representation** of the shape of a cube by a set of **line** entities, the set of **line**s is the only **representation_item**s representing the shape. A **line** in turn references a **cartesian_point** and a **direction** that support the **line** definition and are related with each other and the **line** in the referenced **geometric_representation_context**. The shape, however, is not represented by these **cartesian_point**s and **direction**s.

A **representation** is specified to meet the needs of an application. Often a **representation** is incomplete and does not fully model the concept that is represented.

NOTE 4 - Consider a collection of two dimensional **geometric_representation_item**s used to represent the shape of a machined part. It is not a complete description of the shape, but is suitable for certain applications such as computer aided drafting.

One **representation_item** may be related to more than one **representation_context**. Two **representation**s are not related solely because the same **representation_item** is referenced directly or indirectly from their sets of **items**.

NOTE 5 - Consider a **surface** that is used in the **representation**s of the shape of a casting die and of the shape of the part cast in that die. The **surface** is geometrically founded in two distinct **geometric_representation_context**s, one for the die and one for the part. However, the **representation**s are not related. Instead, each is a separate geometric founding of the common surface. The two unrelated **representation**s simply share a common **representation_item**.

The same **representation_item** may be related multiple times to the same **representation_context** as it is used directly or indirectly in several **representation**s, each referencing the same **representation_context**. This does not have the meaning that each **representation** is creating a new instance of the same **representation_item** in the same **representation_context**. Rather, each **representation** reasserts one instance of the **representation_item** in the **representation_context** for different uses.

NOTE 6 - Consider two **representation**s, each having the same value for **context_of_items**. One is a **representation** of the shape of a cube and indirectly references a **line** as one of its edges. The second simply references the **line** as its **items**. There are not two occurrences of the **line** and its sub-tree of referenced **geometric_representation_item**s in the **geometric_representation_context**. Rather the single use of the **line** in that **geometric_representation_context** has been asserted twice, once in each **representation**. The first might exist as representing the shape of the whole cube. The other might exist as representing the shape of an edge of the same cube.

EXPRESS specification:

```
*)
ENTITY representation;
 name       : label;
 items      : SET[1:?] OF representation_item;
 context_of_items : representation_context;
END_ENTITY;
(*
```

Attribute definitions:

name: An identifier of the **representation**.

items: A set of **representation_item**s that are related in the **context_of_items**.

context_of_items: A **representation_context** in which the **items** are related to form a **representation** of some concept.

NOTE 7 - In the context of the **api_abstract_schema** all the geometric representation items created through the interface function are founded with the OVC as **context_of_item**.

6.1.7.4 Representation_map

A **representation_map** is the identification of a **representation** and a **representation_item** in that **representation** for the purpose of mapping. The **representation_item** defines the origin of the mapping. The **representation_map** is used as the source of a mapping by a **mapped_item**.

NOTE 1 - The definition of a mapping that is used to specify a new **representation_item** comprises a **representation_map** and a **mapped_item** entity. Without both entities, the mapping is not fully defined. Two entities are specified to allow the same source representation (**representation_map.mapped_representation**) to be mapped into multiple new representations (**mapped_items**).

EXPRESS specification:

```
*)
ENTITY representation_map;
 mapping_origin    : representation_item;
 mapped_representation : representation;
INVERSE
 map_usage       : SET[1:?] OF mapped_item FOR mapping_source;
WHERE
 WR1: item_in_context(SELF.mapping_origin,
            SELF.mapped_representation.context_of_items);
END_ENTITY;
(*
```

Attribute definitions:

mapping_origin: A **representation_item** about which the **mapped_representation** is mapped.

NOTE 2 - Consider the Cartesian mapping of one geometric **representation** to another. The **mapping_origin** might be an **axis2_placement** in the context of **mapped_representation** that defines the position about which it is mapped.

mapped_representation: A **representation** that is mapped to at least one **mapped_item**.

map_usage: The set of one or more **mapped_item**s to which the **representation_map** is mapped.

Formal propositions:

WR1: The **mapping_origin** shall be in the **representation_context** of the **mapped_representation**.

6.1.7.5 Mapped_item

A **mapped_item** is the use of a representation, the **mapping_source.mapped_representation**, as it participates in a **representation_map** as a **representation_item**.

NOTE 1 - A **mapped_item** is a subtype of **representation_item**. It enables a representation to be used as a **representation_item** in one or more other representations. The **mapped_item** allows for the definition of a **representation** using other **representation**s.

The mapping is achieved through an operator that is implicitly defined by the **mapping_source.mapping-_origin** and the **mapping_target** attributes. In this respect, the mapping is specified in the same way as for an **item_defined_transformation** (see ISO 10303-43, 4.4.7 for more information).

EXPRESS specification:

```
*)
ENTITY mapped_item
 SUBTYPE OF (representation_item);
 mapping_source : representation_map;
 mapping_target : representation_item;
```

```
WHERE
 WR1: acyclic_mapped_representation(using_representations(SELF), [SELF]);
END_ENTITY;
(*
```

Attribute definitions:

mapping_source: A **representation_map** that is the source of the **mapped_item**;

mapping_target: A **representation_item** that is the target onto which the **mapping_source** is mapped.

Formal propositions:

WR1: A **mapped_item** shall not be self-defining by participating in the definition of the **representation** being mapped.

NOTE 2 - The details of how any particular mapping is achieved is left to various specialisation' s of **mapped_item** and **representation_map**.

EXAMPLE - Consider the Cartesian mapping of one geometric **representation** to another. The **mapping_source** might be a **representation_map** referencing a **representation** and an **axis_placement** founded in the **geometric_representation_context** of the referenced **representation**. The **mapped_item** might be a reference to this **representation_map** and a second **axis_placement**. The **mapped_item** would then be a **representation_item** that is a mapping of the referenced **representation** such that the **representation_map.mapping_origin** is overlaid onto the **mapped_item.mapping_target**.

6.1.8 API_ABSTRACT_SCHEMA entity definition: Geometric representation structures

This subclause declares generic entity resources defined in ISO 10303-42 for geometric representation structures that are part of the **api_abstract_schema**.

6.1.8.1 Geometric_representation_context

A **geometric_representation_context** is a **representation_context** in which **geometric_representation_item**s are geometrically founded.

A **geometric_representation_context** is a distinct coordinate space, spatially unrelated to other coordinate spaces except as those coordinate spaces are specifically related by an appropriate transformation.

EXPRESS specification:

```
*)
ENTITY geometric_representation_context
 SUBTYPE OF (representation_context);
 coordinate_space_dimension : dimension_count;
END_ENTITY;
(*
```

Attribute definitions:

coordinate_space_dimension: The integer **dimension_count** of the coordinate space that is the .

NOTE - In the context of the **api_abstract_schema** the OVC constitutes the **geometric_representation_context** in which all the geometric representation items are geometrically founded. The **dimension_count** may be 2 or 3 according to the *geometrical_power_level* of the current open view.

6.1.8.2 Geometric_representation_item

A **geometric_representation_item** is a **representation_item** that has the additional meaning of having geometric position or orientation or both. This meaning is present by virtue of:

— being a **cartesian_point** or a direction;

— referencing directly a **cartesian_point** or a **direction**;

— referencing indirectly a **cartesian_point** or a **direction**.

NOTE 1 - An indirect reference to a **cartesian_point** or **direction** means that a given **geometric-_representation_item** references the **cartesian_point** or **direction** through one or more intervening attributes. In many cases this information is given in the form of an **axis2_placement**.

EXAMPLE 1 - Consider a circle. It gains its geometric position and orientation by virtue of a reference to **axis2_placement** that in turn references a **cartesian_point** and several **direction**s.

EXAMPLE 2 - A **manifold_solid_brep** is a **geometric_representation_item** that through several layers of **topological_representation_item**s, references **curve**s, **surface**s and **point**s. Through additional intervening entities, **curve**s and **surface**s reference **cartesian_point** and **direction**. See the EXPRESS definition of **manifold_solid_geometry**, **topological_representation_item** and **surface** in ISO 10303-42.

NOTE 2 - The intervening entities, that are all of type **representation_item**, need not be of subtype **geometric_representation_item**. Consider the **manifold_solid_brep** case from the above example. One of the intervening levels of **representation_item** is a **close_shell**. This is a **topological_representation_item** and does not require a **geometric_representation_context** in its own right. When used as part of the definition of a **manifold_solid_brep** that itself is a **geometric_representation_item**, it is founded in a **geometric_representation_context**. See the EXPRESS definition of **close_shell** in ISO 10303-42.

NOTE 3 - A **geometric_representation_item** inherits the need to be related to a **representation_context** in a **representation**. The rule **compatible_dimension** ensures that the **representation_context** is a **geometric_representation_context**. When in the context of geometry, this relationship causes the **geometric_representation_item** to be geometrically founded. See the definition of the term geometrically founded in ISO 10303-42.

EXPRESS specification:

```
*)
ENTITY geometric_representation_item
 SUPERTYPE OF (ONEOF(point, direction, vector, placement, curve,
           annotation_fill_area, surface, solid_model,
           boolean_result, sphere, right_circular_cone,
           right_circular_cylinder, torus, block,
           right_angular_wedge, half_space_solid,
           fill_area_style_hatching,
           one_direction_repeat_factor))
 SUBTYPE OF (representation_item);
DERIVE
 dim : dimension_count := dimension_of(SELF);
WHERE
 api_WR1: SIZEOF (QUERY (using_rep <* using_representations (SELF) |
      NOT ('API_ABSTARCT_SCHEMA.GEOMETRIC_REPRESENTATION_CONTEXT' IN
      TYPEOF (using_rep.context_of_items)))) = 0;
END_ENTITY;
(*
```

Attribute definitions:

dim: The coordinate **dimension_count** of the **geometric_representation_item**.

NOTE 3 - The **dim** attribute is derived from the **coordinate_space_dimension** of a **geometric_representation_-context** in which the **geometric_representation_item** is geometrically founded. See the definition of the term geometrically founded in ISO 10303-42.

NOTE 4 - A **geometric_representation_item** is geometrically founded in one or more **geometric_representation_contexts**, all of them having the same **coordinate_space_dimension**. See the rule **compatible_dimension** in ISO 10303-42 section 4.5.1.

NOTE 5 - In the context of the **api_abstract_schema** all the geometric representation items are subtypes of **geometric_representation_item**.

NOTE 6 - In the context of the **api_abstract_schema** only **point**s, **direction**s, **vector**s, **placement**s, **curve**s, annotation_fill_area**s**, **surface**s, **solid_model**s, **boolean_result**s, **sphere**s, **right_circular_cone**s, **right_circular_cylinder**s, **torus**, **block**s, **right_angular_wedge**s, **half_space_solid**s, **fill_area_style_hatching**s and **one_direction_repeat_factor**s shall be created by the interface. Hence the SUPERTYPE is modified.

Formal propositions:

api_WR1: In the context of **api_abstract_schema** any representation referencing a **geometric_-representation_item** shall be of the type **geometric_representation_context**.

6.1.9 API_ABSTRACT_SCHEMA entity definition: Geometric mathematical entities

This subclause declares generic entity resources defined in ISO 10303-42 that are part of the **api_abstract-_schema**.

6.1.9.1 Point

A **point** is a location in some real cartesian coordinate space R^m, for m = 1, 2 or 3.

EXPRESS specification:

```
*)
ENTITY point
 ABSTRACT SUPERTYPE OF (ONEOF(cartesian_point))
 SUBTYPE OF (geometric_representation_item);
END_ENTITY;
(*
```

NOTE - In the context of the **api_abstract_schema**, only **cartesian_point** shall exist as **point**. **Point** is then defined as an ABSTRACT SUPERTYPE, and all the other subtypes defined in ISO 10303-42 are pruned.

6.1.9.2 Cartesian_point

A **cartesian_point** is a **point** defined by its coordinates in a rectangular Cartesian coordinate system, or in a parameter space. The entity is defined in a one, two or three-dimensional space as determined by the number of coordinates in the list.

NOTE - For the purposes of defining geometry in the **api_abstract_schema** only two or three-dimensional **points** are used, and **cartesian_point** are always defined in Cartesian coordinate systems.

EXPRESS specification:

```
*)
ENTITY cartesian_point
 SUBTYPE OF (point);
 coordinates : LIST [1:3] OF length_measure;
END_ENTITY;
(*
```

Attribute definitions:

coordinates[1]: The first coordinate of the **point** location.

coordinates[2]: The second coordinate of the **point** location, this will not exist in the case of a one-dimensional point.

coordinates[3]: The third coordinate of the **point** location; this will not exist in the case of a one or two-dimensional point.

SELF\geometric_representation_item.dim: The dimensionality of the space in which the **point** is defined. This is an inherited derived attribute from the geometric representation item supertype and for a cartesian point is determined by the number of coordinates in the list.

6.1.9.3 Direction

This entity defines a general direction vector in two or three dimensional space. The actual magnitudes of the components have no effect upon the direction being defined, only the ratios x:y:z or x:y are significant.

NOTE 1 - The components of this entity are not normalised. If an unit vector is required it should be normalised before use.

EXPRESS specification:

```
*)
ENTITY direction
 SUBTYPE OF (geometric_representation_item);
 direction_ratios : LIST [2:3] OF REAL;
WHERE
 WR1  : SIZEOF(QUERY(tmp <* direction_ratios | tmp <> 0.0)) > 0;
 api_WR2: NOT((ABS(direction_ratios[1]) < EPS) AND
        (ABS(direction_ratios[2]) < EPS) AND
        (ABS(direction_ratios[3]) < EPS));
 api_WR3: NOT(((direction_ratios[1] < EPS) AND
        (direction_ratios[1] > ZERO_VALUE)) OR
        ((direction_ratios[2] < EPS) AND
        (direction_ratios[2] > ZERO_VALUE)) OR
        ((direction_ratios[3] < EPS) AND
        (direction_ratios[3] > ZERO_VALUE)));
END_ENTITY;
(*
```

NOTE 2 - In the context of the **api_abstract_schema** the additional WHERE RULEs documents the degeneration of a **direction**.

Attribute definitions:

direction_ratios[1]: The component in the direction of the X axis.

direction_ratios[2]: The component in the direction of the Y axis.

direction_ratios[3]: The component in the direction of the Z axis; this will not be present in the case of a direction in two-dimensional coordinate space.

SELF\geometric_representation_item.dim: The coordinate space dimensionality of the direction. This is an inherited attribute of the **geometric_representation_item** supertype; for this entity it is determined by the number of **direction_ratios** in the list.

Formal propositions:

WR1: The magnitude of the direction vector shall be greater than zero.

api_WR2: The magnitude of the direction vector shall be not less than EPS.

api_WR3: No value of the **direction_ratios** shall be between **EPS** and **ZERO_VALUE**.

6.1.9.4 Vector

This entity defines a vector in terms of direction and the magnitude of the vector. The value of the **magnitude** attribute defines the magnitude of the vector.

NOTE 1 - The magnitude of the vector must not be calculated from the components of the **orientation** attribute. This form of representation was selected to reduce problems with numerical instability.

EXAMPLE - A vector of magnitude 2.0 mm and equally inclined to the coordinate axes could be represented with orientation attribute of (1.0,1.0,1.0).

EXPRESS specification:

```
*)
ENTITY vector
 SUBTYPE OF (geometric_representation_item);
 orientation : direction;
 magnitude   : length_measure;
WHERE
 WR1    : magnitude >= 0.0;
 api_WR2 : MAX >= magnitude;
 api_WR3 : magnitude >= EPS;
END_ENTITY;
(*
```

NOTE 2 - In the context of the **api_abstract_schema**, the additional WHERE RULE documents the degeneration case of a **vector**.

Attribute definitions:

orientation: The direction of the **vector**.

magnitude: The magnitude of the **vector**. All vectors of **magnitude** 0.0 are regarded as equal in value regardless of the **orientation** attribute.

SELF\geometric_representation_item.dim: The dimensionality of the space in which the **vector** is defined.

Formal propositions:

WR1: The magnitude shall be positive or zero.

api_WR2: The magnitude shall be than or equal MAX.

api_WR3: The magnitude shall be greater than or equal EPS.

6.1.9.5 Placement

A **placement** locates a geometric item with respect to the coordinate system of its geometric context. It locates the item to be defined and, in the case of the axis placement subtypes, gives its orientation.

EXPRESS specification:

```
*)
ENTITY placement
 SUPERTYPE OF (ONEOF(axis1_placement,
           axis2_placement_2d,
           axis2_placement_3d))
 SUBTYPE OF (geometric_representation_item);
 location : cartesian_point;
END_ENTITY;
(*
```

Attribute definitions:

location: The geometric position of a reference point, such as the centre of a circle, of the item to be located.

6.1.9.6 Axis1_placement

The direction and location in three-dimensional space of a single axis. An **axis1_placement** is defined in terms of a locating point (inherited from the placement supertype) and an axis direction; this is either the direction of **axis** or defaults to (0.0,0.0,1.0). The actual direction for the axis placement is given by the derived attribute **z**.

NOTE - In the context of the **api_abstract_schema**, a value shall be provided for the direction of the **axis**.

EXPRESS specification:

```
*)
ENTITY axis1_placement
 SUBTYPE OF (placement);
 axis : OPTIONAL direction;
DERIVE
 z   : direction := NVL(normalise(axis), direction([0.0,0.0,1.0]));
WHERE
 WR1    : SELF\geometric_representation_item.dim = 3;
 api_WR2 : EXISTS (SELF.axis) ;
END_ENTITY;
(*
```

Attribute definitions:

SELF\placement.location: A reference point on the axis.

axis: The direction of the local Z axis.

z: The normalised direction of the local Z axis.

SELF\geometric_representation_item.dim: The space dimensionality of the **axis1_placement**, that is determined from its **location**, and is always equal to 3.

Formal propositions:

WR1: The coordinate space dimensionality shall be 3.

api_WR2: The **axis** direction shall exist.

6.1.9.7 Axis2_placement_2d

The location and orientation in two-dimensional space of two mutually perpendicular axes. An **axis2_placement_2d** is defined in terms of a point (inherited from the **placement** supertype) and an axis. It can be used to locate and orientate an object in two-dimensional space and to define a placement coordinate system. The entity includes a point that forms the origin of the placement coordinate system. A direction vector is required to complete the definition of the placement coordinate system. The **ref_direction** defines the placement X axis direction; the placement Y axis direction is derived from this.

NOTE - In the context of the **api_abstract_schema**, a value shall be provided for the **ref_direction**.

EXPRESS specification:

```
*)
ENTITY axis2_placement_2d
 SUBTYPE OF (placement);
 ref_direction : OPTIONAL direction;
DERIVE
  p      : LIST [2:2] OF direction := build_2axes(ref_direction);
WHERE
 WR1  : SELF\geometric_representation_item.dim = 2;
 api_WR2: EXISTS(SELF.ref_direction);
END_ENTITY;
(*
```

Attribute definitions:

SELF\placement.location: The spatial position of the reference point that defines the origin of the associated placement coordinate system.

ref_direction: The direction used to determine the direction of the local X axis. If **ref_direction** is omitted, this direction is taken from the geometric coordinate system.

p: The axis set for the placement coordinate system.

p[1]: The normalised direction of the placement X axis. This is (1.0,0.0) if **ref_direction** is omitted.

p[2]: The normalised direction of the placement Y axis. This is a derived attribute and is orthogonal to **p[1]**.

Formal propositions:

WR1: The space dimensionality of the **axis2_placement_2d** shall be 2.

api_WR2: The **ref_direction** shall exist.

6.1.9.8 Axis2_placement_3d

The location and orientation in three-dimensional space of two mutually perpendicular axes. An **axis2_placement_3d** is defined in terms of a point, (inherited from the placement supertype), and two (ideally orthogonal) axes. It can be used to locate and orientate a non axi-symmetric object in space and to define a placement coordinate system. The entity includes a point that forms the origin of the placement coordinate system. Two direction vectors are required to complete the definition of the placement coordinate system. The **axis** is the placement Z axis direction, and the **ref_direction** is an approximation to the placement X axis direction.

NOTE 1 - Let **z** be the placement Z axis direction and **a** be the approximate placement X axis direction. There are two methods, mathematically identical but numerically different, for calculating the placement X and Y axis directions.

a) The vector **a** is projected onto the plane defined by the origin point **P** and the vector **z** to give the placement X axis direction as **x** = <**a** - (**a** · **z**) **z**>. The placement Y axis direction is then given by **y** = <**z** x **x**>.

b) The placement Y axis direction is calculated as **y** = <**z** x **a**> and then the placement X axis direction is given by **x** = <**y** x **z**>.

The first method is likely to be the more numerically stable of the two, and is used here.

A placement coordinate system referenced by parametric equations is derived from the **axis2_placement_3d** data for conic curves and elementary surfaces.

NOTE 2 - In the context of the **api_abstract_schema**, values shall be provided for the **axis** and **ref_direction**.

EXPRESS specification:

```
*)
ENTITY axis2_placement_3d
 SUBTYPE OF (placement);
 axis     : OPTIONAL direction;
 ref_direction : OPTIONAL direction;
DERIVE
 p      : LIST [3:3] OF direction := build_axes(axis,ref_direction);
WHERE
 WR1  : SELF\placement.location.dim = 3;
 WR2  : (NOT (EXISTS (axis))) OR (axis.dim = 3);
 WR3  : (NOT (EXISTS (ref_direction))) OR (ref_direction.dim = 3);
 WR4  : (NOT (EXISTS (axis))) OR (NOT (EXISTS (ref_direction))) OR
         (cross_product(axis,ref_direction).magnitude > 0.0);
 api_WR5: EXISTS (axis) AND EXISTS (ref_direction);
 api_WR6: cross_product(axis, ref_direction).magnitude >= EPS;
END_ENTITY;
(*
```

NOTE 3 - In the context of the **api_abstract_schema**, **api_WR6** documents the degeneration case of the **axis2_placement_3d**.

Attribute definitions:

SELF\placement.location: The spatial position of the reference point and origin of the associated placement coordinate system.

axis: The exact direction of the local Z axis.

ref_direction: The direction used to determine the direction of the local X axis. If necessary an adjustment is made to maintain orthogonality to the **axis** direction. If **axis** and/or **ref_direction** is omitted, these directions are taken from the geometric coordinate system.

p: The axes for the placement coordinate system. The directions of these axes are derived from the attributes with appropriate default values if required.

p[1]: The normalised direction of the local X axis.

p[2]: The normalised direction of the local Y axis.

p[3]: The normalised direction of the local Z axis.

NOTE 4 - See **Figure 3** for interpretation of attributes.

Formal propositions:

WR1: The space dimensionality of the **SELF\placement.location** shall be 3.

WR2: The space dimensionality of **axis** shall be 3.

WR3: The space dimensionality of **ref_direction** shall be 3.

WR4: The **axis** and the **ref_direction** shall not be parallel or anti-parallel. (This is required by the **build_axes** function.)

api_WR5: The **axis** and **ref_direction** shall exist.

api_WR6: The magnitude of the cross product of **axis** and **direction** shall be less than or equal MAX.

api_WR7: The magnitude of the cross product of **axis** and **direction** shall be greater than or equal EPS.

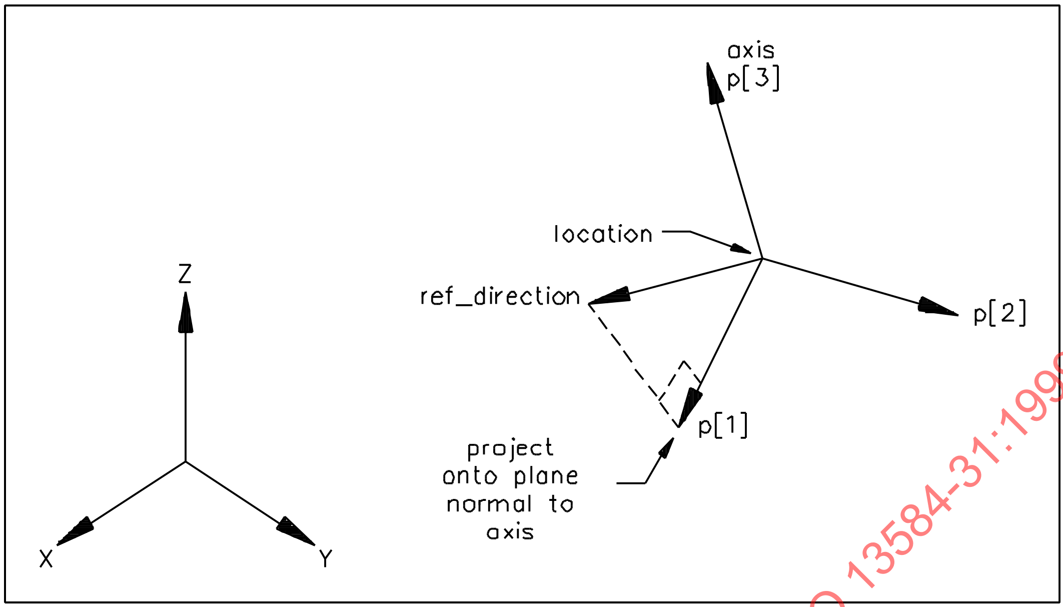


Figure 3 — Axis2 placement 3D

6.1.10 API_ABSTRACT_SCHEMA entity definition: Geometric curves entities

This subclause declares the generic entity resources for **curves** defined in ISO 10303-42 that are part of the **api_abstract_schema.** These entities, with the exception of the **line** entity, may not be created directly by the interface functions. They may only be created indirectly to represent the interface specific entities.

The **line** entity may be created directly to permit constraint-based description of other entities. Nevertheless, the **line** entity is considered as a mathematical entity and is associated with a **null_style**.

6.1.10.1 Curve

A **curve** can be envisioned as the path of a point moving in its coordinate space.

EXPRESS specification:

```
*)
ENTITY curve
 SUPERTYPE OF (ONEOF(line, conic, surface_curve))
 SUBTYPE OF (geometric_representation_item);
END_ENTITY;
(*
```

NOTE - In the context of the **api_abstract_schema** only **line**s, **conic**s, and **surface_curve**s shall be created by the interface. Hence the SUPERTYPE is pruned.

Informal propositions:

IP1: A **curve** shall be arcwise connected.

IP2: A **curve** shall have an arc length greater than zero.

api_IP3: A **curve** shall have an arc length greater than EPS.

6.1.10.2 Line

A line is an unbounded curve with constant tangent direction. A **line** is defined by a **point** and a **direction**. The positive direction of the line is in the direction of the **dir** vector.

The curve is parametrised as follows:

$$\mathbf{P} = \mathrm{pnt}$$

$$\mathbf{V} = \mathrm{dir}$$

$$l\,(u) = \mathbf{P} + u\mathbf{V}$$

and the parametric range is $-\infty < u < \infty$

NOTE - In the context of the **api_abstract_schema, line**s may be created directly to permit constraint-based definition of other entities. It may also created implicitly as **basis_curve** of a **trimmed_curve** that is an **api_line**.

EXPRESS specification:

```
*)
ENTITY line
 SUBTYPE OF (curve);
 pnt : cartesian_point;
 dir : vector;
WHERE
 WR1 : dir.dim = pnt.dim;
END_ENTITY;
(*
```

Attribute definitions:

pnt: The location of the **line**.

dir: The direction of the **line**. The magnitude and units of **dir** affect the parametrisation of the line.

SELF\geometric_representation_item.dim: The dimensionality of the coordinate space for the **line**. This is an inherited attribute from the geometric representation item supertype.

Formal propositions:

WR1: Pnt and **dir** shall both be 2D or both be 3D entities.

6.1.10.3 Bounded_curve

A **bounded_curve** is a **curve** of finite arc length with identifiable end points.

EXPRESS specification:

```
*)
ENTITY bounded_curve
 SUPERTYPE OF (ONEOF(polyline, trimmed_curve, bounded_surface_curve,
          composite_curve))
 SUBTYPE OF (curve);
END_ENTITY;
(*
```

NOTE - In the context of the **api_abstract_schema**, only **polyline**s, **trimmed_curve**s, **bounded_surface_curve** and **composite_curve**s shall be created by the interface. Hence the SUPERTYPE is pruned.

6.1.10.4 Trimmed_curve

A trimmed curve is a bounded curve that is created by taking a selected portion, between two identified points, of the associated basis curve. The basis curve itself is unaltered and more than one **trimmed_curve** may reference the same basis curve. Trimming points for the curve may be identified:

— by parametric value, or

— by geometric position, or

— by both.

At least one of these shall be specified at each end of the curve. The **sense** makes it possible to unambiguously define any segment of a closed curve such as a circle. The combinations of sense and ordered end points make it possible to define four distinct directed segments connecting two different points on a circle or other closed curve. For this purpose cyclic properties of the parameter range are assumed.

EXAMPLE 1 - 370 degrees is equivalent to 10 degrees.

The **trimmed_curve** has a parametrisation that is inherited from that of the particular basis curve referenced. More precisely the parameter *s* of the **trimmed_curve** is derived from the parameter *t* of the basis curve as follows:

If sense is TRUE: $s = t - t_1$.

If sense is FALSE: $s = t_2 - t$.

In the above equations, t_1 is the value given by trim_1 or the parameter value corresponding to point_1 and t_2 is the parameter value given by trim_2 or the parameter corresponding to point_2. The resultant **trimmed_curve** has a parameter *s* ranging from 0 at the first trimming point to $|t_2 - t_1|$ at the second trimming point.

NOTE 1 - In the case of a closed basis curve, it may be necessary to increment t_1 or t_2 by the parametric length for consistency with the sense flag.

EXAMPLE 2 - If **sense_agreement** = TRUE and $t_2 < t_1$, then t_2 should be increased by the cyclic range.

EXAMPLE 3 - If **sense_agreement** = FALSE and $t_1 < t_2$, then t_1 should be increased by the cyclic range.

EXPRESS specification:

```
*)
ENTITY trimmed_curve
 SUPERTYPE OF (ONEOF (api_line, api_circular_arc, api_elliptical_arc,
           api_hyperbolic_arc, api_parabolic_arc))
 SUBTYPE OF (bounded_curve);
 basis_curve      : curve;
 trim_1        : SET[1:2] OF trimming_select;
 trim_2        : SET[1:2] OF trimming_select;
 sense_agreement    : BOOLEAN;
 master_representation : trimming_preference;
WHERE
 WR1: (HIINDEX(trim_1) = 1) XOR (TYPEOF(trim_1[1]) <> TYPEOF(trim_1[2]));
 WR2: (HIINDEX(trim_2) = 1) XOR (TYPEOF(trim_2[1]) <> TYPEOF(trim_2[2]));
END_ENTITY;
(*
```

NOTE 2 - In the context of the **api_abstract_schema**, specific SUBTYPES are defined to permit the specification of the range or the target of some interface functions.

NOTE 3 - In the context of the **api_abstract_schema master_representation** shall be implementation dependent.

NOTE 4 - In the context of the **api_abstract_schema** and in the case of a closed **basic_curve**, the closed **trimmed_curve** that corresponds to the complete closed **basic_curve** shall be represented by using parametric value for trimming point identifications.

EXAMPLE 2 - The circular arc defined by **sense_agreement** = FALSE, **trim_1** = 450 and **trim_2** = 90 is a closed clockwise oriented circular arc whose both trimming points are defined by the intersection of the **circle basis_curve** with the y axis of its position **axis2_placement**.

Attribute definitions:

basis_curve: The **curve** to be trimmed. For curves with multiple representations any parameter values given as **trim_1** or **trim_2** refer to the master representation of the **basis_curve** only.

trim_1: The first trimming point that may be specified as a cartesian point (point_1), as a real parameter value (parameter_1 = t_1), or both.

trim_2: The second trimming point that may be specified as a cartesian point (point_2), as a real parameter value (parameter_2 = t_2), or both.

sense_agreement: Flag to indicate whether the direction of the **trimmed_curve** agrees with or is opposed to the direction of **basis_curve**.

— ***sense_agreement*** = *TRUE* if the curve is being traversed in the direction of increasing parametric value;

— ***sense_agreement*** = *FALSE* otherwise.

For an open curve, ***sense_agreement*** = *FALSE* if $t_1 > t_2$. If $t_2 > t_1$, then ***sense_agreement*** = *TRUE*.

The sense information is redundant in this case but is essential for a closed curve.

master_representation: Where both parameter and point are present at either beginning or end of the curve this indicates the preferred form. Multiple representations provide the ability to communicate data more than one form, even though the data is expected to be geometrically identical.

NOTE 5 - The **master_representation** attribute acknowledges the impracticality of ensuring that multiple forms are indeed identical and allows the indication of a preferred form. This would probably be determined by the creator of the data. All characteristics, such as parametrisation, domain, and results of evaluation, for an entity having multiple representations, are derived from the master representation. Any use of the other representations is a compromise for practical considerations.

Formal propositions:

WR1: Either a single value is specified for **trim_1**, or, the two trimming values are of different types (point and parameter).

WR2: Either a single value is specified for **trim_2**, or, the two trimming values are of different types (point and parameter).

Informal propositions:

IP1: Where both the parameter value and the **cartesian_point** exist for **trim_1** or **trim_2,** they shall be consistent, i.e., the **basis_curve** evaluated at the parameter value shall coincide with the specified point.

IP2: When a **cartesian_point** is specified by **trim_1** or by **trim_2**, it shall lie on the **basis_curve**.

IP3: Except in the case of a closed **basis_curve**, where both **parameter_1** and **parameter_2** exist they shall be consistent with the sense flag, i.e., **sense** = (**parameter_1** < **parameter_2**).

IP4: If both **parameter_1** and **parameter_2** exist then **parameter_1** <> **parameter_2**.

IP5: When a parameter value is specified by **trim_1** or **trim_2,** it shall lie within the parametric range of the **basis_curve**.

6.1.10.5 Composite_curve

A **composite_curve** is a collection of curves joined end-to-end. The individual segments of the **curve** are themselves defined as **composite_curve_segment**s. The parametrisation of the composite curve is an accumulation of the parametric ranges of the referenced **bounded_curve**s. The first segment is parametrised from 0 to l_1, and, for $i \geq 2$, the i^{th} segment is parametrised from

$$\sum_{k=1}^{k=i-1} l_k \quad \text{to} \quad \sum_{k=1}^{k=i} l_k$$

where l_k is the parametric length (i.e. difference between maximum and minimum parameter values) of the curve underlying the k^{th} segment.

NOTE 1 - In the context of the **api_abstract_schema**, **composite_curve**s shall only be created either to represent an interface **api_contour** or to represent a **boundary_curve** of a **curve_bounded_surface**, both of them shall be planar, closed and not **self_intersect**ing.

EXPRESS specification:

```
*)
ENTITY composite_curve
 SUBTYPE OF (bounded_curve);
 segments    : LIST [1:?] OF composite_curve_segment;
 self_intersect : LOGICAL;
DERIVE
 n_segments    : INTEGER := SIZEOF(segments);
 closed_curve   : BOOLEAN
         := segments[n_segments].transition <> discontinuous;
WHERE
 WR1  : ((NOT closed_curve) AND (SIZEOF(QUERY(temp <* segments |
          temp.transition = discontinuous)) = 1)) OR
        ((closed_curve) AND (SIZEOF(QUERY(temp <* segments |
          temp.transition = discontinuous)) = 0));
 api_WR2: closed_curve ;
 api_WR3: NOT self_intersect ;
END_ENTITY;
(*
```

NOTE 2 - In the context of the **api_abstract_schema** the additional WHERE RULE documents the requirement for a **composite_curve** created by the interface to be closed.

Attribute definitions:

n_segments: The number of component curves.

segments: The component bounded curves, their transitions and senses. The transition attribute for the last segment defines the transition between the end of the last segment and the start of the first; this transition attribute may take the value **discontinuous**, that indicates an open curve. (See 6.1.3.2 of this international standard).

self_intersect: Indication of whether the curve intersects itself or not; this is for information only.

dim: The dimensionality of the coordinate space for the composite curve. This is an inherited attribute from the geometric representation item supertype.

closed_curve: Indication of whether the curve is closed or not; this is derived from the transition code on the last segment.

NOTE 3 - See **Figure 4** for further information on attributes.

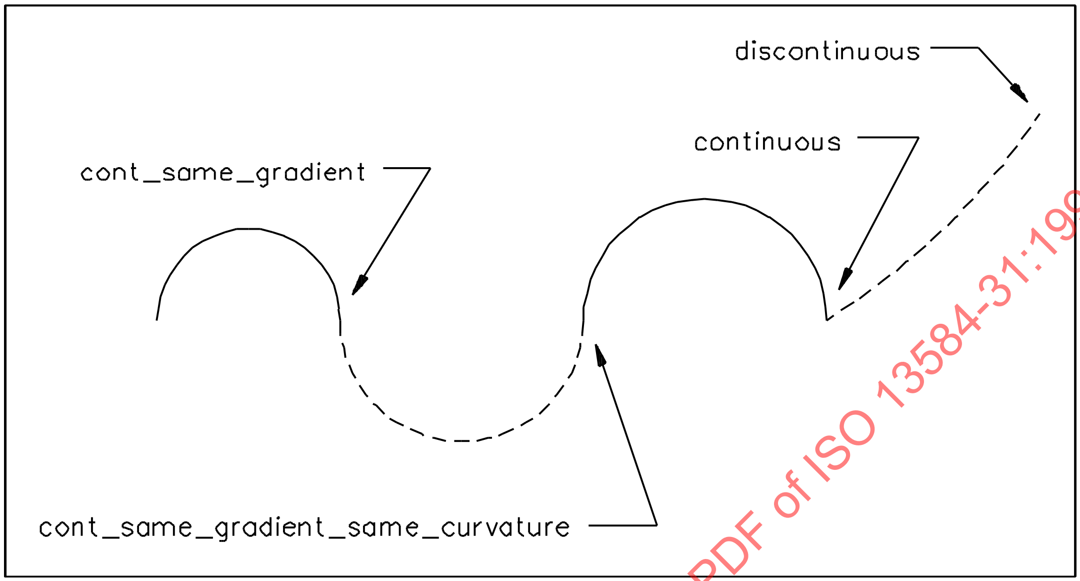


Figure 4 — Composite curve

Formal propositions:

WR1: No **transition** code shall be discontinuous, except for the last code of an open curve.

api_WR2: The composite_curve shall be closed.

api_WR3: The **composite_curve** shall not be self intersecting.

Informal propositions:

IP1: The **same_sense** attribute of each segment correctly specifies the senses of the component curves. When traversed in the direction indicated by **same_sense** the segments shall join end to end.

api_IP2: The **composite_curve** shall be planar.

6.1.10.6 Composite_curve_segment

A **composite_curve_segment** is a bounded curve together with transition information that is used to construct a **composite_curve**.

NOTE - In the context of the **api_abstract_schema**, **composite_curve_segment**s are automatically computed by the interface when creating an **api_contour** or an **api_planar_surface**. Hence **transition** shall not be discontinuous.

EXPRESS specification:

```
*)
ENTITY composite_curve_segment;
 transition   : transition_code;
 same_sense   : BOOLEAN;
 parent_curve : curve;
INVERSE
 using_curves : BAG[1:?] OF composite_curve FOR segments;
```

```
WHERE
 WR1  : ('API_ABSTRACT_SCHEMA.BOUNDED_CURVE' IN TYPEOF(parent_curve));
 api_WR2: (transition = continuous) OR (transition = cont_same_gradient);
END_ENTITY;
(*
```

Attribute definitions:

transition: The state of transition (i.e. geometric continuity from the last point of this segment to the first point of the next segment) in a **composite_curve**.

same_sense: An indicator of whether or not the sense of the segment agrees with, or opposes, that of the

parent_curve. If **same_sense** is false, the point with highest parameter value is taken as the first

point of the segment.

parent_curve: The **bounded_curve** that defines the geometry of the segment.

using_curves: The set of **composite_curve**s that use this **composite_curve_segment** as a segment. This set shall not be empty.

Formal propositions:

WR1: The **parent_curve** shall be a **bounded_curve**

api_WR2: The **transition** is either **continuous** or **cont_same_gradient**.

6.1.10.7 Surface_curve

A **surface_curve** is a curve on a surface. The curve is represented as a curve (**curve_3d**) in three-dimensional space and possibly as a curve, corresponding to a **pcurve**, in the two dimensional parametric space of a surface. The ability of this curve to reference a list of 1 or 2 **pcurve_or_surface**s enables this entity to define either a curve on a single surface, or an intersection curve that has two distinct surface associations. A 'seam' on a closed surface can also be represented by this entity; in this case each **associated_geometry** will be a **pcurve** lying on the same surface. Each pcurve, if it exists, shall be parametrised to have the same sense as **curve_3d**. The surface curve takes its parametrisation directly from either **curve_3d** or a **pcurve** as indicated by the attribute **master_representation**.

NOTE 1 - In the context of the **api_abstract_schema**, **surface_curves** are automatically computed by the interface when creating an **api_planar_surface**, they shall reference a **plane**.

EXPRESS specification:

```
*)
ENTITY surface_curve
 SUPERTYPE OF (bounded_surface_curve)
 SUBTYPE OF (curve);
 curve_3d        : curve;
 associated_geometry  : LIST[1:2] OF pcurve_or_surface;
 master_representation : preferred_surface_curve_representation;
DERIVE
 basis_surface     : SET[1:2] OF surface
            := get_basis_surface(SELF);
WHERE
 WR1   : curve_3d.dim = 3;
 WR2   : ('GEOMETRY_SCHEMA.PCURVE' IN TYPEOF(associated_geometry[1])) OR
                (master_representation <> pcurve_s1);
 WR3   : ('GEOMETRY_SCHEMA.PCURVE' IN TYPEOF(associated_geometry[2])) OR
                (master_representation <> pcurve_s2);
 WR4   : NOT ('GEOMETRY_SCHEMA.PCURVE' IN TYPEOF(curve_3d));
 api_WR5 : master_representation = curve_3D;
 api_WR6 : SIZEOF(SELF.associated_geometry) = 1;
 api_WR7 : 'API_ABSRACT_SCHEMA.PLANE' IN
                   TYPEOF (SELF.associated_geometry [1]);
 api_WR8 : SELF.associated_geometry[1] :=: SELF.basis_surface;
END_ENTITY;
(*
```

Attribute definitions:

curve_3d: The curve that is the three-dimensional representation of the **surface_curve**.

associated_geometry: A list of one or two pcurves or surfaces that define the surface or surfaces associated with the surface curve. Two elements in this list indicate that the curve has two surface associations that need not be two distinct surfaces. When a pcurve is selected it identifies a surface and also associates a basis curve in the parameter space of this surface.

NOTE 2 - In the context of the **api_abstract_schema**, **associated_geometry** refers to the **plane** of the **api_planar_surface**.

master_representation: Indication of representation "preferred". The master representation defines the curve used to determine the unique parametrisation of the surface curve.

The master representation takes one of the values **curve_3d**, **pcurve_s1** or **pcurve_s2** to indicate a preference for the 3D curve, or the first or second pcurve, in the associated geometry list, respectively. Multiple representations provide the ability to communicate data in more than one form, even though the data is expected to be geometrically identical.

NOTE 3 - The **master_representation** attribute acknowledges the impracticality of ensuring that multiple forms are indeed identical and allows the indication of a preferred form. This would probably be determined by the creator of the data. All characteristics, such as parametrisation, domain, and results of evaluation, for an entity having multiple representations, are derived from the master representation. Any use of the other representations is a compromise for practical considerations.

NOTE 4 - In the context of the **api_abstract_schema** the **master_representation** shall be the **curve_3d**.

basis_surface: The surface on which the **surface_curve** lies. This is determined from the first element of the **associated_geometry** list.

NOTE 5 - In the context of the **api_abstract_schema** this surface is the **plane** of the **api_planar_surface**.

Formal propositions:

WR1: The **curve_3d** shall be defined in three-dimensional space.

WR2: The **pcurve_s1** shall only be nominated as the master representation if the first element of the **associated_geometry** list is a **pcurve**.

WR3: The **pcurve_s2** shall only be nominated as the master representation if the second element of the associated geometry list is a pcurve. This also requires that **pcurve_s2** shall not be nominated when the **associated_geometry** list contains a single element.

WR4: The **curve_3d** shall not be a **pcurve**.

api_WR5: The **master_representation** shall be the **curve_3D**.

api_WR6: The **associated_geometry** shall contain only one element.

api_WR7: The unique **associated_geometry** item shall be a **plane**.

api_WR8: The derived **basis_surface** shall be the same entity as the unique entity contained in **associated_geometry**.

Informal propositions:

IP1: Where **curve_3d** and one or more **pcurve**s exist, they shall represent the same mathematical point set. (i.e. They shall coincide geometrically but may differ in parametrisation.)

IP2: The **curve_3d** and any associated pcurves shall agree with respect to their senses.

6.1.10.8 Composite_curve_on_surface

A **composite_curve_on_surface** is a collection of segments that are curves on a surface. Each segment shall lie on the basis surface, and may be

— a **surface_curve** or

— a **pcurve** or

— a **composite_curve_on_surface**.

NOTE 1 - A **composite_curve_on_surface** can be included as the **parent_curve** attribute of a **composite_curve_segment** since it is a **bounded_curve** subtype.

NOTE 2 - In the context of the **api_abstract_schema**, each segment shall be a **surface_curve**.

There shall be at least positional continuity between adjacent segments. The parametrisation of the **composite_curve** is obtained from the accumulation of the parametric ranges of the segments. The first segment is parametrised from 0 to l_1, and, for i_2, the i^{th} segment is parametrised from

$$\sum_{k=1}^{k=\iota-1} l_k \quad \text{to} \quad \sum_{k=1}^{k=i} l_k$$

where l_k is the parametric length (i.e. difference between maximum and minimum parameter values) of the k^{th} curve segment.

EXPRESS specification:

```
*)
ENTITY composite_curve_on_surface
 SUPERTYPE OF (boundary_curve)
 SUBTYPE OF (composite_curve);
DERIVE
 basis_surface : SET[0:2] OF surface := get_basis_surface(SELF);
WHERE
 WR1  : SIZEOF(basis_surface) > 0;
 WR2  : constraints_composite_curve_on_surface(SELF);
 api_WR3: SIZEOF(QUERY(temp<*SELF\composite_curve.segments |
           'GEOMETRY_SCHEMA.PCURVE' IN TYPEOF (temp.parent_curve)
            )  -- end of query
         ) = 0;
END_ENTITY;
(*
```

Attribute definitions:

basis_surface: The surface on which the composite curve is defined.

SELF\composite_curve.n_segments: The number of component curves.

SELF\composite_curve.segments: The component bounded curves, their transitions and senses. The transition for the last segment defines the transition between the end of the last segment and the start of the first; this element may take the value **discontinuous**, that indicates an open curve.

NOTE 3 - In the context of the **api_abstract_schema** the inherited api specific WHERE RULE ensures that the transition is not **discontinuous**.

SELF\composite_curve.self_intersect: Indication of whether the curve intersects itself or not.

SELF\composite_curve.dim: The dimensionality of the coordinate space for the **composite_curve**.

SELF\composite_curve.closed_curve: Indication of whether the curve is closed or not.

Formal propositions:

WR1: The **basis_surface** SET shall contain at least one surface. This ensures that all segments reference curves on the same surface.

WR2: Each segment shall reference a **pcurve**, or a **surface_curve**, or a **composite_curve_on_surface**.

api_WR3: No segment shall reference a **pcurve**

Informal propositions:

IP1: Each **parent_curve** referenced by a **composite_curve_on_surface** segment shall be a **curve_on_surface** and a bounded curve.

6.1.10.9 Bounded_surface_curve

A **bounded_surface_curve** is a specialised subtype of **surface_curve** that also has the properties of a **bounded_curve**.

EXPRESS specification:

```
*)
ENTITY bounded_surface_curve
 SUBTYPE OF (surface_curve, bounded_curve);
WHERE
api_WR1 : ('API_ABSTRACT_SCHEMA.BOUNDED_CURVE' IN
      TYPEOF (SELF\surface_curve.curve_3d);
END_ENTITY;
(*
```

Formal propositions:

api_WR1: The **curve_3d** attribute of the **surface_curve** supertype shall be a **bounded_curve**.

6.1.11 API_ABSTRACT_SCHEMA entity definition: Geometric conic entities

This subclause declares the generic entity resources for **curves** defined in ISO 10303-42 that are part of the **api_abstract_schema.** These entities may not be created directly by the interface functions. They may only be created indirectly to represent the interface specific entities.

6.1.11.1 Conic

A **conic** is a planar curve that could be produced by intersecting a plane with a cone.

A **conic** curve is defined in terms of its intrinsic geometric properties rather than being described in terms of other geometry.

A **conic** entity always has a Placement Coordinate System defined by **axis2_placement**; the parametric representation is defined in terms of this Placement Coordinate System.

NOTE - In the context of the **api_abstract_schema**, conics shall only be created as the **basis_curve** of **api_circular_arc**, **api_elliptical_arc**, **api_hyperbolic_arc**, or **api_parabolic-_arc**.

EXPRESS specification:

```
*)
ENTITY conic
 ABSTRACT SUPERTYPE OF (ONEOF(circle, ellipse, hyperbola, parabola))
 SUBTYPE OF (curve);
 position: axis2_placement;
WHERE
 api_WR1: SIZEOF( USEDIN(SELF,
          'API_ABSTRACT_SCHEMA.TRIMMED_CURVE.BASIS_CURVE') ) = 1;
END_ENTITY;
(*
```

Attribute definitions:

position: The location and orientation of the **conic**. Further details of the interpretation of this attribute are given for the individual subtypes.

Formal propositions:

api_WR1: Each **conic** shall be used as **basis_curve** by one **trimmed_curve**.

6.1.11.2 Circle

A **circle** is defined by a radius and the location and orientation of the circle. Interpretation of the data shall be as follows:

$\mathbf{C}$ = position.location (centre)

$\mathbf{x}$ = position.p[1]

$\mathbf{y}$ = position.p[2]

$\mathbf{z}$ = position.p[3]

R = radius

and the **circle** is parametrised as

$$\lambda(u) = \mathbf{C} + \boldsymbol{R}((\cos u)\mathbf{x} + (\sin u)\mathbf{y})$$

The parametrisation range is $0 \leq u \leq 360$ degrees.

In the **placement** coordinate system defined above, the **circle** is the equation $C = 0$, where

$$C(x,y,z) = x^2 + y^2 - R^2$$

The positive sense of the **circle** at any point is in the tangent direction, T, to the **curve** at the **point,** where

$$\mathbf{T} = (-C_y, C_x, 0)$$

NOTE 1 - A circular arc is defined by using the **trimmed_curve** entity in conjunction with the circle entity.

NOTE 2 - In the context of the **api_abstract_schema**, a **circular_arc** is defined by the **api_circular_arc** entity.

NOTE 3 - In the context of the **api_abstract_schema**, **circle** shall only be created by the interface as **basic_curve** of **api_circular_arc**s.

EXPRESS specification:

```
*)
ENTITY circle
 SUBTYPE OF (conic);
 radius : positive_length_measure;
END_ENTITY;
(*
```

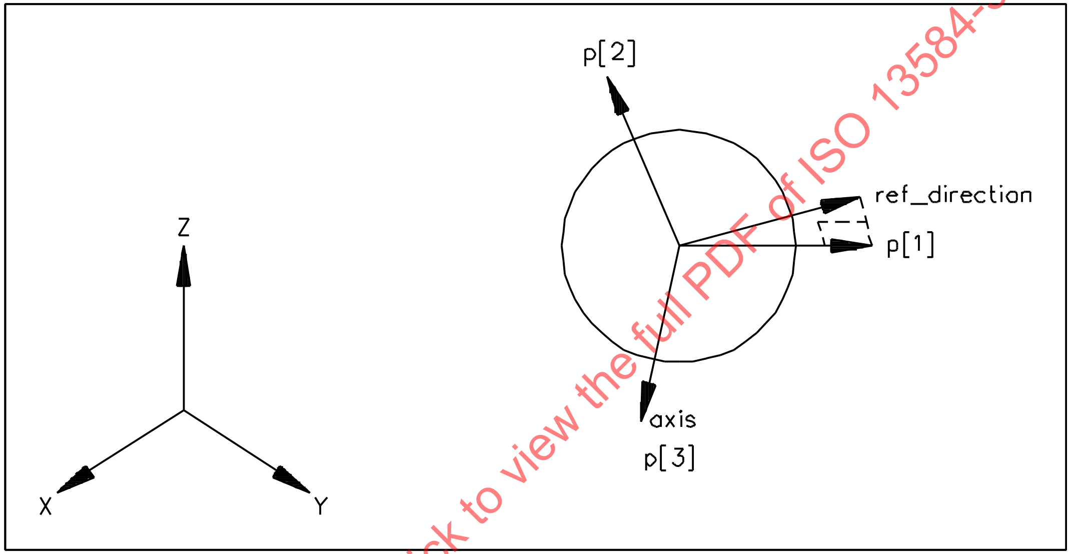


Figure 5 — Circle

Attribute definitions:

SELF\conic.position.location: This inherited attribute defines the centre of the circle.

radius: The radius of the circle, that shall be greater than zero.

NOTE 4 - See **Figure 5** for interpretation of attributes.

6.1.11.3 Ellipse

An **ellipse** is a conic section defined by the lengths of the semi-major and semi-minor diameters and the position (centre or mid point of the line joining the foci) and orientation of the curve.

Interpretation of the data shall be as follows:

C = position.location

x = position.p[1]

y = position.p[2]

z = position.p[3]

$$R_1 = \text{semi\_axis\_1}$$

$$R_2 = \text{semi\_axis\_2}$$

and the ellipse is parametrised as

$$\lambda(u) = \mathbf{C} + (R_1 \cos u)\mathbf{x} + (R_2 \sin u)\mathbf{y}$$

The parametrisation range is $0 \leq u \leq 360$ degrees.

In the **placement** coordinate system defined above, the **ellipse** is the equation $C = 0$, where

$$C(x,y,z) = x^2/R_1^2 + y2/R_2^2 - 1$$

The positive sense of the **ellipse** at any point is in the tangent direction, T, to the **curve** at the point, where

$\mathbf{T} = (-C_y, C_x, 0)$.

NOTE 1 - In the context of the **api_abstract_schema**, **ellipse**s shall only be created as **basis_curve**s of **api_elliptical_arc**s.

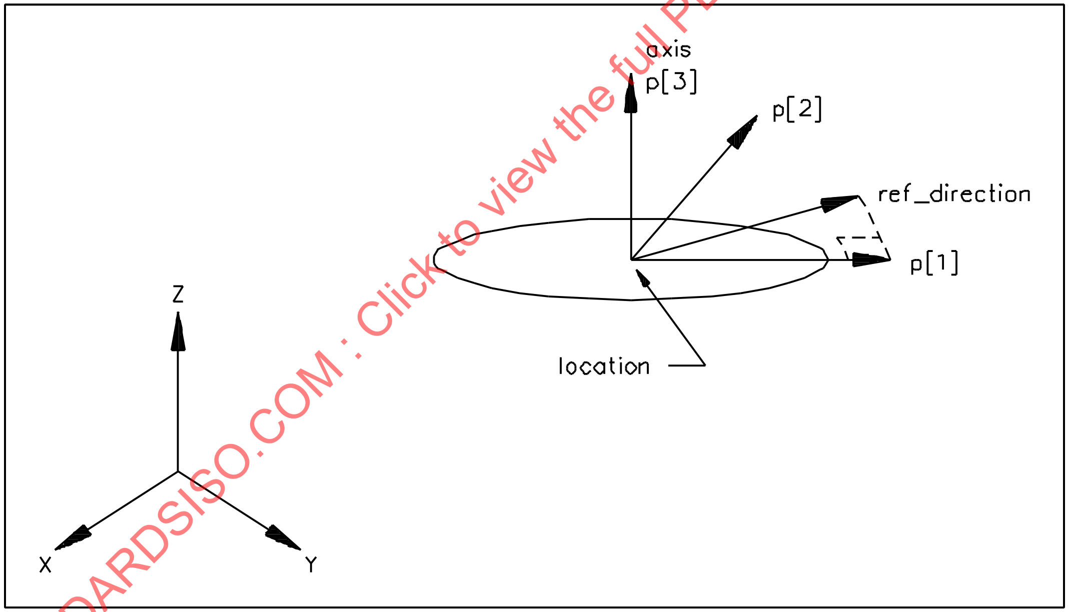


Figure 6 — Ellipse

EXPRESS specification:

```
*)
ENTITY ellipse
 SUBTYPE OF (conic);
 semi_axis_1 : positive_length_measure;
 semi_axis_2 : positive_length_measure;
END_ENTITY;
(*
```

Attribute definitions:

SELF\conic.position: conic.position.location is the centre of the ellipse, and **conic.position.p[1]** is the direction of the **semi_axis_1**.

semi_axis_1: The first radius of the ellipse that shall be positive.

semi_axis_2: The second radius of the ellipse that shall be positive.

NOTE 2 - See **Figure 6** for interpretation of attributes.

6.1.11.4 Hyperbola

A **hyperbola** is a conic section defined by the lengths of the major and minor radii and the position (mid point of the line joining two foci) and orientation of the curve. Interpretation of the data shall be as follows:

$\mathbf{C}$ = position.location

$\mathbf{x}$ = position.p[1]

$\mathbf{y}$ = position.p[2]

$\mathbf{z}$ = position.p[3]

R_1 = semi_axis

R_2 = semi_imag_axis

and the **hyperbola** is parametrised as

$$\lambda(u) = \mathbf{C} + (R_1 \cosh u)\mathbf{x} + (R_2 \sinh u)\mathbf{y}$$

The parametrisation range is $-\infty < u < \infty$

In the **placement** coordinate system defined above, the **hyperbola** is represented by the equation $C = 0$, where

$$C(x,y,z) = x^2/R_1^2 - y^2/R_2^2 - 1$$

The positive sense of the **hyperbola** at any point is in the tangent direction, ***T***, to the curve at the point, where

$$\mathbf{T} = (-C_y, C_x, 0).$$

The branch of the **hyperbola** represented is that pointed to by the **x** direction.

NOTE 1 - In the context of the **api_abstract_schema**, **hyperbola**s shall only be created as **basis_curve**s of **api_hyperbolic_arc**s

EXPRESS specification:

```
*)
ENTITY hyperbola
 SUBTYPE OF (conic);
 semi_axis    : positive_length_measure;
 semi_imag_axis : positive_length_measure;
END_ENTITY;
(*
```

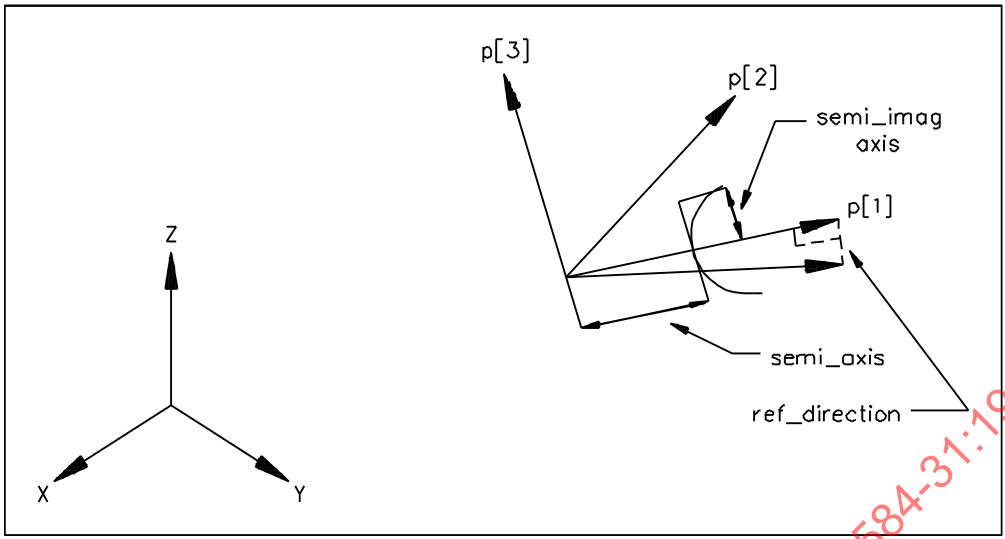


Figure 7 — Hyperbola

Attribute definitions:

SELF\conic.position: The location and orientation of the curve. **conic.position.location** is the centre of the hyperbola and **conic.position.p[1]** is in the direction of the semi-axis. The branch defined is on the side of **position.p[1]** positive.

semi_axis: The length of the semi axis of the hyperbola. This is positive and is half the minimum distance between the two branches of the hyperbola.

semi_imag_axis: The length of the semi imaginary axis of the hyperbola that shall be positive.

NOTE 2 - See **Figure 7** for interpretation of attributes.

Formal propositions:

WR1: The length of the **semi_axis** shall be greater than zero.

WR2: The length of the **semi_imag_axis** shall be greater than zero.

6.1.11.5 Parabola

A **parabola** is a conic section defined by its focal length, position (apex), and orientation.

Interpretation of the data shall be as follows:

$\mathbf{C}$ = position.location

$\mathbf{x}$ = position.p[1]

$\mathbf{y}$ = position.p[2]

$\mathbf{z}$ = position.p[3]

F = focal_dist

and the **parabola** is parametrised as

$$\lambda(u) = \mathbf{C} + F(u^2\mathbf{x} + 2u\mathbf{y})$$

The parametrisation range is $-\infty < u < \infty$.

In the **placement** coordinate system defined above, the **parabola** is represented by the equation $C = 0$, where

$$C(x,y,z) = 4Fx - y^2$$

The positive sense of the **curve** at any point is in the tangent direction, **T**, to the curve at the point, where

$$\mathbf{T} = (-C_y, C_x, 0)$$

NOTE 1 - In the context of the **api_abstract_schema**, parabolas shall only be created as **basis_curve**s of **parabolics_arc**s.

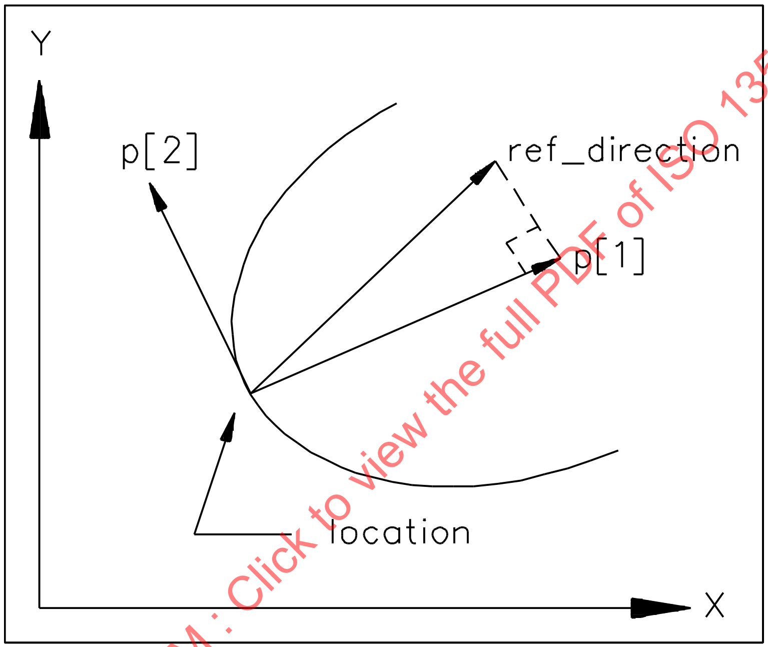


Figure 8 — Parabola

EXPRESS specification:

```
*)
ENTITY parabola
 SUBTYPE OF (conic);
 focal_dist : length_measure;
WHERE
 WR1: focal_dist <> 0.0;
END_ENTITY;
(*
```

Attribute definitions:

SELF\conic.position: The location and orientation of the curve. **conic.position.location** is the apex of the parabola, and **conic.position.p[1]** is the axis of symmetry.

focal_dist: The distance of the focal point from the apex point.

NOTE 2 - See **Figure 8** for interpretation of attributes.

Formal propositions:

WR1: The focal distance shall not be zero.

6.1.12 API_ABSTRACT_SCHEMA entity definition: api specific basic curves

This subclause declares the api specific basic curves entities that may be computed and generated by the interface through constraint-based specification. Hence, utility functions are provided by the interface to obtain the characteristics of such entities. The orientation of these entities, from **trim_1** to **trim_2**, that shall be consistent with **sense_agreement,** is used to avoid ambiguousness in geometric constructions. These entities shall exist in the target modelling system. Hence no simulation process is defined for their implementation.

6.1.12.1 Api_line

An **api_line** is a **trimmed_curve** of one linear segment. It is defined by using a **trimmed_curve** entity in conjunction with a **line** entity.

EXPRESS specification:

```
*)
ENTITY api_line
 SUBTYPE OF (trimmed_curve) ;
WHERE
 api_WR1 : 'API_ABSTRACT_SCHEMA.LINE' IN
                  TYPEOF (SELF\TRIMMED_CURVE.BASIS_CURVE);
END_ENTITY;
(*
```

NOTE 1 - This api specific entity is introduced to permit the specification of the range of some interface functions.

NOTE 2 - In the context of the **api_abstract_schema**, the **master_representation** shall be implementation dependent.

NOTE 3 - This entity may be implemented as a **trimmed_curve**.

Attribute definitions:

SELF\trimmed_curve.basis_curve: The **line** to be trimmed.

SELF\trimmed_curve.trim_1: The first trimming point that may be specified as a **cartesian_point** (point_1), as a real parameter value (parameter_1 = t_1), or both.

SELF\trimmed_curve.trim_2: The second trimming point that may be specified as a **cartesian_point** (point_2), as a real parameter value (parameter_2 = t_2), or both.

SELF\trimmed_curve.sense_agreement: Flag to indicate whether the direction of the **trimmed_curve** agrees with or is opposed to the direction of **basis_curve**.

master_representation: Where both parameter and point are present at either end of the curve this indicates the preferred form. Multiple representations provide the ability to communicate data more than one form, even though the data is expected to be geometrically identical.

Formal propositions:

api_WR1: The **basis_curve** of the **trimmed_curve** shall be a **line**.

Informal propositions:

api_IP1: The length of the **api_line** shall not be less than **EPS** and greater than **MAX**.

6.1.12.2 Api_circular_arc

An **api_circular_arc** is a **trimmed_curve** of one circular segment. It is defined by using a **trimmed_curve** entity in conjunction with a **circle** entity.

EXPRESS specification:

```
*)
ENTITY api_circular_arc
 SUBTYPE OF (trimmed_curve);
WHERE
 api_WR1 : 'API_ABSTRACT_SCHEMA.CIRCLE' IN
                  TYPEOF (SELF\TRIMMED_CURVE.BASIS_CURVE);
END_ENTITY;
(*
```

NOTE 1 - This api specific entity is introduced to permit the specification of the range of some interface functions.

NOTE 2 - In the context of the **api_abstract_schema**, the **master_representation** shall be implementation dependent.

NOTE 3 - This entity may be implemented as a **trimmed_curve**.

NOTE 4 - If the two trimming points of the **api_circular_arc** are identical, the **api_circular_arc** is the whole **circle** with the direction defined by the **sense_agreement**.

Attribute definitions:

SELF\trimmed_curve.basis_curve: The **circle** to be trimmed.

SELF\trimmed_curve.trim_1: The first trimming point that may be specified as a **cartesian_point** (point_1), as a real parameter value (parameter_1 = t_1), or both.

SELF\trimmed_curve.trim_2: The second trimming point that may be specified as a **cartesian_point** (point_2), as a real parameter value (parameter_2 = t_2), or both.

SELF\trimmed_curve.sense_agreement: Flag to indicate whether the direction of the **trimmed_curve** agrees with or is opposed to the direction of **basis_curve**.

master_representation: Where both parameter and point are present at either end of the curve this indicates the preferred form. Multiple representations provide the ability to communicate data more than one form, even though the data is expected to be geometrically identical.

Formal propositions:

api_WR1: The **basis_curve** of the **trimmed_curve** shall be a circle.

Informal propositions:

api_IP1: The arc length of the **api_circular_arc** shall not be less than EPS.

6.1.13 API_ABSTRACT_SCHEMA entity definition: api specific conic arcs

This subclause declares the api specific conic arc entities that may be generated by the interface functions. When a function that creates a conic arc is triggered, a **conic** is first created as the **basis_curve** of the conic arc, then the conic arc is created as a SUBTYPE of a **trimmed_curve**.

The entities defined in this subclause are introduced to permit the specification of the range of some interface functions.

NOTE - The conic arcs may be implemented as **trimmed_curve**s.

If conic entities do not exist in the target modelling system a simulation has to be performed by the interface. This simulation is realised, for each entity, through interpolation. This interpolation is based on the two end points of the entity to be simulated, and on a number of interior points defined by the *interpolation_nodes_number* entry of the interface status table. The interpolating curve must be continuous and with a continuous tangent and must preserve the tangents to the entity at both its end points. The curve used in this interpolation is implementation dependent. It may be, for instance, a **circular_arc**. It may be a curve type defined only internally by the target modelling system and by the interface (e.g. Bezier curves). The selection of interior interpolation points is also implementation dependent. The only specified requirement is that they must be, in some sense, uniformly distributed.

The *interpolation_nodes_number* entry of the interface status table may be interrogated by the application program. It may also be set provided that the number of interpolation points are less than or equal to the *max_interpolation_nodes_number* value defined in the interface description table. The *max_interpolation-_nodes_number* shall be greater than or equal to 1.

6.1.13.1 Api_elliptical_arc

An **api_elliptical_arc** is a **trimmed_curve** of one **ellipse** curve segment. It is defined by using a **trimmed_curve** entity in conjunction with an **ellipse** entity.

EXPRESS specification:

```
*)
ENTITY api_elliptical_arc
 SUBTYPE OF (trimmed_curve);
WHERE
 api_WR1 : 'API_ABSTRACT_SCHEMA.ELLIPSE' IN
                 TYPEOF (SELF\TRIMMED_CURVE.BASIS_CURVE);
END_ENTITY;
(*
```

NOTE 1 - This api specific entity is introduce to permit the specification of the range of some interface functions.

NOTE 2 - In the context of the **api_abstract_schema**, the **master_representation** shall be implementation dependent.

NOTE 3 - This entity may be implemented as a **trimmed_curve**.

NOTE 4 - If the two trimming points of the **api_elliptical_arc** are identical, then the **api_elliptical_arc** is the whole ellipse, with the direction defined by the **sense_agreement**.

Attribute definitions:

SELF\trimmed_curve.basis_curve: The **ellipse** to be trimmed.

SELF\trimmed_curve.trim_1: The first trimming point that may be specified as a **cartesian_point** (point_1), as a real parameter value (parameter_1 = t_1), or both.

SELF\trimmed_curve.trim_2: The second trimming point that may be specified as a **cartesian_point** (point_2), as a real parameter value (parameter_2 = t_2), or both.

SELF\trimmed_curve.sense_agreement: Flag to indicate whether the direction of the **trimmed_curve** agrees with or is opposed to the direction of **basis_curve**.

master_representation: Where both parameter and point are present at either end of the curve this indicates the preferred form. Multiple representations provide the ability to communicate data more than one form, even though the data is expected to be geometrically identical.

Formal propositions:

api_WR1: The **basis_curve** of the **trimmed_curve** shall be an **ellipse**.

Informal propositions:

IP1: The arc length of the **api_elliptical_arc** shall not be less than EPS.

6.1.13.2 Api_hyperbolic_arc

An **api_hyperbolic_arc** is a **trimmed_curve** of one hyperbola curve segment. It is defined by using a **trimmed_curve** entity in conjunction with a **hyperbola.**

EXPRESS specification:

```
*)
ENTITY api_hyperbolic_arc
 SUBTYPE OF (trimmed_curve);
WHERE
 api_WR1 : 'API_ABSTRACT_SCHEMA.HYPERBOLA' IN
                  TYPEOF (SELF\TRIMMED_CURVE.BASIS_CURVE);
END_ENTITY;
(*
```

NOTE 1 - This api specific entity is introduce to permit the specification of the range of some interface functions.

NOTE 2 - In the context of the **api_abstract_schema**, the **master_representation** shall be implementation dependent.

NOTE 3 - This entity may be implemented as a **trimmed_curve**.

Attribute definitions:

SELF\trimmed_curve.basis_curve: The **hyperbola** to be trimmed.

SELF\trimmed_curve.trim_1: The first trimming point that may be specified as a **cartesian_point** (point_1), as a real parameter value (parameter_1 = t_1), or both.

SELF\trimmed_curve.trim_2: The second trimming point that may be specified as a **cartesian_point** (point_2), as a real parameter value (parameter_2 = t_2), or both.

SELF\trimmed_curve.sense_agreement: Flag to indicate whether the direction of the **trimmed_curve** agrees with or is opposed to the direction of **basis_curve**.

master_representation: Where both parameter and point are present at either end of the curve this indicates the preferred form. Multiple representations provide the ability to communicate data more than one form, even though the data is expected to be geometrically identical.

Formal propositions:

api_WR1: The **basis_curve** of the **trimmed_curve** shall be an **hyperbola**.

Informal propositions:

api_IP1: The arc length of the **api_hyperbolic_arc** shall not be less than EPS.

6.1.13.3 Api_parabolic_arc

An **api_parabolic_arc** is a **trimmed_curve** of one **parabola** curve segment. It is defined by using a **trimmed_curve** entity in conjunction with a **parabola** entity.

EXPRESS specification:

```
*)
ENTITY api_parabolic_arc
 SUBTYPE OF (trimmed_curve);
WHERE
 api_WR1 : 'API_ABSTRACT_SCHEMA.PARABOLA' IN
                  TYPEOF (SELF\TRIMMED_CURVE.BASIS_CURVE);
END_ENTITY;
(*
```

NOTE 1 - This api specific entity is introduced to permit the specification of the range of some interface functions.

NOTE 2 - In the context of the **api_abstract_schema**, the **master_representation** shall be implementation dependent.

NOTE 3 - This entity may be implemented as a **trimmed_curve**.

Attribute definitions:

SELF\trimmed_curve.basis_curve: The **parabola** to be trimmed.

SELF\trimmed_curve.trim_1: The first trimming point that may be specified as a **cartesian_point** (point_1), as a real parameter value (parameter_1 = t_1), or both.

SELF\trimmed_curve.trim_2: The second trimming point that may be specified as a **cartesian_point** (point_2), as a real parameter value (parameter_2 = t_2), or both.

SELF\trimmed_curve.sense_agreement: Flag to indicate whether the direction of the **trimmed_curve** agrees with or is opposed to the direction of **basis_curve**.

master_representation: Where both parameter and point are present at either end of the curve this indicates the preferred form. Multiple representations provide the ability to communicate data more than one form, even though the data is expected to be geometrically identical.

Formal propositions:

api_WR1: The **basis_curve** of the **trimmed_curve** shall be a **parabola**.

Informal propositions:

IP1: The arc length of the **api_parabolic_arc** shall not be less than EPS.

6.1.14 API_ABSTRACT_SCHEMA entity definition: curve entities

This subclause declares the two curve entities that may be created through the interface functions: the **polyline**, that is a generic entity resource defined in ISO 10303-42 and that is part of the **api_abstract_schema,** and the api specific **api_contour**.

6.1.14.1 Polyline

A **polyline** is a **bounded_curve** of *n-1* linear segments defined by a list of *n* **points**, P_1, P_2, P_n.

The *i*th segment of the curve is parametrised as follows:

$$\lambda(u) = \mathbf{P}_i(i-u) + \mathbf{P}_{i+1}(u+1-i), \quad for\ 1 \leq i \leq n-1$$

where $i - 1 \leq u \leq i$ and with parametric range of $0 \leq u \leq n - 1$.

NOTE 1 - If the **polyline** does not exist in the CAD system, it shall be simulated through connected lines. The maximum number of points per polyline that shall be allowed by the interface implementation is equal to or greater than that specified in clause **10** of this International Standard.

NOTE 2 - In the context of the **api_abstract_schema** the length of each linear segment shall not be less than EPS and greater than MAX.

EXPRESS specification:

```
*)
ENTITY polyline
 SUBTYPE OF (bounded_curve);
 points : LIST [2:?] OF cartesian_point;
END_ENTITY;
(*
```

Attribute definitions:

points: The **point**s defining the **polyline**.

Informal propositions:

api_IP1: The length of each linear segment shall not be less than EPS and greater than MAX.

6.1.14.2 Api_contour

An **api_contour** is a non self-intersecting oriented planar closed **composite_curve** built up by the interface from basic entities, conic arcs and/or **polyline**s. An **api_contour** cuts the plane in two subsets. The bounded subset is called the interior. The interface shall ensure that the contour is closed, and that the entities that result from transformations of a contour by geometric manipulation functions provided by the interface remain closed.

An **api_contour** is defined by the application program as an unordered list of **curves** entities. Any basic entities (i.e., **api_line**s, **api_circular_arc**s), conics arcs or **polyline**s may be used to define an **api_contour** provided that :

1) for any extremity of an entity, there exists exactly one extremity of another entity in a neighbourhood of **ZERO_VALUE** ;

2) the curve obtained by connecting these entities by their neighbouring extremities shall be planar, closed, non self-intersecting.

These two assertions are first checked by the interface. During this process the list of the entities defined by the application program is logically re-ordered within the interface. The first entity of the re-ordered list is the first entity of the initial list. The second entity is the entity that contains the only extremity that is in the neighbouring of the end extremity of the first entity, and so on. The beginning point of the first entity shall be both the beginning point and the end point of the contour.

If both conditions are true, the **api_contour** is computed by the interface. This process is performed in two steps. (1) Some entities may be simulated. (2) The resulting entities are slightly adjusted to ensure **api_contour** closure.

1) **Api_contour**s are defined for building **annotation_fill_area**, **api_planar_surfaces** and **solid** bodies. Hence, some entities may not be allowed in contour representation (e.g. because they are not supported by the CAD system). Only the basic curves entities (i.e., **api_line**s, and **api_circular_arc**s) shall be permitted by any interface in contour representation. If some other curve entity, used in the contour generation function, is not permitted by the interface for contour representation, this entity shall be simulated by the simulation process defined for this entity.

NOTE 1 - The entities that are permitted for contour representation are defined in the *contour_entities* entries of the interface description table (see **8.1**). The maximum number of entities per **api_contour** that shall be allowed by the interface implementation is equal to or greater than that specified in clause **10** of this International Standard.

2) A closed **api_contour** is then constructed by the following process.

   1) The first **curve** of the re-ordered list is duplicated, together with its **basis_curve** if it is a **trimmed_curve**.

   2) A first **composite_curve_segment** is built, using this duplicated **curve** as **parent_curve** and with **same_sense** equals **true**.

   3) The **direction** of the tangent vector to this current c**omposite_curve_segment** at its end point is then computed. The **direction** of the tangent vector to the next **curve** in the re-ordered **curve** list at its extremity that is in the neighbourhood of the end point of the current **composite_curve_segment** is also computed. In this process, the next **curve** of the last **curve** of the **curve** re-ordered list is the first curve.

   4) If both **direction** are parallel, the **transition** of the current **composite_curve_segment** is set to **cont_same_gradient**, else it is set to **continuous.**

   5) Until the end of the **curve** list, each **curve** is used to compute a **composite_curve_segment** using a similar process.

      a) The curve is duplicated, together with its **basic_curve** if it is a trimmed_curve.

      b) A **composite_curve_segment** is built, using this duplicated curve as **parent_curve**. This **composite_curve_segment** is called the current **composite_curve_segment**.

      c) If the first extremity of the duplicated curve is the neighbouring extremity of the end extremity of the previous **composite_curve_segment** then the **same_sense** attribute of the current **composite_curve_segment** is set to true. Else, it is sent to false. This attribute defines the orientation of the current **composite_curve_segment**, and then, its beginning point and end point.

      d) If the beginning point of the current **composite_curve_segment** is not identical to the end point of the previous **composite_curve_segment**, the **basis_curve** of the current **composite_curve_segment** is translated to ensure **api_contour** closure.

      e) The direction of the tangent vectors to the current **composite_curve_segment** at its end point, and to the next curve in the re-ordered curve list at its extremity that is in the neighbourhood of the end point of the current **composite_curve_segment** are both computed. If both directions are parallel, the transition of the current **composite_curve_segment** is set to **cont_same_gradient**, else it is set to continuous. In the process, the next curve of the last curve of the curve re-ordered list is the first curve.

   6) If the end point of the last **composite_curve_segment** and the beginning point of the first **composite_curve_segment** are identical, the **api_contour** is created, consisting of the ordered list of computed **composite_curve_segment**s. If these two points are not identical, and if the **transition** codes of the two last **composite_curve_segment**s are both continuous, the last **composite_curve_segment** is slightly modified to ensure contour closure. If the contour is not closed and if either of the **transition** code of the two last **composite_curve_segment**s **cont_same_gradient**, then the interface tries to approximate the last **composite_curve_segment** is by one or two **composite_curve_segment**s. The approximation process is implementation dependent but it shall ensure:

      — contour closure;

      — correction of the **transition** code of the last but one previous **composite_curve_segment**;

— a value for the **transition** code(s) of the approximating entity (entities) is equal to the value of the transition code of the approximated **composite_curve_segment**;

— that the extend of the approximated entity (entities) is (are) greater than EPS.

If the interface cannot compute such an approximation, then the last entity is moved slightly to ensure contour closure, and the transition code of the last two **composite_curve_segment**s are modified accordingly. In the process, circularity of the **curve** re-ordered list is always assumed: the previous entity of the first entity is the last entity.

NOTE 2 - It is always possible, for the application program, to avoid any entity modification during contour construction: it shall use only permitted contour entities (at the minimum: **api_line** and **api_circular_arc**) and ensure contour closure.

<u>EXPRESS specification:</u>

```
*)
ENTITY api_contour
 SUBTYPE OF (composite_curve);
END_ENTITY;
(*
```

NOTE 3 - This api-specific entity is introduced to permit the specification of the range of some interface functions.

NOTE 4 - This entity may be implemented as a **composite_curve**.

<u>Attribute definitions:</u>

SELF\composite_curve.segments: The ordered list of the **composite_curve_segment**s that are computed by the interface and that constitute the **api_contour**.

SELF\composite_curve.self_intersect: A LOGICAL attribute that shall have a **false** value and that indicates that an **api_contour** is not self intersecting.

SELF\composite_curve.n_segments: The number of entities that was provided by the application program plus one if the computation process defined above required an additional **composite_curve_segment** to ensure **api_contour** closure.

SELF\composite_curve.closed_curve: A Boolean attribute that shall have a **true** value and that indicates **api_contour** closure.

<u>Informal proposition:</u>

api_IP1: The interior of a contour shall be capable of containing a circle of diameter EPS.

6.1.15 API_ABSTRACT_SCHEMA entity definition: fill area

This subclause declares the generic entity resources for fill area that are defined in ISO 10303-46 and that are part of the **api_abstract_schema**. A fill area is modelled through an **annotation_fill_area**. In the context of the **api_abstract_schema**, an **annotation_fill_area** is only allowed in 2D views, i.e., when the interface is open with *geometrical_power_level* equals 1. An **annotation_fill_area** is planar connective 2-manifold whose external boundary is an **api_contour** and that may have internal boundaries defined by **api_contour**s. The maximum number of inner boundaries that shall be allowed by the interface implementation is equal to or greater than that specified in clause **10** of this International Standard. All the contours are in the same plane and shall not intersect each other. All the (possible) contours that define the internal boundaries of the fill area shall belong to the interior of the contour that defines its external boundary and have non intersecting interiors. In the context of the **api_abstract_schema**, an **annotation_fill_area** plays two roles.

1) It may be hatched, the hatching being defined by an **annotation_fill_area_occurrence** that assigned a fill style **fill_area_style_hatching** onto the representation item **annotation_fill_area** ;

2) It may be filled by the current fill area style colour that shall always be the background colour, and then, it may hide or blank out other entities if the global values of both entries in the interface status table are equal to ON for *hidden_line* entry and equal to TRUE for *hidden_line_involved* entry.

6.1.15.1 Annotation_fill_area

An **annotation_fill_area** is a set of **curves** that may be filled with hatching, shading, colour or tiling. The **annotation_fill_area** is described by boundaries, that consist of non-intersecting and non-self-intersecting closed **curves**. These **curves** form the boundary of planar areas to be filled according to the style for the **annotation_fill_area**. The filling is defined by the following rules:

— A **curve** that is not surrounded by any other **curve** is a border between an unfilled area on the outside and a filled area on the inside.

NOTE 1 - see **Figure 9** (a)

— A **curve** (curve 2) surrounds an unfilled area if it is surrounded by another **curve** (curve 1) whose inside is a filled area.

NOTE 2 - see **Figure 9** (b)

— If a third **curve** (curve 3) is placed inside of curve 2 this **curve** surrounds a filled area.

NOTE 3 - see **Figure 9** (c)

— For each additional **curve** the procedure is applied in the same manner.

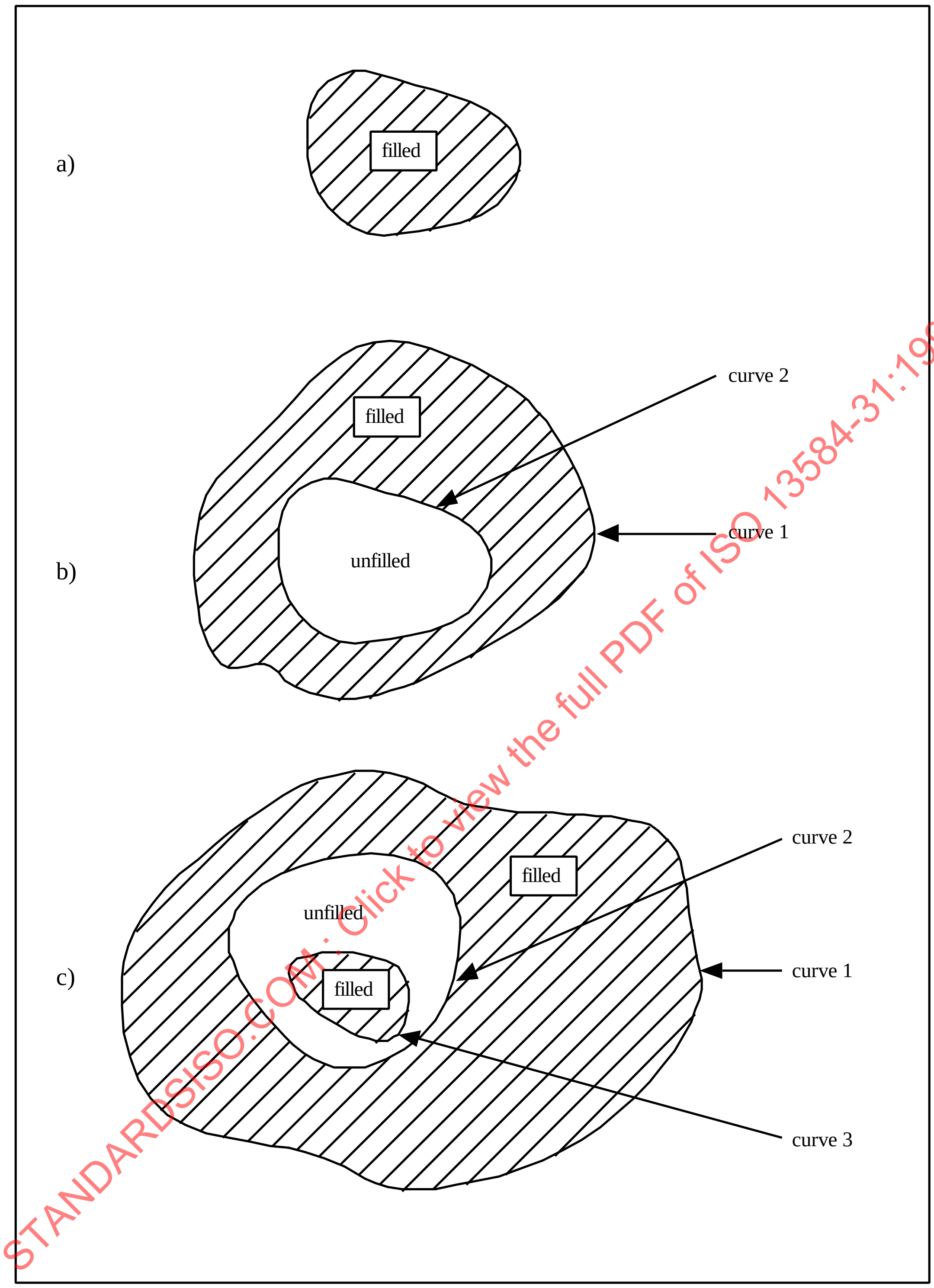


Figure 9 — Filling of annotation fill areas

EXPRESS specification:

```
*)
ENTITY annotation_fill_area
 SUBTYPE OF (geometric_representation_item);
 boundaries : SET [1:?] OF curve;
WHERE
api_WR1: SIZEOF(QUERY( temp <* SELF.boundaries |
           'API_ABSTRACT_SCHEMA.API_CONTOUR' IN TYPEOF (SELF) )
        ) = SIZEOF(SELF.boundaries);
END_ENTITY;
(*
```

Attribute definitions:

boundaries: A set of **curves** that define the boundaries of the fill area.

Formal propositions:

api_WR1: All the boundaries shall be **api_contour**s.

Informal propositions:

IP1: All the **curves** in the set **SELF.boundaries** shall be closed and planar.

IP2: If there are two or more **curves** in the set **SELF.boundaries,** all of these **curves** shall be coplanar, and no two curves shall intersect each other.

IP3: The x axis and the y axis of **SELF.filling_position** shall be coplanar with the curve **SELF.boundaries** [1].

api_IP3: If there are two or more **api_contour**s in the set **SELF.boundaries**, the distance between two **api_contour**s shall be not less than EPS.

6.1.16 API_ABSTRACT_SCHEMA entity definition : Geometric surface entities

This subclause declares the generic entity resources for **surfaces** defined in ISO 10303-42 that are part of the **api_abstract_schema.** These entities may not be created directly by the interface functions. They may only be created indirectly to represent the interface specific entities.

6.1.16.1 Surface

A **surface** can be envisioned as a set of connected points in 3-dimensional space that is always locally 2-dimensional, but need not be a manifold. A surface shall not be a single point or in part, or entirely, a curve.

NOTE 1 - For more information see 3.1 and 4.4.48 of ISO 10303-42.

EXPRESS specification:

```
*)
ENTITY surface
 SUPERTYPE OF (ONEOF(elementary_surface, bounded_surface))
 SUBTYPE OF (geometric_representation_item);
END_ENTITY;
(*
```

NOTE 2 - In the context of the **api_abstract_schema**, only **plane**s, and **api_planar_surface**s are allowed. Hence the SUPERTYPE is pruned.

Informal propositions:

IP1: A **surface** has non zero area.

IP2: A **surface** is arcwise connected.

6.1.16.2 Elementary surface

An **elementary_surface** is a simple analytic surface with defined parametric representation.

EXPRESS specification:

```
*)
ENTITY elementary_surface
 SUPERTYPE OF (ONEOF(plane))
 SUBTYPE OF (surface);
 position : axis2_placement_3d;
END_ENTITY;
(*
```

NOTE - In the context of the **api_abstract_schema**, only **plane**s are allowed as **elementary_surface**s. Hence the SUPERTYPE is pruned.

Attribute definitions:

position: The position and orientation of the surface. This attribute is used in the definition of the parametrisation of the surface.

6.1.16.3 Plane

A **plane** is an unbounded surface with a constant normal. A **plane** is defined by a point on the plane and the normal direction to the plane. The data is to be interpreted as follows:

C = position.location

x = position.p[1]

y = position.p[2]

z = position.p[3] = normal to plane

and the surface is parametrised as

$$\sigma(u,v) = \mathbf{C} + \mathbf{x}u + \mathbf{y}v$$

where the parametrisation range is $-\infty < u,v < +\infty$. In the above parametrisation, the length unit for the unit vectors **x** and **y** is derived from the context of the plane.

EXPRESS specification:

```
*)
ENTITY plane
 SUBTYPE OF (elementary_surface);
END_ENTITY;
(*
```

Attribute definitions:

SELF\elementary_surface.position: The location and orientation of the surface. This attribute is inherited from the **elementary_surface** supertype.

position.location: A point in the plane.

position.p[3]: This direction, that is equal to **position.axis** defines the normal to the plane.

6.1.16.4 Bounded_surface

A **bounded_surface** is a surface of finite area with identifiable boundaries.

EXPRESS specification:

```
*)
ENTITY bounded_surface
 SUPERTYPE OF (ONEOF(curve_bounded_surface))
 SUBTYPE OF (surface);
END_ENTITY;
(*
```

NOTE - In the context of the **api_abstract_schema**, a **bounded_surface** may only created as a **curve_bounded_surface** entity. Hence the SUPERTYPE is pruned.

Informal propositions:

IP1: A **bounded_surface** has a finite non-zero surface area.

IP2: A **bounded_surface** has boundary curves.

6.1.16.5 Curve_bounded_surface

The **curve_bounded_surface** is a parametric surface with curved boundaries defined by one or more boundary curves. One of these may be the outer boundary; any number of inner boundaries is permissible. The outer boundary may be defined implicitly as the natural boundary of the surface; this is indicated by the **implicit_outer** flag being true. In this case, at least one inner boundary shall be defined. For certain types of closed surface (e.g. cylinder), it may not be possible to identify any given boundary as outer. The region of the **curve_bounded_surface** in the **basis_surface** is defined to be the portion of the basis surface in the direction of *N x T* from any point on the boundary, where **N** is the surface normal and **T** the boundary curve tangent vector at this point. The region so defined shall be arcwise connected.

NOTE 1 - In the context of the **api_abstract_schema**, a **curve_bounded_surface** may only be created as its **api_planar_surface** subtype.

EXPRESS specification:

```
*)
ENTITY curve_bounded_surface
 SUBTYPE OF (bounded_surface);
 basis_surface   : surface;
 boundaries      : SET [1:?] OF boundary_curve;
 implicit_outer  : BOOLEAN;
WHERE
 WR1: NOT(implicit_outer AND
          ('API_ABSTRACT_SCHEMA.OUTER_BOUNDARY_CURVE' IN
          TYPEOF(boundaries)));
 WR2: (NOT(implicit_outer)) OR
          ('API_ABSTRACT_SCHEMA.BOUNDED_SURFACE' IN
          TYPEOF(basis_surface));
 WR3: SIZEOF(QUERY(temp <* boundaries |
          'API_ABSTRACT_SCHEMA.OUTER_BOUNDARY_CURVE' IN
          TYPEOF(temp))) <= 1;
 WR4: SIZEOF(QUERY( temp <* boundaries |
          (temp\composite_curve_on_surface.basis_surface [1] :<>:
                         SELF.basis_surface))) = 0;
END_ENTITY;
(*
```

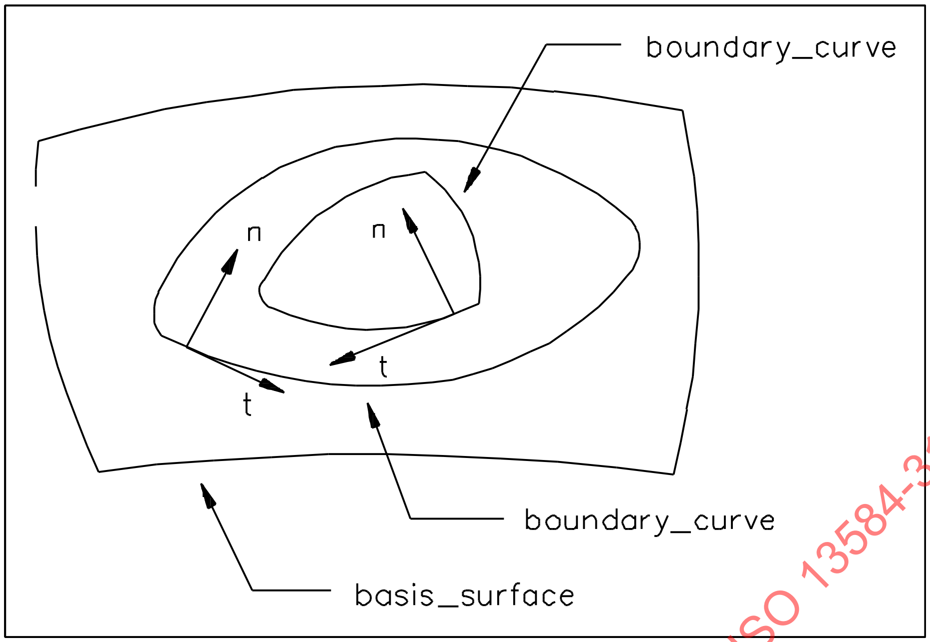


Figure 10 — Curve bounded surface

Attribute definitions:

basis_surface: The surface to be bounded.

boundaries: The bounding curves of the surface, other than the implicit outer boundary, if present. At most one of these may be identified as an outer boundary by being of type **outer_boundary_curve**.

implicit_outer: A logical flag that, if true, indicates the natural boundary of the surface is used as an outer boundary.

NOTE 2 - See **Figure 10** for interpretation of these attributes.

Formal propositions:

WR1: No explicit outer boundary shall be present when **implicit_outer** is TRUE.

WR2: The outer boundary shall only be implicitly defined if the **basis_surface** is bounded.

WR3: At most one outer boundary curve shall be included in the list of boundaries.

WR4: Each **boundary_curve** shall lie on the **basis_surface**. This is verified from the **basis_surface** attribute of the **composite_curve_on_surface** supertype for each element of the **boundaries** list.

Informal propositions:

IP1: Each curve in the set of **boundaries** shall be closed.

IP2: No two curves in the set of **boundaries** shall intersect.

IP3: At most, one of the boundary curves may enclose any other boundary curve. If an **outer_boundary_curve** is designated, then only that curve may enclose any other boundary curve.

6.1.16.6 Boundary_curve

A **boundary_curve** is a type of bounded curve suitable for the definition of a surface boundary.

EXPRESS specification:

```
*)
ENTITY boundary_curve
 SUBTYPE OF (composite_curve_on_surface);
WHERE
 WR1: SELF\composite_curve.closed_curve;
END_ENTITY;
(*
```

Formal propositions:

WR1: The derived **closed_curve** attribute of the **composite_curve** supertype shall be TRUE.

6.1.16.7 Outer_boundary_curve

This is a special sub-type of **boundary_curve** that has the additional semantics of defining an outer boundary of a surface. No more than one such curve shall be included in the set of **boundaries** of a **curve_bounded_surface**.

EXPRESS specification:

```
*)
ENTITY outer_boundary_curve
 SUBTYPE OF (boundary_curve);
END_ENTITY;
(*
```

6.1.17 API_ABSTRACT_SCHEMA entity definition : api specific surface entities

This subclause declares the only surface entity that may be created directly by the interface functions.

6.1.17.1 Api_planar_surface

The **api_planar_surface** is the only surface entity that may be created by the interface. An **api_planar_surface** is specified through an **api_contour** that correspond to the external boundary of the surface, and a list of **api_contour** that correspond to the (possible) inner boundaries of the surface. The maximum number of inner boundaries that shall be allowed by the interface implementation is equal to or greater than that specified in clause **10** of this International Standard. All the contours shall be in the same plane and shall not intersect each other. All the contours that corresponds to internal boundaries shall belong to the bounded surface defined by the **api_contour** that corresponds to the external boundary and none of them shall belong to the bounded surface defined by another **api_contour**. That is, the **api_planar_surface** shall be arcwise connected. If these conditions hold, the **api_planar_surface** is computed by the interface by the following process.

1) The **plane** of the surface is computed by its **position**. The **position.location** shall be the first point of the first **composite_curve_segment** of the **api_contour** that corresponds to the external boundary. The x axis of the **position,** that is the **position.p[1]**, is the tangent to this **composite_curve_segment** in the sense defined by its **same_sense** attribute. The z axis of the **position**, that is the **position.p[3]** is computed as orthogonal to the plane that contains the **api_contour** that corresponds to the external boundary. Its sense is defined in such a way that this **api_contour** is oriented counter clockwise with respect to this oriented axis.

2) for each **api_contour** that defines the **api_planar_surface**, a **bounded_surface_curve** is instantiated, each one referring to this **api_contour** as its **curve_3d**. The **associated_geometry** attribute of this **surface_curve** contains only one element that is the **plane** of the **api_planar_surface** computed in step 1. The **master_representation** attribute of this **surface_curve** equals **curve_3d.**

3) For each computed **surface_curve**, a closed **composite_curve_segment** is instantiated. This **composite_curve_segment**:

   — refers to the **surface_curve** it corresponds to, as its parent_curve;

   — contains the **transition** value of the last **composite_curve_segment** of the **api_contour** that is the **curve_3d** of the **surface_curve** it corresponds to, as its **transition** attribute;

   v contains a **same_sense** attribute whose value equals **true** for the **composite_curve_segment** that correspond to the external boundary, and whose value is computed to ensure that all the other closed **composite_curve_segment**s are oriented clockwise with respect to the z axis of the **plane** of the **api_planar_surface** (i.e., the **position.p[3]**, see step 1).

4) an **outer_boundary_curve** is then instantiated whose **segments** contains only one element : the **composite_curve_segment** whose **parent_curve** refers, as its **curve_3d**, to the **api_contour** that corresponds to the external boundary of the **api_planar_surface.**

5) For each other **composite_curve_segment**, a **boundary_curve** is instantiated whose **segments** contains only this **composite_curve_segment**.

6) Finally, the **api_planar_surface** is instantiated. Its **basis_surface** is the **plane** of the **api_planar_surface**. Its **boundaries** are the (possible) **boundary_curves** and the **outer_boundary_curve** computed in step 5 and 4. Its **implicit_outer** attribute equals **false**.

EXPRESS specification:

```
*)
ENTITY api_planar_surface
 SUBTYPE OF (curve_bounded_surface);
WHERE
api_WR1: 'API_ABSTRACT_SCHEMA.PLANE' IN TYPEOF (SELF.basis_surface);
api_WR2: SIZEOF(QUERY( temp <* SELF.boundaries |
            'API_ABSTRACT_SCHEMA.OUTER_BOUNDARY_CURVE' IN
            TYPEOF (temp) )) =1;
api_WR3: QUERY (temp <* SELF.boundaries | SIZEOF(temp.segments) <> 1) = [];
api_WR4: SELF.implicit_outer = false;
END_ENTITY;
(*
```

NOTE 1 - This api specific entity is introduced to permit the specification of the range of some interface functions.

NOTE 2 - This entity may be implemented as its **curve_bounded_surface** supertype.

Attribute definitions:

SELF\curve_bounded_surface.basis_surface: The **plane** of the **api_planar_surface.**

SELF\curve_bounded_surface.boundaries: The **boundary_curve**s that corresponds to the different **api_contour**s that bound the **api_planar_surface.**

SELF\curve_bounded_surface.implicit_outer: A false value that specifies that the outer boundaries is explicitly defined in the set **SELF\curve_bounded_surface.boundaries.**

Formal definitions:

api_WR1: An **api_planar_surface** shall lies on a **plane.**

api_WR2: There exists exactly one **outer_boundary_curve** in **SELF.boundaries.**

api_WR3: Each boundary refers to exactly one **composite_curve_segment.**

Informal proposition:

api_IP1: If there are two or more **api_contour**s in the specification of the **api_planar_surface**, the distance between two **api_contour**s shall be not less than EPS.

6.1.18 API_ABSTRACT_SCHEMA entity definition : Geometric solid entities

This subclause declares the generic entity resources for **boolean_result**s and **solid**s defined in ISO 10303-42 that are part of the **api_abstract_schema.**

6.1.18.1 Solid_model

A **solid_model** is a complete representation of the nominal shape of a product such that all points in the interior are connected. Any point can be classified as being inside, outside or on the boundary of a solid.

There are several different types of solid model representations.

EXPRESS specification:

```
*)
ENTITY solid_model
 SUPERTYPE OF (ONEOF( csg_solid, swept_area_solid))
 SUBTYPE OF (geometric_representation_item);
END_ENTITY;
(*
```

NOTE - In the context of the **api_abstract_schema** only **csg_solid**s, and **swep_area_solid**s, may exist. Hence the SUPERTYPE is modified.

6.1.18.2 Csg_solid

A solid represented as a CSG model is defined by a collection of so-called primitive solids, combined using regularised Boolean operations. The allowed operations are intersection, union and difference. As a special case a CSG solid can also consist of a single CSG primitive.

A regularised subset of space is the closure of its interior, where this phrase is interpreted in the usual sense of point set topology. For **boolean_results** regularisation has the effect of removing dangling edges and other anomalies produced by the original operations.

A CSG solid requires two kinds of information for its complete definition: geometric and structural.

The geometric information is conveyed by **solid_models**. These typically are primitive volumes such as cylinders, wedges and extrusions. **Solid_model**s can also be **solid_replica**s (transformed solids) and **half_space_solid**s.

The structural information is in a tree (strictly, an acyclic directed graph) of **boolean_result**s and **csg_solid**s, that represent a 'recipe' for building the solid. The terminal nodes are the geometric primitives and other solids. Every **csg_solid** has precisely one **boolean_result** associated with it that is the root of the tree that defines the **solid**. (There may be further **boolean_result**s within the tree as operands). The significance of a **csg_solid** entity is that the solid defined by the associated tree is thus identified as a significant object in itself, and in this way it is distinguished from other **boolean_result** entities representing intermediate results during the construction process.

NOTE - In the context of the **api_abstract_schema** a specific interface function permits building a **csg_solid** from a **boolean_result**. A **csg_solid** shall be arcwise connected, a **boolean_result** may not be arcwise connected.

EXPRESS specification:

```
*)
```

```
ENTITY csg_solid
 SUBTYPE OF (solid_model);
 tree_root_expression : csg_select;
END_ENTITY;
(*
```

Attribute definitions:

tree_root_expression: Boolean expression of primitives and regularised operators describing the solid. The root of the tree of Boolean expressions is given here explicitly as a **boolean_result** entity, or as a **csg_primitive**.

6.1.18.3 Boolean_result

A **boolean_result** is the result of a regularised operation on two solids to create a new solid. Valid operations are regularised union, regularised intersection, and regularised difference. For purposes of Boolean operations, a solid is considered to be a regularised set of points.

The final **boolean_result** depends upon the operation and the two operands. In the case of the difference operator the order of the operands is also significant. The operator can be either union, intersection or difference. The effect of these operators is described below.

Union on two solids is the new solid that contains all the points that are in either the **first_operand** or the **second_operand** or both.

Intersection on two solids is the new solid that is the regularisation of the set of all points that are in both the **first_operand** and the **second_operand**.

The result of the **difference** operation on two solids is the regularisation of the set of all points that are in the **first_operand**, but not in the **second_operand**.

EXAMPLE - If the first operand is a block and the second operand is a solid cylinder of suitable dimensions and location the **boolean_result** produced with the difference operator would be a block with a circular hole.

EXPRESS specification:

```
*)
ENTITY boolean_result
 SUBTYPE OF (geometric_representation_item);
 operator    : boolean_operator;
 first_operand : boolean_operand;
 second_operand : boolean_operand;
END_ENTITY;
(*
```

Attribute definitions:

operator: The Boolean operator used in the operation to create the result

first_operand: The first operand to be operated upon by the Boolean operation

second_operand: The second operand specified for the operation

6.1.18.4 Csg_primitive

This subclause contains the definitions of all the CSG primitives. The CSG primitives are **sphere**, **right_circular_cone**, **right_circular_cylinder**, **torus**, **block** and **right_angular_wedge**. See also select type definition.

6.1.18.4.1 Sphere

A **sphere** is a CSG primitive with a spherical shape defined by a centre and a radius.

EXPRESS specification:

```
*)
ENTITY sphere
 SUBTYPE OF (geometric_representation_item);
 radius : positive_length_measure;
 centre : point;
END_ENTITY;
(*
```

Attribute definitions:

radius: The radius of the **sphere.**

centre: The location of the centre of the **sphere.**

6.1.18.4.2 Right_circular_cone

A **right_circular_cone** is a CSG primitive in the form of a cone that may be truncated. It is defined by an axis, a point on the axis, the semi-angle of the cone, and a distance giving the location in the location in the negative direction along the axis from the point to the base of the cone. In addition, a radius is given, that, if non zero, gives the size and location of a truncated face of the cone.

EXPRESS specification:

```
*)
ENTITY right_circular_cone
 SUBTYPE OF (geometric_representation_item);
 position   : axis1_placement;
 height    : positive_length_measure;
 radius    : length_measure;
 semi_angle : plane_angle_measure;
WHERE
 WR1: radius >= 0.0;
END_ENTITY;
(*
```

Attribute definitions:

position: The location of a point on the axis and the direction of the axis.

position.location: A point on the axis of the cone and on one of the planar circular faces, or, if radius is zero, at the apex.

position.axis: The direction of the central axis of symmetry of the cone.

height: The distance between the planar circular faces of the cone, if radius is greater than zero; or from the base to the apex, if radius equals zero.

radius: The radius of the cone at the point on the axis (**position.location**). If the **radius** is zero, the cone has an apex at this point. If the **radius** is greater than zero, the cone is truncated.

semi_angle: One half the angle of the cone. This is the angle between the axis and a generator of the conical surface.

Formal propositions:

WR1: The radius shall be non-negative.

Informal propositions:

IP1: The **semi_angle** shall be between 0° and 90°.

6.1.18.4.3 Right_circular_cylinder

A **right_circular_cylinder** is a CSG primitive in the form of a solid cylinder of finite height. It is defined by an axis point at the centre of one planar circular face, an axis, a height, and a radius. The faces are perpendicular to the axis and are circular discs with the specified radius. The height is the distance from the first circular face centre in the positive direction of the axis to the second circular face centre.

EXPRESS specification:

```
*)
ENTITY right_circular_cylinder
 SUBTYPE OF (geometric_representation_item);
 position : axis1_placement;
 height   : positive_length_measure;
 radius   : positive_length_measure;
END_ENTITY;
(*
```

Attribute definitions:

position: The location of a point on the axis and the direction of the axis.

position.location: A point on the axis of the cylinder and at the centre of one of the planar circular faces.

position.axis: The direction of the central axis of symmetry of the cylinder.

height: The distance between the planar circular faces of the cylinder

radius: The radius of the cylinder.

6.1.18.4.4 Torus

A torus is a solid primitive defined by sweeping the area of a circle (the generatrix) about a larger circle (the directrix). The directrix is defined by a location and direction (**axis1_placement**).

EXPRESS specification:

```
*)
ENTITY torus
 SUBTYPE OF (geometric_representation_item);
 position     : axis1_placement;
 major_radius : positive_length_measure;
 minor_radius : positive_length_measure;
WHERE
 WR1: major_radius > minor_radius;
END_ENTITY;
(*
```

Attribute definitions:

position: The location of the central point on the axis and the direction of the axis. This defines the centre and plane of the directrix.

major_radius: The radius of the directrix.

minor_radius: The radius of the directrix.

Formal propositions:

WR1: The **major_radius** shall be greater than the **minor_radius**.

6.1.18.4.5 Block

A block is a solid rectangular parallelepiped, defined with a location and placement coordinate system. The block is specified by the positive lengths x, y, and z along the axes of the placement coordinate system, and has one vertex at the origin of the placement coordinate system.

EXPRESS specification:

```
*)
ENTITY block
 SUBTYPE OF (geometric_representation_item);
 position : axis2_placement_3d;
 x    : positive_length_measure;
 y    : positive_length_measure;
 z    : positive_length_measure;
END_ENTITY;
(*
```

Attribute definitions:

position: The location and orientation of the axis system for the primitive. The block has one vertex at **position.location** and its edges aligned with the placement axes in the positive sense.

x: The size of the block along the placement X axis, (position.p[1]).

y: The size of the block along the placement Y axis, (position.p[2]).

z: The size of the block along the placement Z axis, (position.p[3]).

6.1.18.4.6 Right_angular_wedge

A **right_angular_wedge** can be envisioned as the result of intersecting a **block** with a **plane** perpendicular to one of its faces. It is defined with a location and local coordinate system. A triangular/trapezoidal face lies in the plane defined by the placement X and Y axes. This face is defined by positive lengths X and Y along the placement X and Y axes, by the length LTX (if non zero) parallel to the X axis at a distance Y from the placement origin, and by the line connecting the ends of the X and LTX segments. The remainder of the wedge is specified by the positive length Z along the placement Z axis that defines a distance through which the trapezoid or triangle is extruded. If LTX = 0, the wedge has five faces; otherwise, it has six faces. See **Figure 11** for interpretation of attributes.

EXPRESS specification:

```
*)
ENTITY right_angular_wedge
 SUBTYPE OF (geometric_representation_item);
 position : axis2_placement_3d;
 x    : positive_length_measure;
 y    : positive_length_measure;
 z    : positive_length_measure;
 ltx   : length_measure;
WHERE
 WR1: ((0.0 <= ltx) AND (ltx < x));
```

```
END_ENTITY;
(*
```

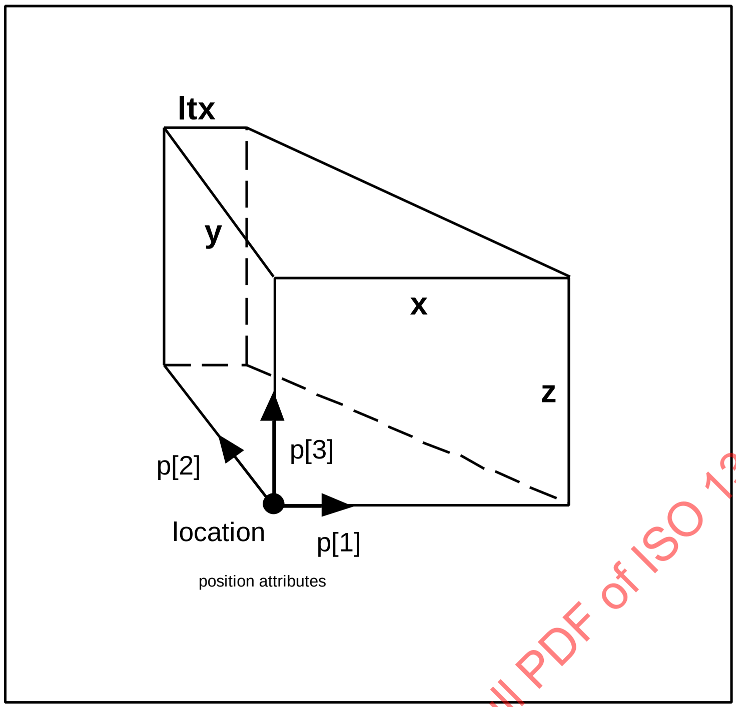


Figure 11 — Right_angular_wedge and its attributes

Attribute definitions:

position: The location and orientation of the placement axis system for the primitive. The wedge has one vertex at position.location and its edges aligned with the placement axes in the positive sense.

x: The size of the wedge along the placement X axis.

y: The size of the wedge along the placement Y axis.

z: The size of the wedge along the placement Z axis.

ltx: The length in the positive X direction of the smaller surface of the wedge.

Formal propositions:

WR1: **ltx** shall be non-negative and less than x.

6.1.18.5 Swept_area_solid

The **swept_area_solid** entity collects the entities that are defined procedurally by a sweeping action on planar bounded surfaces. The position in space of the swept solid will be dependent upon the position of the **swept_area**. The **swept_area** will be a face of the resulting **swept_area_solid**, except for the case of a **revolved_area_solid** with angle equal to 360 degrees.

EXPRESS specification:

```
*)
ENTITY swept_area_solid
 SUPERTYPE OF (ONEOF(revolved_area_solid, extruded_area_solid))
 SUBTYPE OF (solid_model);
 swept_area : curve_bounded_surface;
WHERE
 api_WR1: 'API_ABSTRACT_SCHEMA.PLANE' IN
      TYPEOF (swept_area.basis_surface);
END_ENTITY;
(*
```

Attribute definitions:

swept_area: The **curve_bounded_surface** defining the area to be swept. The extent of this area is defined by the **boundaries** attribute of the referenced **curve_bounded_surface**.

Formal propositions:

api_WR1: The **swept_area** shall be planar. The **basis_surface** attribute of the **curve_bounded_surface** referenced shall be a **plane**.

6.1.18.6 Extruded_area_solid

An **extruded_area_solid** is a solid defined by sweeping a bounded planar surface. The direction of translation is defined by a **direction** vector, and the length of the translation is defined by a distance **depth**. The planar area may have holes that will sweep into holes in the solid.

EXPRESS specification:

```
*)
ENTITY extruded_area_solid
 SUBTYPE OF (swept_area_solid);
 extruded_direction : direction;
 depth       : positive_length_measure;
WHERE
 WR1: dot_product(
    (SELF\swept_area_solid.swept_area.basis_surface\
    elementary_surface.position.p[3]), extruded_direction) <> 0.0;

END_ENTITY;
(*
```

Attribute definitions:

SELF\swept_area_solid.swept_area: The bounded surface to be extruded to produce the solid.

extruded_direction: The **direction** in which the area is to be swept.

depth: The distance the area is to be swept.

Formal propositions:

WR1: **extruded_direction** shall not be perpendicular to the normal to the plane of the **swept_area**.

6.1.18.7 Revolved_area_solid

A **revolved_area_solid** is a solid formed by revolving a planar bounded surface about an axis. The axis shall be in the plane of the surface and the axis shall not intersect the interior of the bounded surface. The bounded surface may have holes that will sweep into holes in the solid. The direction of revolution is clockwise when viewed along the axis in the positive direction. More precisely if **A** is the axis location and **d** is the axis direction and **C** is an arc on the surface of revolution generated by an arbitrary point **p** on the boundary of the face, then C leaves p in direction d x (p-A) as the area is revolved.

EXPRESS specification:

```
*)
ENTITY revolved_area_solid
 SUBTYPE OF (swept_area_solid);
 axis   : axis1_placement;
 angle   : plane_angle_measure;
DERIVE
 axis_line : line := line(axis.location, axis.z);
END_ENTITY;
(*
```

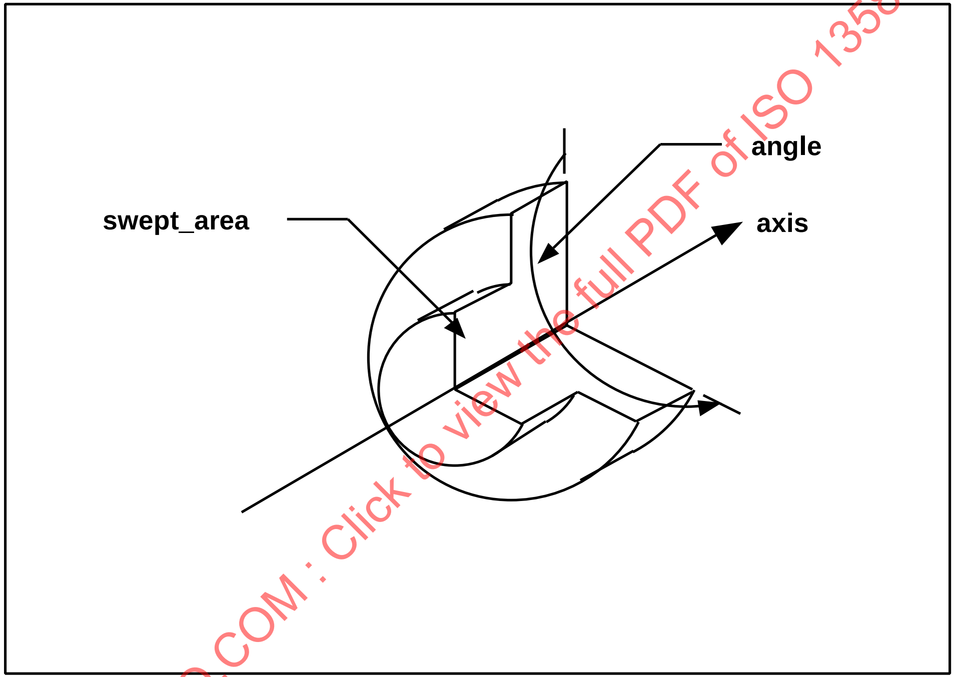


Figure 12 — Revolved area solid

Attribute definitions:

SELF\swept_area_solid.swept_area: The **curve_bounded_surface** to be revolved to produce the solid.

axis: Axis about which revolution will take place.

angle: Angle through which the sweep will be made. This angle is measured from the plane of the swept area.

axis_line: The line of the axis of revolution.

Informal propositions:

IP1: **axis_line** shall lie in plane of **swept_face** attribute of the **swept_face_solid** supertype.

IP2: The **axis_line** shall not intersect the interior of the **swept_face**

IP3: **angle** shall be between 0° and 360°.

6.1.18.8 Half_space_solid

An **half_space_solid** is defined by the half space that is the regular subset of the domain that lies on one side of an unbounded surface. The side of the surface that is in the half space is determined by the surface normals and the agreement flag. If the agreement flag is TRUE, then the subset is the one the normals point away from. If the agreement flag is FALSE, then the subset is the one the normals point into.

For a valid **half_space_solid**, the surface shall divide the domain into exactly two subsets. Also, within the domain the surface shall be manifold and all the surface normals shall point into the same subset.

NOTE 1 - An **half_space_solid** is not a sub-type of **solid_model**; **half_space_solid**s are only useful as operands in **boolean_expression**s.

NOTE 2 - In the context of the **api_abstract_schema**, only **half_space_solid** whose **base_surface** is a **plane** is allowed.

<u>EXPRESS specification:</u>

```
*)
ENTITY half_space_solid
 SUBTYPE OF(geometric_representation_item);
 base_surface  : surface;
 agreement_flag : BOOLEAN;
WHERE
 api_WR1: 'API_ABSTRACT_SCHEMA.PLANE' IN TYPEOF (base_surface);
END_ENTITY;
(*
```

<u>Attribute definitions:</u>

base_surface: **surface** defining side of half space

agreement_flag: The **agreement_flag** is TRUE if the normal to the **base_surface** points away form the material of the **half_space_solid**

<u>Formal proposition:</u>

api_WR1: The surface that defines the **half_space_solid** shall be a **plane**

6.1.19 API_ABSTRACT_SCHEMA entity definition : api specific entities for structuring

This subclause declares the api-specific resources for structuring the geometric representation items that may be created by the interface functions.

The specification and the behaviour of structured entities and the structuring of geometrical data into **set**s, are described in **5.4** (Entities structure) of this International Standard.

6.1.19.1 Api_group

An **api_group** is a **group** that is to be created in the TDB.

<u>EXPRESS specification:</u>

```
*)
ENTITY api_group
 SUBTYPE OF(group);
WHERE
 api_WR1: ( (SELF\group.name = 'TDB') AND (USEDIN(SELF,'') = []) )
      OR
      ( (LENGTH(SELF\group.name) = 0 ) AND
       (SIZEOF(USEDIN(SELF,'API_ABSTRACT_SCHEMA'+
                 'API_GROUP_ASSIGNMENT.ITEMS')
          ) = 1)
      );
 api_WR2: tree_api_group_structure(SELF);
END_ENTITY;
(*
```

Formal propositions:

api_WR1: Either the **name** of an **api_group** is 'TDB' and it shall not be referenced by any other entity, or its name is the empty string and it belongs to exactly one **api_group.**

api_WR2: api_groups are tree structured.

6.1.19.2 Api_group_assignment

An **api_group_assignment** is an assignment of one or more **point**s, **curve**s, **surfaces**, **vector**s, **direction**s, **placement**s, **fill_area**s, **solid**s or **group**s to a **group.**

EXPRESS specification:

```
*)
ENTITY api_group_assignment
 SUBTYPE OF(group_assignment);
 items : SET [0:?] OF api_grouped_item;
WHERE
 api_WR1: 'API_ABSTRACT_SCHEMA.API_GROUP' IN
              TYPEOF (SELF\ group_assignment.assigned_group);
END_ENTITY;
(*
```

Attribute definitions:

items: The **api_grouped_item**s that are assigned to the **api_group**

Formal propositions:

api_WR1: An **api_group_assignment** shall assign an **api_group .**

6.1.19.3 Api_set

An **api_set** is a **group** that is to be created in the CAD system database.

EXPRESS specification:

```
*)
ENTITY api_set
 SUBTYPE OF (group);
WHERE
 api_WR1: ( (SELF\group.name = 'VIEW') AND (USEDIN(SELF,'') = []) )
      OR
      ( (SELF\group.name <> 'VIEW') AND
       (SIZEOF(USEDIN(SELF,'API_ABSTRACT_SCHEMA'+
                 'API_GROUP_ASSIGNMENT.ITEMS')
          ) = 1)
      );
 api_WR2: tree_api_set_structure (SELF);
END_ENTITY;
(*
```

Formal propositions:

api_WR1: Either the **name** of an **api_set** is 'VIEW' and it shall not be referenced by any other entity, or its name is different from 'VIEW' and it belongs to exactly one **api_set.**

api_WR2: api_sets are tree-structured

6.1.19.4 Api_set_assignment

An **api_set_assignment** represents the assignment of one or more **direction**s, **vector**s, **placements**, **point**s, **curve**s, **surface**s, **fill_area**s, **half_space_solid**s, **boolean_result**s, **solid**s or **api_set**s to an **api_set.**

EXPRESS specification:

```
*)
ENTITY api_set_assignment
 SUBTYPE OF(group_assignment);
 items : SET [0:?] OF api_set_item;
WHERE
 api_WR1: 'API_ABSTRACT_SCHEMA.API_SET' IN
      TYPEOF (SELF\group_assignment.assigned_group);
END_ENTITY;
(*
```

Attribute definitions:

items: The **api_set_item**s that are assigned to the **api_set**

Formal propositions:

api_WR1: An **api_set_assignment** shall assign an **api_set**.

6.2 Visual appearance of geometric representation items

All the **representation_item**s that are explicitly created through the interface, both in the TDB and in the CAD system database, are assigned a presentation style. The mathematical entities are assigned a **null_style**. The presentation of these **representation_item**s is implementation dependent. When the creation of a **representation_item** requires the implicit creation of another **representation_item**, e.g., when a **curve** is implicitly created as the **basis_curve** of a **trimmed_curve**, this implicitly created entity is associated with a **null_style**.

The style assignment is made by the interface at the creation time of each **representation_item**. The style assignment is made by instanciating for each **representation_item** a **styled_item** that refers to this **representation_item** together with its **presentation_style_assignment**. A **presentation_style_assignment** itself is a collection of different presentation styles such as point style, curve style or text style. Styling an unstyled **representation_item** produces a new **representation_item** that has presentation style assigned. The **presentation_style_assignment** of a **styled_item** affects the appearance of the referenced **representation_item** as well as the appearance of all **representation_items** referenced directly or indirectly by that item. Only those **representation_items** are affected, that are not already styled. This means, styling a styled **representation_item** has no effect; styling a partially styled **representation_item** affect only the appearance of the unstyled parts, and styling an **representation_item** affects the appearance of the whole item. Only styled **representation_items** may be presented. Whether they are actually presented depends on hidden line removal, as well as the CAD system viewing pipeline (see **5.3.5**.).

In the context of the **api_abstract_schema** a **styled_item** shall refer to only one **presentation_style_assignment**, and, in the CAD system, a **presentation_style_assignment** shall consist, for all but the **annotation_fill_area** entity, of only one **presentation_style** that is the current value of the **presentation_style** that corresponds to the created **representation_item** in the interface status table. The **presentation_style_assignment** of an **annotation_fill_area** always contains one **presentation_style_select**s that specifies whether or not the fill area shall be filled with the background colour. This style is defined by an **api_externally_defined_fill_area_style**. The **presentation_style_assignment** of an **annotation_fill_area** may also contains any number of **fill_area_style**, each one referring to a **fill_area_style_hatching**.

When the current view is two-dimensional and when the hidden line removal process is activated, the **presentation_style_assignement**s that corresponds to **point**s **curve**s and **fill_area**s may also contain two other api-specific styles. The **api_pre_defined_occlusion_style** specifies that an entity shall be involved in the hidden line removal process, together with its virtual height in the virtual 3D space. The **api_pre_defined_virtual_sent_style** specifies that the corresponding entity is preserve in the TDB only for hidden line removal purpose.

The entity name returned by an interface function that creates a **representation_item** is the name of the **styled_item** that refers to it. If the entity remains in the TDB, its presentation style may be changed afterward by a change function *Chg_*... (see annex **A**, **A.10.3**).

In order that the part supplier is only allowed logical control on the visual appearance of the **representation_item**s created through the interface function, all the styles, but the **fill_area_style_hatching** are defined as **externally_defined_style**. These **externally_defined_style**s shall be defined either in this part of ISO 13584, or in any part of the view exchange protocol series of parts. The part where an **externally_defined_style** is defined constitutes the external source of this **externally_defined_item**. If some interface implementation implements a view exchange protocol that is referred to from a part supplier program to specify the **externally_defined_style** to be used for some **representation_item,** then the first style defined for this **representation_item** in this part of ISO 13584 shall be used in place of the unknown style and no error shall be reported.

6.2.1 API_ABSTRACT_SCHEMA type definition : Visual presentation

This subclause declares the generic type resources defined in ISO 10303-46 that are part of the **api_abstract_schema**.

6.2.1.1 Presentation_style_select

The **presentation_style_select** is used by a **presentation_style_assignment** to associate style with a **representation_item**. A different style is provided for each kind of **representation_item** to be styled.

EXPRESS specification:

```
*)
TYPE presentation_style_select = SELECT
 (pre_defined_presentation_style,
  point_style,
  curve_style,
  surface_style_usage,
  symbol_style,
  fill_area_style,
  text_style,
  approximation_tolerance,
  externally_defined_style,
  null_style);
END_TYPE;
(*
```

6.2.1.2 Null_style

The **null_style** specifies that no specific style is assigned directly to an item that is to be presented. The style or styles to be used in presenting the item are specified within the definition of the item. If no styles are specified within the definition, then the item shall not be presented..

NOTE 1 - In the context of the **api_abstract_schema**, a **null_style** is assigned to all the **representation_items** that are implicitly created through the interface to permit representation of the explicitly created entities.

EXAMPLE 1 - When creating a **trimmed_curve** (e.g., **api_circular_arc**) its **basis_curve** (e.g., **circle**) is implicitly created.

EXPRESS specification:

```
*)
TYPE null_style = ENUMERATION OF
 (null);
END_TYPE;
(*
```

Enumerated_item_definitions:

null: The **representation_item** to which the style is applied shall be presented using the style or styles contained in its definition, if any.

NOTE 2 - In the context of the **api_abstract_schema**, a **null_style** is assigned to all the **representation_items** that are implicitly created through the interface to permit representation of the explicitly created entities.

EXAMPLE 2 - When creating a **trimmed_curve** (e.g. **api_circular_arc**) its **basis_curve** (e.g., **circle**) is implicitly created.

6.2.1.3 Size_select

The **size_select** is used to specify the size of marker symbols or the width of curves.

EXPRESS specification:

```
*)
TYPE size_select = SELECT
 (positive_length_measure,
  measure_with_unit,
  descriptive_measure,
  pre_defined_size);
END_TYPE;
(*
```

6.2.1.4 Curve_font_or_scaled_curve_font_select

The **curve_font_or_scaled_curve_font_select** is a selection of a **curve_style_font_select** or a **curve_style_font_and_scaling**. It is used to specify the font for presenting a curve.

EXPRESS specification:

```
*)
TYPE curve_font_or_scaled_curve_font_select = SELECT
 (curve_style_font_select,
  curve_style_font_and_scaling);
END_TYPE;
(*
```

6.2.1.5 Curve_style_font_select

The **curve_style_font_select** is a selection of a **curve_style_font**, a **pre_defined_curve_font,** or an **externally_defined_curve_font**. It is used to specify an unscaled font for presenting a curve.

EXPRESS specification:

```
*)
TYPE curve_style_font_select = SELECT
 (curve_style_font,
  pre_defined_curve_font,
  externally_defined_curve_font);
END_TYPE;
(*
```

6.2.1.6 Fill_style_select

The **fill_style_select** is a selection between different fill styles.

EXPRESS specification:

```
*)
TYPE fill_style_select = SELECT
 (fill_area_style_colour,
  pre_defined_tile_style,
  externally_defined_tile_style,
  fill_area_style_tiles,
  pre_defined_hatch_style,
  externally_defined_hatch_style,
  fill_area_style_hatching);
END_TYPE;
(*
```

6.2.2 API_ABSTRACT_SCHEMA type definition : api specific types for visual presentation

This subclause declares the api-specific type defined in the **api_abstract_schema** for visual presentation.

6.2.2.1 Virtual_height_ratio

A **virtual_height_ratio** is a real value that defines the virtual height of a **geometric_representation_item** that is geometrically founded in a two dimensional **geometric_representation_context**. This **virtual_height_ratio** is used for occlusion precedence and hidden line removal.

EXPRESS specification:

```
*)
TYPE virtual_height_ratio = REAL;
END_TYPE;
(*
```

6.2.3 API_ABSTRACT_SCHEMA entities definition : Visual presentation

This subclause declares the generic entities resources defined in ISO 10303-46 that are part of the **api_abstract_schema**.

6.2.3.1 Styled_item

A **styled_item** is a **representation_item** with associated presentation style.

EXPRESS specification:

```
*)
ENTITY styled_item
 SUBTYPE OF (representation_item);
 styles : SET [1:?] OF presentation_style_assignment;
 item   : representation_item;
WHERE
 WR1  : (SIZEOF(SELF.styles) = 1)
          XOR
     (SIZEOF(QUERY( pres_style <* SELF.styles |
      NOT ('PRESENTATION_APPEARANCE_SCHEMA.' +
        'PRESENTATION_STYLE_BY_CONTEXT' IN
      TYPEOF(pres_style))
      )) = 0);
 api_WR2: api_legal_style_number (SELF);
END_ENTITY;
(*
```

Attribute definitions:

styles: The styles assigned to the item.

item: The item to which styles are assigned.

Formal propositions:

WR1: The set **styles** shall contain only one style or all members of the set shall be **presentation_-style_by_context** entities.

NOTE - This is to ensure that there are no style conflicts; more than one style may be specified only when the context in which each style applies is given.

api_WR2: The **api_legal_style_number** function checks the number of styles indirectly assigned to a representation item.

6.2.3.2 Presentation_style_assignment

A **presentation_style_assignment** is a set of styles that are assigned to a **representation_item** for the purpose of presenting the item.

EXPRESS specification:

```
*)
ENTITY presentation_style_assignment;
 styles : SET [1:?] OF presentation_style_select;
WHERE
 WR1: SIZEOF(QUERY(style1 <* SELF.styles |
     NOT (SIZEOF(QUERY (style2 <* (SELF.styles - style1) |
      NOT ((TYPEOF (style1) <> TYPEOF(style2)) OR
      (SIZEOF(['PRESENTATION_APPEARANCE_SCHEMA.'+
           'SURFACE_STYLE_USAGE',
           'API_ABSTRACT_SCHEMA.'+
           'EXTERNALLY_DEFINED_STYLE'] *
                   TYPEOF(style1)) = 1)))) = 0 ))) = 0;
 WR2: SIZEOF(QUERY (style1 <* SELF.styles |
     'PRESENTATION_APPEARANCE_SCHEMA.SURFACE_STYLE_USAGE' IN
                          TYPEOF(style1))) <= 2;
END_ENTITY;
(*
```

Attribute definitions:

styles: The set of presentation styles that are assigned to a **representation_item**.

Formal propositions:

WR1: The same style shall not appear more than once in the set of styles, except for **externally_defined_style** and **surface_style_usage**.

WR1: **surface_style_usage** shall not occur more than twice in the set of styles.

Informal propositions:

IP1: Externally defined styles shall not conflict with other styles in the same **presentation_style_assignment** entity, including other externally defined styles.

NOTE - For one style to conflict with the other, it specifies a different style for the same characteristic, such as colour or width. For example, one style might say blue, and the other green, and both be applied to the same entity.

IP2: Each style type is unique.

EXAMPLE - If a **line** is given a style that is a curve style, it shall appear. If a **line** is given both curve and point style, both the **curve** and its related **cartesian_point**s would appear.

IP3: If there are two instances of **surface_style_usage** in the set of styles, each shall specify the style for opposite sides of the surface being styled.

6.2.3.3 Externally_defined_style

An **externally_defined_style** is an external reference to a presentation style.

NOTE - In the context of the **api_abstract_schema** the **external_source** shall only be one part of ISO 13584.

EXPRESS specification:

```
*)
ENTITY externally_defined_style
 SUBTYPE OF (externally_defined_item);
WHERE
api_WR1 : (SELF\externally_defined_item.source.source_id LIKE
                             'ISO_13584_31')
     OR
     (SELF\externally_defined_item.source.source_id LIKE
                        'ISO_13584'+'-1'+'##');
END_ENTITY;
(*
```

Attribute definitions:

SELF\externally_defined_item.source: The name of the part of ISO 13854 where the style is defined.

SELF\externally_defined_item.item_id: The **identifier** of the style that shall be used.

Formal propositions:

api_WR1: The **source** of the **externally_defined_style** shall be either this part of ISO 13584 or one part of the view exchange protocol series of parts.

6.2.3.4 Curve_style

A **curve_style** specifies the visual appearance of a curve.

NOTE - In the context of the **api_abstract_schema**, a **curve_style** is only used for curve style of hatch lines. This **curve_style** uses **pre_defined_curve_font**, **pre_defined_size** and **colour** (the value **curve_colour** is (impl. dep.) deferred to the application)

EXPRESS specification:

```
*)
ENTITY curve_style;
 name     : label;
 curve_font  : curve_font_or_scaled_curve_font_select;
 curve_width : size_select;
 curve_colour : colour;
WHERE
 api_WR1 : USEDIN(SELF,'API_ABSTRACT_SCHEMA.'+
            'FILL_AREA_STYLE_HATCHING.'+
            'HATCH_LINE_APPEARANCE') <> [];
END_ENTITY;
(*
```

Attribute definitions:

name: The word, or group of words, by which the **curve_style** is referred to.

curve_font: The **curve_style_font**, scaled **curve_style_font**, **pre_defined_curve_font**, scaled **pre_-defined_curve_font**, **externally_defined_curve_font** or scaled **externally_defined_curve_font** that is used to present a curve.

curve_width: The width of the visible part of the presented curve in **presentation_area** units.

curve_colour: The **colour** of the visible part of the curve.

Formal propositions:

api_WR1: The **curve_style** shall be used to define the **hatch_line_appearance** of a **fill_area_style_hatching**.

6.2.3.5 Fill_area_style

A style for filling visible curve segments, annotation fill areas or surfaces with tiles or hatching.

NOTE - In the context of the **api_abstract_schema** an explicit **fill_area_style** is only used for hatching a **fill_area**.

EXPRESS specification:

```
*)
ENTITY fill_area_style;
 name    : label;
 fill_styles : SET [1:?] OF fill_style_select;
WHERE
 WR1  : SIZEOF(QUERY(fill_style <* SELF.fill_styles |
         'PRESENTATION_APPEARANCE_SCHEMA.'+
         'FILL_AREA_STYLE_COLOUR' IN
         TYPEOF(fill_style)
         )) <= 1;
 api_WR2: QUERY(fill_style <* SELF.fill_styles |
       NOT ( 'API_ABSTRACT_SCHEMA.FILL_AREA_STYLE_HATCHING' IN
          TYPEOF(fill_style)) ) = [];
END_ENTITY;
(*
```

Attribute definitions:

name: The word, or group of words, by which the **fill_area_style** is referred to.

fill_styles: The set of fill area styles to use in presenting visible curve segments, annotation fill areas, or surfaces.

Formal propositions:

WR1: There shall be not more than one **fill_area_style_colour** in the **fill_styles** set.

api_WR2: All the **fill_style_select**s shall be a **fill_area_style_hatching**

6.2.3.6 Fill_area_style_hatching

A **fill_area_style_hatching** defines a styled pattern of curves for hatching visible curve segments, annotation fill areas or surfaces.

NOTE 1 - In the context of the **api_abstract_schema fill_area_style_hatching** is only used for styling a **fill_area**.

NOTE 2 - In the context of the **api_abstract_schema hatch_line_appearance** shall be defined through **pre_defined_item**s

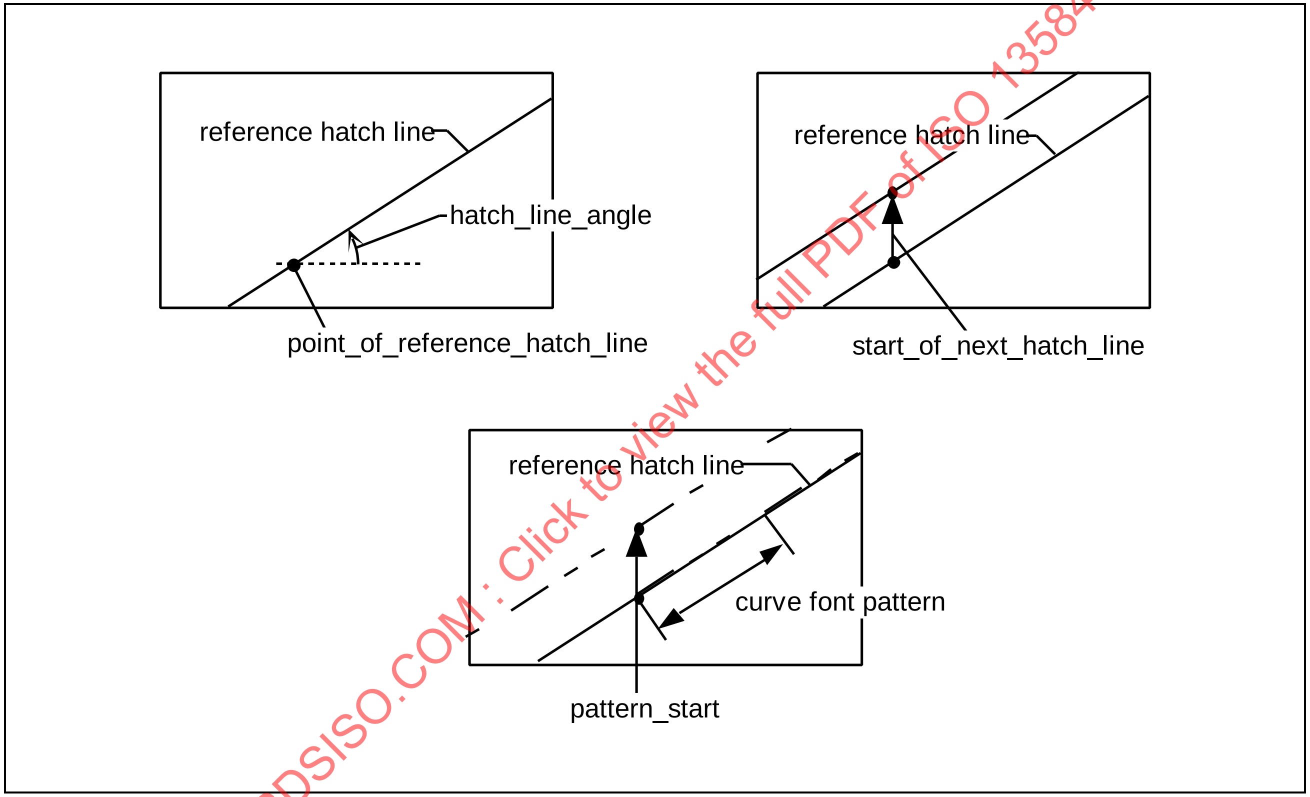


Figure 13 — Fill area style hatching

EXPRESS specification:

```
*)
ENTITY fill_area_style_hatching
 SUBTYPE OF (geometric_representation_item);
 hatch_line_appearance      : curve_style;
 start_of_next_hatch_line    : one_direction_repeat_factor;
 point_of_reference_hatch_line : cartesian_point;
 pattern_start          : cartesian_point;
 hatch_line_angle        : plane_angle_measure;
WHERE
 api_WR1: 'API_ABSTRACT_SCHEMA.'+
      'API_PRE_DEFINED_HATCH_CURVE_FONT' IN
              TYPEOF (SELF.HATCH_LINE_APPEARANCE.CURVE_FONT);
 api_WR2: 'API_ABSTRACT_SCHEMA.'+
      'API_PRE_DEFINED_HATCH_WIDTH' IN
              TYPEOF (SELF.HATCH_LINE_APPEARANCE.CURVE_WIDTH);
 api_WR3: 'API_ABSTRACT_SCHEMA.'+
      'API_PRE_DEFINED_HATCH_COLOUR' IN
             TYPEOF (SELF.HATCH_LINE_APPEARANCE.CURVE_COLOUR);
END_ENTITY;
(*
```

Attribute definitions:

hatch_line_appearance: The **curve_style** of the hatch lines. Any **curve_style** pattern shall start at the origin of each hatch line. The origin of the reference hatch line is specified by **pattern_start**. The origin of any other hatch line is determined by adding a multiple of **start_of_next_hatch_line** to **pattern_start**.

start_of_next_hatch_line: The displacement between adjacent hatch lines, specified as a vector.

point_of_reference_hatch_line: The origin for mapping the **fill_area_style_hatching** onto a curve, annotation fill area, or surface.

pattern_start: The start point for the **curve_style** of the **reference_hatch_line**.

hatch_line_angle: The angle determining the direction of the parallel hatching lines.

NOTE 3 - **Figure 13** shows the definition of **fill_area_style_hatching**.

Formal propositions:

api_WR1: The **curve_font** of the **hatch_line_appearance** shall be defined as an **api_pre_defined_-hatch_curve_font.**

api_WR2: The **curve_width** of the **hatch_line_appearance** shall be defined as an **api_pre_defined_hatch_curve_width.**

api_WR3: The **curve_colour** of the **hatch_line_appearance** shall be defined as an **api_pre_defined_hatch_colour.**

6.2.3.7 One_direction_repeat_factor

A **one_direction_repeat_factor** is a vector used in a **fill_area_style_hatching** for determining the origin of a repeated hatch line relative to the origin of the previous hatch line. Given the initial position I of any hatch line, the **one_direction_repeat_factor** R determines two new positions according to the expression:

$$I + k \cdot R \qquad k = -1, 1$$

NOTE - **Figure 14** shows the positions defined by an **one_direction_repeat_factor.**

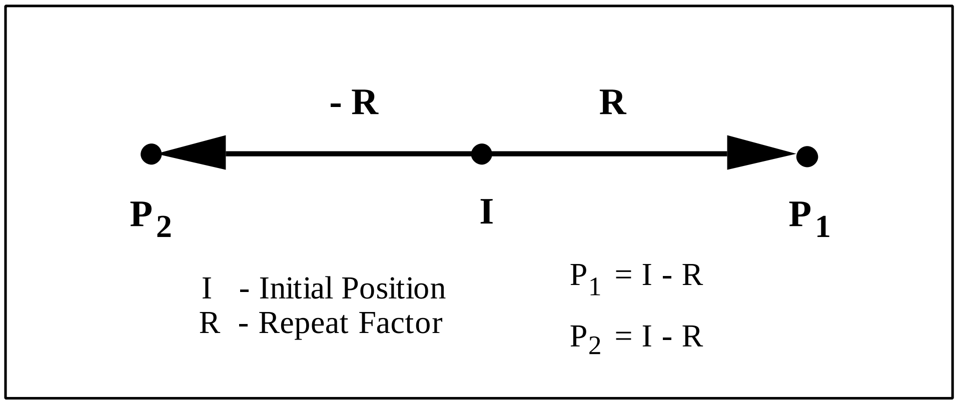


Figure 14 — One direction repeat factor

EXPRESS specification:

```
*)
ENTITY one_direction_repeat_factor
 SUBTYPE OF (geometric_representation_item);
 repeat_factor : vector;
END_ENTITY;
(*
```

Attribute definitions:

repeat_factor: The **vector** that specifies the relative positioning of hatch lines.

6.2.3.8 Colour

A **colour** defines a basic appearance property of an element with respect to the light reflected by it.

EXPRESS specification:

```
*)
ENTITY colour;
END_ENTITY;
(*
```

6.2.3.9 Pre_defined_size

A **pre_defined_size** may be used to define an application-specific size for markers.

NOTE 1 - Application Resources or Application Protocols specify the use of this entity.

NOTE 2 - In the context of the **api_abstract_schema**, a **pre_defined_size** is used to define width of hatch lines.

EXPRESS specification:

```
*)
ENTITY pre_defined_size
 SUBTYPE OF (pre_defined_item);
END_ENTITY;
(*
```

6.2.3.10 Pre_defined_curve_font

A **pre_defined_curve_font** may be used to define application-specific **curve_font**s.

NOTE 1 - Application Resources or Application Protocols specify the use of this entity.

NOTE 2 - In the context of the **api_abstract_schema**, a **pre_defined_curve_font** is used to define the font of hatch lines.

EXPRESS specification:

```
*)
ENTITY pre_defined_curve_font
 SUBTYPE OF (pre_defined_item);
END_ENTITY;
(*
```

6.2.3.11 Pre_defined_colour

A **pre_defined_colour** is provided to allow an application-specific colour definition.

NOTE 1 - Application Resources or Application Protocols specify the use of this entity. The **pre_defined_colour** entity further enables application resources or application protocols to fix colour values or components of colour values for their particular uses.

NOTE 2 - In the context of the **api_abstract_schema**, a **pre_defined_colour** is used to define colour of hatch lines.

EXPRESS specification:

```
*)
ENTITY pre_defined_colour
 SUBTYPE OF (pre_defined_item, colour);
END_ENTITY;
(*
```

6.2.3.12 Annotation_occurrence

An **annotation_occurrence** defines occurrences of annotation by combining two-dimensional geometry or annotation elements with style for presentation purposes, i.e., **area_dependent_annotation_representation**, **view_dependent_annotation_representation**, **curve_style_curve_pattern**, **fill_area_style_tile_curve_with_style**, or **fill_area_style_tile_coloured_region**. See ISO 10303-46 for more information about these entities.

NOTE 1 - In the context of **api_abstract_schema** the usage of an **annotation_occurrence** is defined only for presentation propose of an **annotation_fill_area** assigned with a **fill_area_style_hatching**.

NOTE 2 - In the context of the **api_abstract_schema** only an **annotation_fill_area_occurrence** shall be inherited by the interface. Hence the SUPERTYPE is prunded.

EXPRESS specification:

```
*)
ENTITY annotation_occurrence
 SUPERTYPE OF (annotation_fill_area_occurrence)
 SUBTYPE OF (styled_item);
WHERE
 WR1: 'API_ABSTRACT_SCHEMA.GEOMETRIC_REPRESENTATION_ITEM' IN
                              TYPEOF (SELF);
END_ENTITY;
(*
```

Formal propositions:

WR1: An **annotation_occurrence** shall be a **geometric_representation_item**.

6.2.3.13 Annotation_fill_area_occurrence

An **annotation_fill_area_occurrence** is the assignment of a style to an **annotation_fill_area**; it includes the specification of the point to be used as the starting point of the **fill_area**.

NOTE 1 - In the context of **api_abstract_schema**, an **annotation_fill_area_occurrence** is used to assign a **fill_area_style_hatching** to an **annotation_fill_area**, using a transformation with **fill_area_style_-hatching.point_of_reference_hatch_line** as origin and **fill_style_target** as target point for the assignment.

NOTE 2 - In the context of **api_abstract_schema**, where **annotation_fill_area** and **fill_area_style_hatching** may only be defined in two-dimensional **geometric_representation_context**, the x axis of **geometric_representation_context** of the **fill_area_style_hatching** shall be implicitly mapped on the x axis of the **geometric_representation_context** to which the point **fill_style_target** belongs.

EXPRESS specification:

```
*)
ENTITY annotation_fill_area_occurrence
 SUBTYPE OF (annotation_occurrence);
 fill_style_target : point;
WHERE
 WR1   : 'API_ABSTRACT_SCHEMA.ANNOTATION_FILL_AREA' IN
                            TYPEOF (SELF.item);
 api_WR2 : SIZEOF(QUERY(psa <*
             SELF\annotation_occurrence\styled_item.styles |
       SIZEOF(QUERY(pss <* psa.styles |
        (NOT ('API_ABSTRACT_SCHEMA.FILL_AREA_STYLE' IN
                               TYPEOF(pss)))
       AND
       (SIZEOF(QUERY(fss <* pss.fill_styles |
        (NOT ('API_ABSTRACT_SCHEMA.FILL_AREA_STYLE_HATCHING)
        ))) =0))) = 0) = 0)) = 0;
 api_WR3 : SIZEOF(QUERY(psa <*
             SELF\annotation_occurrence\styled_item.styles |
       SIZEOF(QUERY(pss <* psa.styles |
       NOT pss.point_of_reference_hatch_line =
                    SELF.fill_style_target))= 0)) = 0;
END_ENTITY;
(*
```

Attribute definitions:

fill_style_target: The point that specifies the starting location for the **fill_area_style** assigned to the **annotation_fill_area**.

Formal propositions:

WR1: The styled item shall be an **annotation_fill_area**.

api_WR2: The the **fill_styles** in the set of **fill_style_select**, for filling an **annotation_fill_area**, shall be only of type **fill_area_style_hatching**.

api_WR3: The starting location point **fill_style_target** shall be refer to **point_of_reference_hatch_line** of **fill_area_style_hatching** in the set of **fill_styles** assigned to an **annotation_fill_area** representation item.

6.2.4 API_ABSTRACT_SCHEMA entities definition : externally-defined styles for visual presentation

This subclause declares the externally defined styles that are specified in this part of ISO 13584, and may be referenced to by the application programs to define the (logical) visual appearance of the geometric or annotation entities.

These styles are only defined logically in order to permit the interface end user to customise its interface.

They are defined as **externally_defined_style** to permit the extension of the set of available styles in the different view exchange protocol series of parts of ISO 13584.

All the styles defined in this part of ISO 13584 shall be implemented. If one **externally_defined_style** defined in some view exchange protocol is not implemented on some interface implementation, then the first style defined for this kind of item in this part of ISO 13584 shall be used and no error shall be reported.

6.2.4.1 Api_externally_defined_point_style

An **api_externally_defined_point_style** specifies the visual appearance of **point**s.

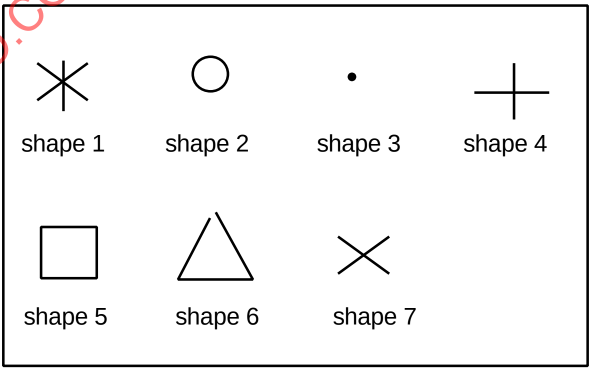
The following point styles are defined in this part of ISO 13584. The size and colour of the styles point are implementation dependent (impl. dep.). Default values shall be provided by the interface implementor. These default values shall be customisable by the interface end user.

NOTE 1 - Other **api_externally_defined_point_style**s may be defined by the view exchange protocol series of parts of ISO 13584. If one interface implementation does not support this view exchange protocol, the fist style defined by this part of ISO13584 shall be used, and no error shall be reported.

Table 7 — Externally defined point styles

item_id	shape	colour, size
'asterisk_point'	Table , shape 1	impl. dep.
'circle_point'	Table , shape 2	impl. dep.
'dot_point'	Table , shape 3	impl. dep.
'plus_point'	Table , shape 4	impl. dep.
'square_point'	Table , shape 5	impl. dep.
'triangle_point'	Table , shape 6	impl. dep.
'x_point'	Table , shape 7	impl. dep.
'virtual_point'	undefined	undefined

Table 8 — Shapes of the externally defined point styles



<u>EXPRESS specification:</u>

```
*)
ENTITY api_externally_defined_point_style
 SUBTYPE OF (externally_defined_style);
END_ENTITY;
(*
```

NOTE 2 - This entity may be implemented as an **externally_defined_style**.

Attribute definitions:

SELF\externally_defined_item.source : The name of the part of ISO 13584 where the style is defined

SELF_externally_defined_item.item_id: The **identifier** of the style that shall be used.

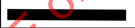
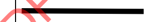
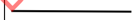
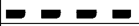
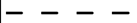
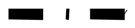
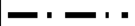
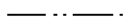
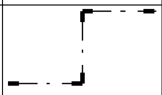
6.2.4.2 Api_externally_defined_curve_style

An **api_externally_defined_curve_style** specifies the visual appearance of **curve**s.

The following curve styles are defined in this part of ISO 13584. The size, colour and precise pattern of these styles are implementation dependent (impl. dep.). Default values shall be provided by the interface implementor. These default values shall be customisable by the interface end user.

NOTE 1 - Other **api_externally_defined_curve_style**s may be defined by the view exchange protocol series of parts of ISO 13584. If one interface implementation does not support this view exchange protocol, the fist style define by this part of ISO13584 shall be used, and no error shall be reported.

Table 9 — Externally defined curve styles

item_id	**description**	**drawing**	**colour, width and pattern**
'plain_solid_line_thick'	continuous - thick		impl. dep.
‘ plain_solid_line_middle'	continuous - middle thick.		impl. dep.
'plain_solid_line_thin'	continuous - thin		impl. dep.
'plain_dashed_line_thick'	dashed - thick		impl. dep.
'plain_dashed_line_thin'	dashed - thin.		impl. dep.
'alternate_long_dash_dot_line _thick'	Alternate long dash followed by a dot - thick		impl. dep.
'alternate_long_dash_dot_line _thin'	Alternate long dash followed by a dot - thin		impl. dep.
'alternate_long_dash_double_ dot_line_thin'	Alternate long dash followed by two dots - thin		impl. dep.
'alternate_long_dash_dot_line _thin_thicken_parts'	Alternate long dash followed by a dot - thin, continuos thick at the beginning and end and at each change of direction		impl. dep.
'virtual_line'	used in intermediate constructions	undefined	undefined

EXPRESS specification:

```
*)
ENTITY api_externally_defined_curve_style
 SUBTYPE OF (externally_defined_style);
END_ENTITY;
(*
```

NOTE 2 - This entity may be implemented as an **externally_defined_style**.

Attribute definitions:

SELF\externally_defined_item.source: The name of the part of ISO 13584 where the style is defined

SELF\externally_defined_item.item_id: The **identifier** of the style that shall be used.

6.2.4.3 Api_externally_defined_fill_area_style

An **api_externally_defined_fill_area_style** specifies the visual appearance of a fill area. The following fill area styles are defined in this part of ISO 13584. The colour for 'opaque_fill_area' is implementation dependent (impl. dep.). It shall correspond to the background colour

NOTE 1 - Other **api_externally_defined_fill_area_style**s may be defined by the view exchange protocol series of parts of ISO 13584. If one interface implementation does not support this view exchange protocol, the fist style define by ISO_13584_31 shall be used, and no error shall be reported.

Table 10 — Externally defined fill area styles

item_id	**Description**	**colour**
'opaque_fill_area'	The fill area is filled with the background colour. It may hide or blank out other entities. It may also be hatched	impl. dep.
'empty_fill_area'	The fill area is not filled with any colour. It may not hide or blank out other entities. It may be hatched	no colour.

EXPRESS specification:

```
*)
ENTITY api_externally_defined_fill_area_style
 SUBTYPE OF (externally_defined_style);
END_ENTITY;
(*
```

NOTE 2 - This entity may be implemented as an **externally_defined_style**.

Attribute definitions:

SELF\externally_defined_item.source: The name of the part of ISO 13584 where the style is defined

SELF\externally_defined_item.item_id: The identifier of the style that shall be used.

6.2.4.4 Api_externally_defined_surface_style

An **api_externally_defined_surface_style** specifies the visual appearance of a surface.

The following surface style is defined in this part of ISO 13584. All the visual appearance of the surface are implementation dependent (impl. dep.). Default values shall be provided by the interface implementor. These default values shall be customisable by the interface end user.

NOTE 1 - Other **api_externally_defined_surface_style**s may be defined by the view exchange protocol series of parts of ISO 13584. If one interface implementation does not support this view exchange protocol, the fist style define by this part of ISO13584 shall be used, and no error shall be reported.

Table 11 — Externally defined surface style

item_id	**Description**	**appearance**
'solid_surface'	The surface shall be displayed. It shall be presented in the current style defined on the receiving system for surface rendering	impl. dep.

EXPRESS specification:

```
*)
ENTITY api_externally_defined_surface_style
```

```
 SUBTYPE OF (externally_defined_style);
END_ENTITY;
(*
```

NOTE 2 - This entity may be implemented as an **externally_defined_style**.

Attribute definitions:

SELF\externally_defined_item.source: The name of the part of ISO 13584 where the style is defined

SELF\externally_defined_item.item_id: The identifier of the style that shall be used.

6.2.5 API_ABSTRACT_SCHEMA entities definition : pre-defined styles for visual presentation

This subclause declares the pre-defined styles that are specified in this part of ISO 13584 and may be referenced to by the application programs to define the (logical) visual appearance of the geometric or annotation entities. Pre-defined style properties are specified for hatch lines and for properties that refer to occlusion precedence in two dimensional views.

6.2.5.1 Api_pre_defined_hatch_width

An **api_pre_defined_hatch_width** specifies the logical width of an hatch line.

The following widths are pre-defined in this part of ISO 13584. The precise value of the width is implementation dependent (Implement. dep.). Default values shall be provided by the interface implementor. These default values shall be customisable by the interface end user.

Table 12 — Pre_defined hatch line width

Name	description	width value
'thin_hatch_line'	thin. Used for hatching material like iron	impl. dep.
'middle_thick_hatch_line'	middle thick as used for specific purpose	impl. dep.
'thick_hatch_line'	thick as used for specific purpose	impl. dep.

EXPRESS specification:

```
*)
ENTITY api_pre_defined_hatch_width
 SUBTYPE OF (pre_defined_size);
WHERE
 api_WR1: SELF\pre_defined_item.name IN ['thin_hatch_line',
                    'middle_thick_hatch_line',
                    'thick_hatch_line'];
END_ENTITY;
(*
```

NOTE - When the hatching is created in the CAD system database, the curve width of the hatching may be represented by a **positive_length_measure** according to the value generated by the interface implementation.

Attribute definitions:

SELF\pre_defined_item.name: The **label** of the width that shall be used.

Formal propositions:

api_WR1: The **name** shall be one **name** defined in this part of ISO 13584.

6.2.5.2 Api_pre_defined_hatch_curve_font

An **api_pre_defined_hatch_curve_font** specifies the curve font of an hatch line.

The following curve fonts are pre-defined in this part of ISO 13584.

Table 13 — Line segment and space lengths for Pre_defined hatch curve font

Name	segment (mm)	space (mm)	segment (mm)	space (mm)	segment (mm)	space (mm)
'continuous'						
'dashed'	4.0	1.5	1.0	1.0		
'chain'	7.0	1.0	1.0	1.0	1.0	1.0
'chain_double_dash'	7.0	1.0				
'dotted'	1.0	1.0				

<u>EXPRESS specification:</u>

```
*)
ENTITY api_pre_defined_hatch_curve_font
 SUBTYPE OF (pre_defined_curve_font);
WHERE
 api_WR1: SELF\pre_defined_item.name IN ['continuous',
                     'dashed',
                     'chain',
                     'chain_double_dash',
                     'dotted'];
END_ENTITY;
(*
```

<u>Attribute definitions:</u>

SELF\pre_defined_item.name: The **label** of the curve font that shall be used.

<u>Formal propositions:</u>

api_WR1: The **name** shall be one **name** defined in this part of ISO 13584.

6.2.5.3 Api_pre_defined_hatch_colour

An **api_pre_defined_hatch_colour** specifies the logical colour of an hatch line.

The following colour is pre-defined in this part of ISO 13584. The precise value of the colour is implementation dependent (Implement. dep.). Default values shall be provided by the interface implementor. This default value shall be customisable by the interface end user.

Table 14 — Pre_defined hatching colour

Name	description	colour value
'hatch_line_colour'	a colour specified for hatching	impl. dep.

<u>EXPRESS specification:</u>

```
*)
ENTITY api_pre_defined_hatch_colour
 SUBTYPE OF (pre_defined_colour);
```

```
WHERE
 api_WR1: SELF\pre_defined_item.name ='hatch_line_colour';
END_ENTITY;
(*
```

NOTE - When the hatching is created in the CAD system database, the colour of the hatching may be represented by a colour value according to the precise colour generated by the interface implementation.

Attribute definitions:

SELF\pre_defined_item.name : The **label** of the colour that shall be used.

Formal propositions:

api_WR1: The **name** shall be the only **name** defined in this part of ISO 13584.

6.2.5.4 Api_pre_defined_occlusion_style

An **api_pre_defined_occlusion_style** specifies that the **styled_item** shall be involved in the global hidden line removal process and defines the virtual height of the **styled_item** in the virtual 3D space.

NOTE 1 - When a **styled_item** is not associated with this style, it is not involved in the hidden line removal process.

NOTE 2 - If the CAD system provides resources for occlusion precedence relationship in 2D, this style shall be mapped onto these resources and the hidden line removal process shall be performed by the CAD system.

NOTE 3 - When the **shape_representation** is to be created in an ISO 10303 repository, conforming to an AP that provides for occlusion precedence relationship, the hidden line removal process shall map this style onto the occlusion precedence relationship. If the ISO 10303 AP does not provide for occlusion precedence relationship, the hidden line shall be computed by the interface and only the visible EXPRESS entities shall be created in the ISO 10303 AP conforming repository.

Table 15 — Pre_defined hidden line style

Name	**description**	**colour value**
'hidden_line_no_changed'	the hidden line remains unchanged	impl. dep.
'hidden_line_dashed'	the hidden line are dashed	impl. dep.
'hidden_line_invisible'	the hidden line are invisible	impl. dep.

EXPRESS specification:

```
*)
ENTITY api_pre_defined_occlusion_style
 SUBTYPE OF (pre_defined_presentation_style);
 view_level: virtual_height_ratio;
WHERE
 api_WR1: SELF\pre_defined_item.name IN ['hidden_line_no_changed',
                     'hidden_line_dashed',
                     'hidden_line_invisible'];
END_ENTITY;
(*
```

Attribute definitions:

SELF\pre_defined_item.name : The **label** of the hidden line style that shall be used.

view_level: The virtual height of the **styled_item** in the virtual 3D space.

Formal propositions:

api_WR1: The pre-defined **name** of this style shall be 'hidden_line_no_changed', 'hidden_line_dashed' or 'hidden_line_invisible'.

6.2.5.5 Api_pre_defined_virtually_sent_style

An **api_pre_defined_virtually_sent_style** specifies that a **styled_item** that remains in the TDB for the global hidden line removal process is no longer accessible to the application program and shall be sent to the CAD system after the hidden line removal process.

NOTE 1 - This **pre_defined_presentation_style** shall only be used for the entities that are within the TDB.

NOTE 2 - When an entity is virtually sent, it shall be removed from the **api_group** structure and shall belong to the root group that cannot be used in any group manipulation function.

EXPRESS specification:

```
*)
ENTITY api_pre_defined_virtually_sent_style
 SUBTYPE OF (pre_defined_presentation_style);
 api_set_name: STRING;
WHERE
 api_WR1: SELF\pre_defined_item.name ='virtually_sent';
END_ENTITY;
(*
```

Attribute definitions:

Api_set_name The **name** of the **api_set** that was the open set when the entity was virtually sent.

Formal propositions:

api_WR1: The pre-defined **name** of this style shall be 'virtually_sent'.

Informal propositions:

api_IP1: This style shall only be assigned to entities that are in the TDB.

api_IP2: The **api_set_name** shall correspond to the **name** of an **api_set**.

6.3 API_ABSTRACT_SCHEMA function definition

6.3.1 API_ABSTRACT_SCHEMA function definition : Geometric and topological representations

This subclause declares the functions defined in ISO 10303-42 that are part of the **api_abstract_schema**.

6.3.1.1 Dimension_of

The function **dimension_of** returns the integer **dimension_count** of a **geometric_representation_context** in which the input **geometric_representation_item** is geometrically founded.

By virtue of the constraints in global rule **compatible_dimension**, this value is the **coordinate_space_dimension** of the input **geometric_representation_item**. See the rule **compatible_-dimension** in ISO 10303-42 section 4.5.1.

EXPRESS specification:

```
*)
```

```
FUNCTION dimension_of(item : geometric_representation_item) :
 dimension_count;
 LOCAL
  x : SET OF representation;
  y : representation_context;
 END_LOCAL;
 -- Find the set of representation in which the item is used.
 x := using_representations(item);
 -- Determine the dimension_count of the
 -- geometric_representation_context. Note that the
 -- RULE compatible_dimension ensures that the context_of_items
 -- is of type geometric_representation_context and has
 -- the same dimension_count for all values of x.
 y := x[1].context_of_items;
 RETURN (y\geometric_representation_context.coordinate_space_dimension);
END_FUNCTION;
(*
```

Argument definitions:

item: (input) A **geometric_representation_item** for which the **dimension_count** is determined.

6.3.1.2 Associated_surface

The **associated_surface** function determines the unique surface that is associated with the **pcurve_or_surface** type. It is required by the propositions that apply to surface curve and its subtypes.

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.basis_surface;
 ELSE
  surf := arg;
 END_IF;
 RETURN(surf);
END_FUNCTION;
(*
```

Argument definitions:

arg: (input) The pcurve or surface for which the determination of the associated parent surface is required.

surf: (output) The parent surface associated with **arg**.

6.3.1.3 Base_axis

This function returns three normalised orthogonal directions, u[1], u[2] and u[3]. In the three-dimensional case, with complete input data, u[3] is in the direction of axis3, u[1] is in the direction of the projection of axis1 onto the plane normal to u[3], and u[2] is orthogonal to both u[1] and u[3], taking the same sense as axis2. In the two-dimensional case u[1] is in the direction of axis1 and u[2] is perpendicular to this, taking its sense from axis2. For incomplete input data, suitable default values are derived.

EXPRESS specification:

```
*)
```

```
FUNCTION base_axis(dim : INTEGER; axis1, axis2, axis3 : direction) :
                        LIST [2:3] OF direction;
 LOCAL
  vec  : direction;
  u   : LIST [2:3] OF direction;
  factor : REAL;
 END_LOCAL;
 IF (dim = 3) THEN
  u[3] := NVL(axis3, direction([0.0,0.0,1.0]));
  u[1] := first_proj_axis(u[3],axis1);
  u[2] := second_proj_axis(u[3],u[1],axis2);
 ELSE
  u[3] := ?;
  IF EXISTS(axis1) THEN
   u[1] := normalise(axis1);
   u[2] := orthogonal_complement(u[1]);
   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
    u[2] := normalise(axis2);
    u[1] := orthogonal_complement(u[2]);
    u[1].direction_ratios[1] := -u[1].direction_ratios[1];
    u[1].direction_ratios[2] := -u[1].direction_ratios[2];
   ELSE
    u[1].direction_ratios[1] := 1.0;
    u[1].direction_ratios[2] := 0.0;
    u[2].direction_ratios[1] := 0.0;
    u[2].direction_ratios[2] := 1.0;
   END_IF;
  END_IF;
 END_IF;
 RETURN(u);
END_FUNCTION;
(*
```

Argument definitions:

dim: (input) The integer value of the dimensionality of the space in which the normalised orthogonal directions are required.

axis1: (input) A direction used as a first approximation to the direction of output axis u[1].

axis2: (input) A direction used to determine the sense of u[2].

axis3: (input) The direction of u[3] in the case dim=3, or NULL in the case dim=2.

u: (output) A list of dim (i.e. 2 or 3) mutually perpendicular directions.

6.3.1.4 Build_2axes

This function returns two normalised orthogonal directions. **u[1]** is in the direction of **ref_direction** and **u[2]** is perpendicular to **u[1]**. A default value of (1.0,0.0) is supplied for **ref_direction** if the input data is incomplete.

EXPRESS specification:

```
*)
FUNCTION build_2axes(ref_direction : direction) : LIST [2:2] OF direction;
 LOCAL
  u : LIST[2:2] OF direction;
 END_LOCAL;
 u[1] := NVL(normalise(ref_direction), direction([1.0,0.0]));
 u[2] := orthogonal_complement(u[1]);
 RETURN(u);
END_FUNCTION;
(*
```

Argument definitions:

ref_direction: (input) A reference direction in 2 dimensional space, this may be defaulted to [1.0,0.0].

u: (output) A list of 2 mutually perpendicular directions, u[1] is parallel to ref_direction.

6.3.1.5 Build_axes

This function returns three normalised orthogonal directions. u[3] is in the direction of **axis**, u[1] is in the direction of the projection of **ref_direction** onto the plane normal to u[3] and u[2] is the cross product of u[3] and u[1]. Default values are supplied if input data is incomplete.

EXPRESS specification:

```
*)
FUNCTION build_axes(axis, ref_direction : direction) :
                    LIST [3:3] OF direction;
 LOCAL
  u : LIST[3:3] OF direction;
 END_LOCAL;

 u[3] := NVL(normalise(axis), direction([0.0,0.0,1.0]));
 u[1] := first_proj_axis(u[3],ref_direction);
 u[2] := normalise(cross_product(u[3],u[1])).orientation;
 RETURN(u);
END_FUNCTION;
(*
```

Argument definitions:

axis: (input) The intended direction of u[3], this may be defaulted to [0.0,0.0,1.0].

ref_direction: (input) A direction in a direction used to compute u[1].

u: (output) A list of 3 mutually orthogonal directions in 3D space.

6.3.1.6 Orthogonal_complement

The **orthogonal_complement** function returns a **direction** that is the orthogonal complement of the input **direction**. The input **direction** must be a two-dimensional **direction** and the result is a **vector** of the same type and perpendicular to the input **vector**.

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.direction_ratios[1] := -vec.direction_ratios[2];
  result.direction_ratios[2] := vec.direction_ratios[1];
  RETURN(result);
 END_IF;
END_FUNCTION;
(*
```

Argument definitions:

vec: (input) A direction in 2D space.

result: (output) A direction orthogonal to vec.

6.3.1.7 First_proj_axis

This function produces a 3-dimensional direction that is, with fully defined input, the projection of **arg** onto the plane normal to the **z_axis**. With **arg** defaulted the result is the projection of [1,0,0] onto this plane except that if **z_axis** = [1,0,0], [0,1,0] is the default for **arg**. A violation occurs if **arg** is in the same direction as the input **z_axis**.

EXPRESS specification:

```
*)
FUNCTION first_proj_axis(z_axis, arg : direction) : direction;
 LOCAL
  x_axis : direction;
  v    : direction;
  z    : direction;
  x_vec : vector;
 END_LOCAL;
 IF NOT EXISTS(z_axis) OR (NOT EXISTS(arg)) OR (arg.dim <> 3) THEN
   x_axis := ?;
 ELSE
  z_axis := normalise(z_axis);
  IF NOT EXISTS(arg) THEN
   IF (z_axis <> direction([1.0,0.0,0.0])) THEN
    v := direction([1.0,0.0,0.0]);
   ELSE
    v := direction([0.0,1.0,0.0]);
   END_IF;
  ELSE
   IF ((cross_product(arg,z).magnitude) = 0.0) THEN
    RETURN (?);
   ELSE
    v := normalise(arg);
   END_IF;
  END_IF;
  x_vec := scalar_times_vector(dot_product(v, z), z_axis);
  x_axis := vector_difference(v, x_vec).orientation;
  x_axis := normalise(x_axis);
 END_IF;
 RETURN(x_axis);
END_FUNCTION;
(*
```

Argument definitions:

z_axis: (input) A direction defining a local Z axis.

arg: (input) A direction not parallel to z_axis.

x_axis: (output) A direction that is in the direction of the projection of arg onto the plane with normal z_axis.

6.3.1.8 Second_proj_axis

This function returns the normalised vector that is simultaneously the projection of **arg** onto the plane normal to the vector **z_axis** and onto the plane normal to the vector **x_axis**. If **arg** is NULL then the projection of the vector (0,1,0) onto **z_axis** is returned.

EXPRESS specification:

```
*)
FUNCTION second_proj_axis(z_axis, x_axis, arg: direction) : direction;
 LOCAL
  y_axis : vector;
  v    : direction;
  temp  : vector;
 END_LOCAL;

 IF NOT EXISTS(arg) THEN
  v := direction([0.0,1.0,0.0]);
 ELSE
  v := arg;
 END_IF;

 temp  := scalar_times_vector(dot_product(v, z_axis), z_axis);
 y_axis := vector_difference(v, temp);
 temp  := scalar_times_vector(dot_product(v, x_axis), x_axis);
 y_axis := vector_difference(y_axis, temp);
 y_axis := normalise(y_axis);
 RETURN(y_axis.orientation);
END_FUNCTION;
(*
```

Argument definitions:

z_axis: (input) A direction defining a local Z axis.

x_axis: (input) A direction not parallel to **z_axis**.

arg: (input) A direction that is used as the first approximation to the direction of **y_axis**.

y_axis.orientation: (output) A direction determined by first projecting arg onto the plane with normal **z_axis**, then projecting the result onto the plane normal to **x_axis**.

6.3.1.9 Cross_product

This function returns the vector, or cross, product of two input directions. The input directions must be three-dimensional and are normalised at the start of the computation. The result is always a vector that is unitless. If the input directions are either parallel or anti-parallel a vector of zero magnitude is returned with **orientation = arg1**.

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.direction_ratios[1] := (v1[2]*v2[3] - v1[3]*v2[2]);
   res.direction_ratios[2] := (v1[3]*v2[1] - v1[1]*v2[3]);
   res.direction_ratios[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.orientation := res;
    result.magnitude := SQRT(mag);
   ELSE
    result.orientation := arg1;
    result.magnitude := 0.0;
   END_IF;
   RETURN(result);
  END;
 END_IF;
END_FUNCTION;
(*
```

Argument definitions:

arg1: (input) A direction defining first operand in cross product operation.

arg2: (input) A direction defining second operand in cross product.

result: (output) A vector that is the cross product of **arg1** and **arg2**.

6.3.1.10 Dot_product

This function returns the scalar, or dot (·), product of two directions. The input arguments can be directions in either two- or three-dimensional space and are normalised at the start of computation. The returned scalar is undefined if the input directions have different dimensionality, or if either is undefined.

EXPRESS specification:

```
*)
FUNCTION dot_product(arg1, arg2 : direction) : REAL;
 LOCAL
  scalar : REAL;
  vec1, vec2: direction;
  ndim : INTEGER;
 END_LOCAL;
 IF NOT EXISTS (arg1) OR NOT EXISTS (arg2) THEN
  scalar := ?;
  (* When function is called with invalid data
                        a NULL result is returned
*)
 ELSE
```

```
  IF (arg1.dim <> arg2.dim) THEN
   scalar := ?;
   (* When function is called with invalid data
                             a NULL result is returned
*)
  ELSE
   BEGIN
    vec1  := normalise(arg1);
    vec2  := normalise(arg2);
    ndim  := arg1.dim;
    scalar := 0.0;
    REPEAT i := 1 TO ndim;
     scalar := scalar +
           vec1.direction_ratios[i]*vec2.direction_ratios[i];
    END_REPEAT;
   END;
   RETURN (scalar);
  END_IF;
 END_IF;
END_FUNCTION;
(*
```

Argument definitions:

arg1: (input) A direction defining first operand in dot product, or scalar product operation.

arg2: (input) A direction defining second operand for dot product.

result: (output) A scalar that is the dot product of **arg1** and **arg2**.

6.3.1.11 Normalise

This function returns a vector or direction whose components are normalised to have a sum of squares of 1.0. The output is of the same type (**direction** or **vector**, with the same units) as the input argument. If the input argument is not defined or is of zero length then the output vector is undefined.

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 'API_ABSTRACT_SCHEMA.VECTOR' IN TYPEOF(arg) THEN
   BEGIN
    vec := arg;
    v := arg.orientation;
    IF arg.magnitude = 0.0 THEN
     RETURN(?);
    ELSE
     vec.magnitude := 1.0;
    END_IF;
   END;
  ELSE
   v := arg;
  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 'API_ABSTRACT_SCHEMA.VECTOR' IN TYPEOF(arg) THEN
    vec.orientation := v;
    result := vec;
   ELSE
    result := v;
   END_IF;
  ELSE
   RETURN(?);
  END_IF;
  RETURN (result);
 END_IF;
END_FUNCTION;
(*
```

Argument definitions:

arg: (input) A vector or direction to be normalised.

result: (output) a vector or direction that is parallel to **arg1** and of unit length.

6.3.1.12 Scalar_times_vector

This function returns the vector that is the scalar multiple of the input vector. It accepts as input a scalar and a 'vector' that may be either a **direction** or a **vector**. The output is a **vector** of the same units as the input **vector**, or unitless if a direction is input. If either input argument is undefined, the returned vector is also undefined. The **orientation** of the **vector** is reversed if the scalar is negative.

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
  result := ?;
  (* When function is called with invalid data
                        a NULL result is returned
*)
 ELSE
  IF 'API_ABSTRACT_SCHEMA.VECTOR' IN TYPEOF (vec) THEN
   v  := vec.orientation;
   mag := scalar * vec.magnitude;
  ELSE
```

```
   v  := vec;
   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.orientation := normalise(v);
  result.magnitude  := mag;
 END_IF;
  RETURN (result);
END_FUNCTION;
(*
```

Argument definitions:

scalar: (input) A real number to participate in the product.

vec: (input) A vector or direction that is to be multiplied.

result: (output) A vector that is the product of **scalar** and **vec**.

6.3.1.13 Vector_sum

This function returns the vector sum of the input arguments. The function returns a vector that is the vector sum of the two input 'vectors'. For this purpose **direction**s are treated as unit vectors. The input arguments must both of the same dimensionality but may be either directions or vectors. Where both arguments are vectors, they must be expressed in the same units. A zero sum vector produces a vector of zero magnitude in the direction of **arg1**. If both input arguments are directions, the result is unitless.

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
  result := ?;
  (* When function is called with invalid data
                      a NULL result is returned
*)
 ELSE
  BEGIN
   IF 'API_ABSTRACT_SCHEMA.VECTOR' IN TYPEOF(arg1) THEN
    mag1 := arg1.magnitude;
    vec1 := arg1.orientation;
   ELSE
    mag1 := 1.0;
    vec1 := arg1;
   END_IF;
   IF 'API_ABSTRACT_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;
   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.magnitude := SQRT(mag);
    result.orientation := res;
   ELSE
    result.magnitude := 0.0;
    result.orientation := vec1;
   END_IF;
  END;
 END_IF;
 RETURN (result);
END_FUNCTION;
(*
```

Argument definitions:

arg1: (input) A direction defining first vector in vector sum operation.

arg2: (input) A direction defining second operand in vector sum.

result: (output) A vector that is the vector sum of **arg1** and **arg2**.

6.3.1.14 Vector_difference

This function returns the difference of the input arguments as (**arg1** - **arg2**). The function returns as a vector the vector difference of the two 'vectors'. For this purpose **direction**s are treated as unit vectors. The input arguments shall both be of the same dimensionality but may be either directions or vectors. If both input arguments are vectors, they must be expressed in the same units; if both are directions, an unitless result is produced. A zero difference vector produces a vector of zero magnitude in the direction od **arg1**.

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
  result := ?;
  (* When function is called with invalid data
                        a NULL result is returned
*)
 ELSE
  BEGIN
   IF 'API_ABSTRACT_SCHEMA.VECTOR' IN TYPEOF(arg1) THEN
    mag1 := arg1.magnitude;
    vec1 := arg1.orientation;
   ELSE
    mag1 := 1.0;
    vec1 := arg1;
   END_IF;
   IF 'API_ABSTRACT_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);
   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.magnitude := SQRT(mag);
    result.orientation := res;
   ELSE
    result.magnitude := 0.0;
    result.orientation := vec1;
   END_IF;
  END;
 END_IF;
 RETURN (result);
END_FUNCTION;
(*
```

Argument definitions:

arg1: (input) A direction defining first vector in vector difference operation.

arg2: (input) A direction defining second operand for vector difference.

result: (output) A vector that is the vector difference of **arg1** and **arg2**.

6.3.1.15 Constraints_composite_curve_on_surface

This function checks that the curves referenced by the segments of the **composite_curve_on_surface** are all curves on the surface, including the **composite_curve_on_surface** type, that is admissible as a **bounded_curve**.

EXPRESS specification:

```
*)
FUNCTION constraints_composite_curve_on_surface
       (c : composite_curve_on_surface) : BOOLEAN;
 LOCAL
  n_segments : INTEGER := SIZEOF(c.segments);
 END_LOCAL;

 REPEAT k := 1 TO n_segments;
  IF (NOT('GEOMETRY_SCHEMA.PCURVE' IN
     TYPEOF (c\composite_curve.segments[k].parent_curve))) AND
    (NOT('API_ABSTRACT_SCHEMA.SURFACE_CURVE' IN
     TYPEOF (c\composite_curve.segments[k].parent_curve))) AND
    (NOT('API_ABSTRACT_SCHEMA.COMPOSITE_CURVE_ON_SURFACE' IN
     TYPEOF (c\composite_curve.segments[k].parent_curve))) THEN
   RETURN (FALSE);
  END_IF;
 END_REPEAT;
 RETURN (TRUE);
END_FUNCTION;
(*
```

Argument definitions:

c: (input) A composite curve on surface to be verified.

6.3.1.16 Get_basis_surface

This function returns the basis surface for a curve as a SET of **surface**s. For a curve that is not a **curve_on_surface** an empty SET is returned.

EXPRESS specification:

```
*)
FUNCTION get_basis_surface(c : curve_on_surface) : SET[0:2] OF surface;
 LOCAL
  surfs : SET[0:2] OF surface;
  n    : INTEGER;
 END_LOCAL;

 surfs := [];
 IF 'GEOMETRY_SCHEMA.PCURVE' IN TYPEOF (c) THEN
  surfs := [c\pcurve.basis_surface];
 ELSE
  IF 'API_ABSTRACT_SCHEMA.SURFACE_CURVE' IN TYPEOF (c) THEN
   n := SIZEOF(c\surface_curve.associated_geometry);
   REPEAT i := 1 TO n;
   surfs := surfs +
        associated_surface(c\surface_curve.associated_geometry[i]);
   END_REPEAT;
  END_IF;
 END_IF;
 IF 'API_ABSTRACT_SCHEMA.COMPOSITE_CURVE_ON_SURFACE' IN TYPEOF (c) THEN
  (* For a composite_curve_on_surface the basis_surface is the
   intersection of the basis_surface of all the segments.
*)
  n := SIZEOF(c\composite_curve_on_surface.segments);
  surfs := get_basis_surface(c\composite_curve_on_surface.segments[1].
                              parent_curve);
  IF n > 1 THEN
   REPEAT i := 2 TO n;
    surfs := surfs *
         get_basis_surface(c\composite_curve_on_surface.
                    segments[1].parent_curve);
   END_REPEAT;
  END_IF;
 END_IF;
 RETURN(surfs);
END_FUNCTION;
(*
```

Argument definitions:

c: (input) A curve for which the **basis_surface** is to be determined.

surf: (output) The set containing the **basis_surface** or surface on which **c** lies..

6.3.1.17 List_to_array

The function **list_to_array** converts a generic list to an array with pre-determined array bounds. If the array bounds are incompatible with the number of elements in the original list a null result is returned. This function is used to construct the arrays of control points and weights used in the b-spline entities.

EXPRESS specification:

```
*)
FUNCTION list_to_array(lis : LIST [0:?] OF GENERIC : T;
           low,u : INTEGER) : ARRAY[low:u] 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
  REPEAT i := 1 TO n;
   res[low+i-1] := lis[i];
  END_REPEAT;
  RETURN(res);
 END_IF;
END_FUNCTION;
(*
```

Argument definitions:

lis: (input) A list to be converted.

low: (input) An integer specifying the required lower index of the output array.

u: (input) An integer value for the upper index.

res: (output) The array generated from the input data.

6.3.1.18 Make_array_of_array

The function **make_array_of_array** builds an array of arrays from a list of lists. The function first checks that the specified array dimensions are compatible with the sizes of the lists, and in particular verifies that all the sub-lists contains the same number of elements. A null result is returned if the input data is incompatible with the dimensions. This function is used to construct the arrays of control points and weights for a B-spline surface.

EXPRESS specification:

```
*)
FUNCTION make_array_of_array(lis : LIST[1:?] OF LIST [1:?] OF GENERIC : T;
                low1, u1, low2, u2 : INTEGER):
        ARRAY[low1:u1] OF ARRAY [low2:u2] OF GENERIC : T;
 LOCAL
  n1,n2 : INTEGER;
  res   : ARRAY[low1:u1] OF ARRAY [low2:u2] OF GENERIC : T;
  resl : LIST[1:?] OF ARRAY [low2:u2] OF GENERIC : T;
 END_LOCAL;

 (* Check input dimensions for consistency
*)
 n1 := SIZEOF(lis);
 n2 := SIZEOF(lis[1]);

 IF (n1 <> (u1 -low1 + 1)) AND (n2 <> (u2 - low2 + 1)) THEN
  RETURN(?);
 END_IF;

 REPEAT i := 1 TO n1;
  IF (SIZEOF(lis[i]) <> n2) THEN
   RETURN(?);
  END_IF;
 END_REPEAT;

 (* Build a list of sub-arrays
*)
 REPEAT i := 1 TO n1;
  RESL[i] := list_to_array(lis[i],low2,u2);
 END_REPEAT;

 res := list_to_array(resl,low1,u1);
 RETURN(res);
END_FUNCTION;
(*
```

Argument definitions:

lis: (input) A list of list to be converted.

low1: (input) An integer specifying the required lower index of the first output array.

u1: (input) An integer value for the upper index of the first output array.

low2: (input) An integer specifying the required lower index of the second output array.

u2: (input) An integer value for the upper index of the second output array.

res: (output) The array of array with specified dimensions generated from the input data after verifying consistency.

6.3.2 API_ABSTRACT_SCHEMA function definition: Support resources

This subclause declares the functions defined in ISO 10303-41 that are part of the **api_abstract_schema.**

6.3.2.1 Bag_to_set

This function converts BAGs into SETs.

EXAMPLE - It can be used to convert the BAGs returned by USEDIN function into SETs that can be properly assigned to variables are SETs.

EXPRESS specification:

```
*)
FUNCTION bag_to_set(the_bag : BAG OF GENERIC : intype)
              : SET OF GENERIC : intype;

 LOCAL
  the_set : SET OF GENERIC : intype := [];
  i     : INTEGER;
 END_LOCAL;

 IF SIZEOF (the_bag) > 0 THEN
  REPEAT i := 1 TO HIINDEX (the_bag);
```

```
   the_set := the_set + the_bag [i];
  END_REPEAT;
 END_IF;

 RETURN (the_set);

END_FUNCTION;
(*
```

Argument definitions:

the_bag: The BAG that is to be converted into a SET.

6.3.3 API_ABSTRACT_SCHEMA function definition: Representation structures

This subclause declares the functions defined in ISO 10303-43 that are part of the **api_abstract_schema.**

6.3.3.1 Acyclic_mapped_representation

The function **acyclic_mapped_representation** determines if a given **mapped_item** is self-defining by virtue of mapping a **representation** in which the **mapped_item** is used. The function is extended to check both the **mapped_representation** and the **mapped_representation.items** recursively for any **mapped_item**s or **representation_item**s referencing a **mapped_item** that might cause a cyclic reference. This function returns TRUE if the input candidate **representation_item** does not cause self definition. It returns FALSE otherwise. The type of the function is **BOOLEAN**.

This function is used to constrain the entity **mapped_item**.

EXPRESS specification:

```
*)
FUNCTION acyclic_mapped_representation
 (parent_set  : SET OF representation;
  children_set : SET OF representation_item) : BOOLEAN;
 LOCAL
  x,y : SET OF representation_item;
  i,j : INTEGER;
 END_LOCAL;
 -- Determine the subset of children_set that are mapped_items.
 x := QUERY(z <* children_set | 'API_ABSTRACT_SCHEMA.MAPPED_ITEM'
    IN TYPEOF(z));
 -- Determine that the subset has elements.
 IF SIZEOF(x) > 0 THEN
  -- Check each element of the set.
  REPEAT i := 1 TO HIINDEX(x);
   -- If the selected element maps a representation in the
   -- parent_set, return false.
   IF x[i]\mapped_item.mapping_source.mapped_representation
    IN parent_set THEN
    RETURN (FALSE);
   END_IF;
   -- Recursively check the items of the mapped_rep.
   IF NOT acyclic_mapped_representation
    (parent_set +
     x[i]\mapped_item.mapping_source.mapped_representation,
     x[i]\mapped_item.mapping_source.mapped_representation.items) THEN
     RETURN (FALSE);
   END_IF;
  END_REPEAT;
 END_IF;
 -- Determine the subset of children_set that are not mapped_items.
 x := children_set - x;
 -- Determine that the subset has elements.
 IF SIZEOF(x) > 0 THEN
  -- For each element of the set:
  REPEAT i := 1 TO HIINDEX(x);
   -- Determine the set of representation_items referenced.
   y := QUERY(z <* bag_to_set( USEDIN(x[i], '') ) |
      'API_ABSTRACT_SCHEMA.REPRESENTATION_ITEM' IN TYPEOF(z));
   -- Recursively check these in case they might be an offending
   -- mapped_item. Return false for any errors encountered.
   IF NOT acyclic_mapped_representation(parent_set, y) THEN
    RETURN (FALSE);
   END_IF;
  END_REPEAT;
 END_IF;
 -- Return true when all elements are checked and
 -- no error conditions found.
 RETURN (TRUE);
END_FUNCTION;
(*
```

Argument definitions:

parent_set: The set of **representation**s in which the **mapped_item** is used. This is input to the function. On initial input, this is the set of **representation**s in which the **mapped_item** being checked is used and is modified in recursive calls.

children_set: The set of **representation_item**s that might possibly be a **mapped_item** and are referenced directly or indirectly through the **items** of the **representations** in the **parent_set**. This is input to the function. On initial input this is the **mapped_item** being checked and is modified in recursive calls.

6.3.3.2 Item_in_context

The function **item_in_context** determines if a **representation_item** is related to a **representation_context**. The function returns TRUE if:

– the **item** argument is related by a **representation** to the input **cntxt** argument.

– the **item** argument is related by a **definitional_representation** to the input **cntxt** argument.

Function **item_in_context** returns FALSE otherwise. The type of the function is **BOOLEAN**.

A **representation_item** is related to a **representation_context** if it is any of the following:

1) referenced in the set of **item**s of a **representation** where **cntxt** appears as the **context_of_items**;

2) referenced in the set of **items** of a **definitional_representation_item** where **cntxt** appears as the **context_of_items**;

3) referenced by a **representation_item** that is an **item_in_context** of the **cntxt**.

NOTE 1 - The third condition is a recursive check allowing for a **representation_item** to be related to a **representation_context** by being part of a tree of related **representation_item**s. The tree is rooted in an entity that is related to a **representation_context** by fulfilling the first or second condition.

NOTE 2 - The function **item_in_context** only determines if an **item** is related to a specific **representation_context**. The relationship of the **item** to some other **representation_context** is not determined.

EXPRESS specification:

```
*)
```

```
FUNCTION item_in_context
 (item : representation_item;
  cntxt : representation_context) : BOOLEAN;

 LOCAL
  i : INTEGER;
  y : BAG OF representation_item;
 END_LOCAL;
 -- If there is one or more representation using both the item
 -- and cntxt return true.
 IF SIZEOF(USEDIN(item,'API_ABSTRACT_SCHEMA.REPRESENTATION.ITEMS')
  * cntxt.representations_in_context) > 0 THEN
  RETURN (TRUE);
  -- Determine the bag of representation_items that reference item.
  ELSE
   y := QUERY(z <* USEDIN (item , '') |
      'API_ABSTRACT_SCHEMA.REPRESENTATION_ITEM' IN TYPEOF(z));

   -- Ensure that the set is not empty.
   IF SIZEOF(y) > 0 THEN

   -- For each element in the set
   REPEAT i := 1 TO HIINDEX(y);

    -- check to see it is an item in the input cntxt.
    IF item_in_context(y[i], cntxt) THEN
     RETURN (TRUE);
    END_IF;
   END_REPEAT;
  END_IF;
 END_IF;
 -- Return false when all possible branches have been checked
 -- with no success.
 RETURN (FALSE);
END_FUNCTION;
(*
```

Argument definitions:

item: The **representation_item** checked for relationship in **cntxt**. This is input to the function.

cntxt: The **representation_context** for which the relationship to **item** is determined. This is input to the function.

6.3.3.3 Using_representations

The function **using_representations** returns the set of **representation**s in which a **representation_item** is used.

A **representation_item** is used in a **representation** if it is:

1) referenced in the set of **items** of the **representation**; or

2) referenced by a **representation_item** used in the **representation**.

NOTE - The second condition is a recursive check allowing for a **representation_item** to be used in a **representation** by being part of a tree of related **representation_item**s. The tree is rooted in an entity used in a **representation** by fulfilling the first condition.

EXPRESS specification:

```
*)
```

```
FUNCTION using_representations(item : representation_item)
 : SET OF representation;
 LOCAL
  results      : SET OF representation;
  result_bag     : BAG OF representation;
  intermediate_items : SET OF representation_item;
  i         : INTEGER;
 END_LOCAL;
 -- Find the representation in which the item is used and add to the
 -- results set.
 result_bag := USEDIN(item,'API_ABSTRACT_SCHEMA.REPRESENTATION.ITEMS');
 IF SIZEOF(result_bag) > 0 THEN
  REPEAT i := 1 TO HIINDEX(result_bag);
   results := results + result_bag[i];
  END_REPEAT;
 END_IF;
 -- Find the set of representation_items in which the item is used.
 intermediate_items := QUERY(z <* bag_to_set( USEDIN(item , '')) |
  'API_ABSTRACT_SCHEMA.REPRESENTATION_ITEM' IN TYPEOF(z));
 -- If the set of intermediate items is not empty;
 IF SIZEOF(intermediate_items) > 0 THEN
  -- For each element in the set recursively add the
  -- using_representations of that element.
  REPEAT i := 1 TO HIINDEX(intermediate_items);
   results := results + using_representations(intermediate_items[i]);
  END_REPEAT;
 END_IF;
 -- Return the set of representation in which the input item is used
 -- directly and indirectly (through intervening representation_items).
 RETURN (results);
END_FUNCTION;
(*
```

Argument definitions:

item: The **representation_item** for which using **representation**s are determined. This is input to the function.

6.3.4 API_ABSTRACT_SCHEMA function definition: api specific functions

This subclause declares the api-specific functions that are used in the **api_abstract_schema.**

6.3.4.1 Tree_api_group_structure

A **tree_api_group_structure** function determines whether or not the **api_group** structure whose root is the **group** argument is tree structured by the **api_group_assignment** relationship.

The function returns a value of TRUE if none of the **api_group**s that are assigned to the **group** argument , either directly or indirectly, are assigned twice. Otherwise it returns a value of FALSE.

The **tree_api_group_structure** function call the **assigned_api_group** function that recursively computes the **api_group**s assigned to the to the **group** argument.

EXPRESS specification:

```
*)
FUNCTION tree_api_group_structure(group : api_group) : BOOLEAN;
 LOCAL
  i       : INTEGER;
  j       : INTEGER;
  assigned_group : BAG [1:?] OF api_group;
 END_LOCAL;
 -- Determine the bag of the api_groups that are assigned to the
 -- group argument.
 assigned_group:= assigned_api_group (group);
 -- Determine that the bag has elements.
 IF SIZEOF(assigned_group) > 0 THEN
  -- Check each element of the bag against each element.
  REPEAT i := 1 TO HIINDEX(assigned_group);
   REPEAT j := 1 TO HIINDEX(assigned_group);
    -- If the two elements are the same
    -- return false.
    IF ((assigned_group[i] :=: assigned_group[j]) AND (i<> j)) THEN
     RETURN (FALSE);
    END_IF;
   END_REPEAT;
  END_REPEAT;
 END_IF;
 -- Return true when all elements are checked and
 -- no error conditions found.
 RETURN (TRUE);
END_FUNCTION;
(*
```

6.3.4.1.1 Assigned_api_group

The **assigned_api_group** function is used in a **tree_api_group_structure** function to computes recursively the **api_group**s assigned to the **group** argument.

EXPRESS specification:

```
*)
FUNCTION assigned_api_group( group : api_group) : BAG [0:?] OF api_group;
 LOCAL
  i            : INTEGER;
  assignment       : SET OF api_group_assignment;
  assigned_items     : BAG OF api_grouped_item;
  local_assigned_groups : BAG OF api_group;
  assigned_groups     : BAG OF api_group;
 END_LOCAL;
 assigned_items    := [];
 local_assigned_groups := [];
 -- Determine the subset of the api_group_assignments that assign items to
 -- the group argument .
 assignment := USEDIN (group, 'API_ABSTRACT_SCHEMA.API_GROUP_ASSIGNMENT\'
              + 'GROUP_ASSIGNMENT.ASSIGNED_GROUP');
 -- gathers all the api_groupeed_items
 REPEAT i := 1 TO HIINDEX(assignment);
  assigned_items := assigned_items + assignment[i].items;
 END_REPEAT;
 -- Determine the subset of api_groupeed_item that are api_groups.
 local_assigned_groups := QUERY( z <* assigned_items |
                'API_ABSTRACT_SCHEMA.API_GROUP' IN
                TYPEOF(z)
               );
 -- initializes the assigned_groups
 assigned_groups := local_assigned_groups;
 -- Determine that the subset has elements.
 IF SIZEOF(local_assigned_groups) > 0 THEN
  -- compute all the assigned api_group of the bag.
  REPEAT i := 1 TO HIINDEX(local_assigned_groups);
   assigned_groups := assigned_groups +
             assigned_api_group(local_assigned_groups[i]);
  END_REPEAT;
 END_IF;
RETURN (assigned_groups);
END_FUNCTION;
(*
```

6.3.4.2 Tree_api_set_structure

A **tree_api_set_structure** function determines whether or not the **api_set** structure whose root is the **set** argument is tree structured by the **api_set_assignment** relationship.

The function returns a value of TRUE if none of the **api_set**s that are assigned to the **set** argument , either directly or indirectly, are assigned twice. Otherwise it returns a value of FALSE.

The **tree_api_set_structure** function call the **assigned_api_set** function that recursively computes the **api_set**s assigned to the to the **the_set** argument.

EXPRESS specification:

```
*)
FUNCTION tree_api_set_structure(the_set : api_set) : BOOLEAN;
```

```
 LOCAL
  i      : INTEGER;
  j      : INTEGER;
  assigned_set : BAG [1:?] OF api_set;
 END_LOCAL;
 -- Determine the bag of the api_sets that are assigned to the set
 -- argument .
 assigned_set:= assigned_api_set(the_set);
 -- Determine that the bag has elements.
 IF SIZEOF(assigned_set) > 0 THEN
  -- Check each element of the bag against each element.
  REPEAT i := 1 TO HIINDEX(assigned_set);
   REPEAT j := 1 TO HIINDEX(assigned_set);
    -- If the two elements are the same
    -- return false.
    IF ((assigned_set[i] :=: assigned_set[j]) AND (i<> j)) THEN
     RETURN (FALSE);
    END_IF;
   END_REPEAT;
  END_REPEAT;
 END_IF;
 -- Return true when all elements are checked and
 -- no error conditions found.
 RETURN (TRUE);
END_FUNCTION;
(*
```

6.3.4.2.1 Assigned_api_set

The **assigned_api_set** function is used in a **tree_api_set structure function** to computes recursively the **api_set**s assigned to the **the_set** argument.

EXPRESS specification:

```
*)
FUNCTION assigned_api_set( the_set : api_set) : BAG [0:?] OF api_set;
 LOCAL
  i           : INTEGER;
  assignment      : SET OF api_set_assignment;
  assigned_items    : BAG OF api_set_item;
  local_assigned_sets : BAG OF api_set;
  assigned_sets     : BAG OF api_set;
 END_LOCAL;
 assigned_items     := [];
 local_assigned_sets := [];
 -- Determine the subset of the api_set_assignments that assign items to
 -- the set argument .
 assignment := USEDIN(the_set, 'API_ABSTRACT_SCHEMA.API_SET_ASSIGNMENT\'
              + 'GROUP_ASSIGNMENT.ASSIGNED_GROUP');
 -- gathers all the api_set_items
 REPEAT i := 1 TO HIINDEX(assignment);
  assigned_items := assigned_items + assignment[i].items;
 END_REPEAT;
 -- Determine the subset of api_set_item that are api_sets.
 local_assigned_sets := QUERY( z <* assigned_items |
               'API_ABSTRACT_SCHEMA.API_SET' IN
               TYPEOF(z)
              );
 -- initializes the assigned_sets
 assigned_sets := local_assigned_sets;
 -- Determine that the subset has elements.
 IF SIZEOF(local_assigned_sets) > 0 THEN
```

```
  -- compute all the assigned api_set of the bag.
  REPEAT i := 1 TO HIINDEX(local_assigned_sets);
   assigned_sets := assigned_sets +
            assigned_api_set(local_assigned_sets[i]);
  END_REPEAT;
 END_IF;
RETURN (assigned_sets );
END_FUNCTION;
(*
```

6.3.4.3 Api_legal_style_number

The function **api_legal_style_number** determines if the styles assigned to a **geometric_representation_item** generated by the interface are legal.

EXPRESS specification:

```
*)
FUNCTION api_legal_style_number(item: styled_item): BOOLEAN;
 LOCAL
  repr  : SET [0:?] of representation;
  nb_style: INTEGER;
 END_LOCAL;
 -- only one presentation_style_assignment
 IF SIZEOF(item.styles) > 1 THEN RETURN (FALSE); END_IF;
 -- one style is always allowed
 IF SIZEOF(item.styles[1].styles) = 1 THEN RETURN (TRUE); END_IF;
 -- identification of geometric space dimensionality
 repr:=USEDIN (item, 'API_ABSTRACT_SCHEMA.REPRESENTATION.ITEMS');
 -- only geometric_representation_item may be styled
 IF SIZEOF (repr) = 0 THEN RETURN (FALSE); END_IF;
 IF ( NOT ('API_ABSTRACT_SCHEMA.GEOMETRIC_REPRESENTATION_CONTEX' IN
      TYPEOF(repr[1].context_of_items) )
   )
 THEN
  RETURN (FALSE);
 END_IF;
 -- no hidden line in 3D
 IF dimension_of (item.item) = 3 THEN
  IF ( QUERY(temps <*item.styles[1].styles |
        ('API_ABSTRACT_SCHEMA.API_PREDEFINED_OCCLUSION_STYLE' IN
        TYPEOF (temps))
          OR
        ('API_ABSTRACT_SCHEMA.' +
        'API_PREDEFINED_VIRTUALLY_SENT_STYLE' IN
        TYPEOF(temps))
       ) <> []
    )
  THEN
   RETURN (FALSE); -- hidden line elimination is 2D
  END_IF;
  IF ( 'API_ABSTRACT_SCHEMA.ANNOTATION_FILL_AREA' IN
     TYPEOF (item.item)
    )
  THEN -- annotation fill area in 3D
   RETURN (
       NOT (SIZEOF(QUERY( f_a_style <* item.styles[1].styles |
                 'API_ABSTRACT_SCHEMA.FILL_AREA_STYLE' IN
                 TYPEOF(f_a_style)
                )
            ) <> SIZEOF(item.styles[1].styles) - 1
         )
```

```
             AND
        NOT (SIZEOF(QUERY( f_a_style <* item.styles[1].styles |
                   'API_ABSTRACT_SCHEMA.' +
                   'API_EXTERNALLY_DEFINED_FILL_AREA_STYLE'
                   IN TYPEOF(f_a_style)
                  )
              ) <> 1
          )
        );
   ELSE -- any other geometric_representation_item
    RETURN (SIZEOF(item.styles[1].styles) = 1);
   END_IF;
  END_IF; -- end 3D context
  -- case of 2D space
  nb_style := SIZEOF (item.styles[1].styles);
  IF ( SIZEOF(QUERY( st <* item.styles[1].styles |
            'API_ABSTRACT_SCHEMA.' +
            'API_PREDEFINED_VIRTUALLY_SENT_STYLE' IN
            TYPEOF(st)
           )
        ) = 1
    )
  THEN
   IF ( SIZEOF(QUERY( st <* item.styles[1].styles |
             'API_ABSTRACT_SCHEMA.' +
             'API_PREDEFINED_OCCLUSION_STYLE' IN
             TYPEOF(st)
            )
         ) = 1
     )
   THEN
    nb_style:=nb_style-2;
   ELSE
    RETURN (FALSE); -- virtually sent shall be bl involved
   END_IF;
  ELSE -- not virually sent
   IF ( SIZEOF(QUERY( st <* item.styles[1].styles |
             'API_ABSTRACT_SCHEMA.' +
             'API_PREDEFINED_OCCLUSION_STYLE' IN
             TYPEOF(st)
            )
         ) = 1
     )
   THEN
    nb_style:=nb_style-1;
   END_IF;
  END_IF;
  IF ( 'API_ABSTRACT_SCHEMA.ANNOTATION_FILL_AREA' IN
     TYPEOF (item.item)
    )
  THEN
   nb_style:=nb_style - SIZEOF(QUERY( f_a_style<*item.styles[1].styles |
                    'API_ABSTRACT_SCHEMA.'+
                    'FILL_AREA_STYLE' IN
                    TYPEOF(f_a_style)
                   )
                 );
  END_IF;
  RETURN (nb_style <= 1);
  END_FUNCTION;
 (*
```

6.4 API_ABSTRACT_SCHEMA global rules

This subclause declares the global rules that are associated with this entities that populate the **api_abstract_schema** and restrict their use or their relationships.

6.4.1 Unique_shape_representation

The **unique_shape_representation** rule requires there exist an unique **shape_representation** entity in the population of **api_abstract_schema.** This **shape_representation** corresponds to the shape of the product that is created through the interface in the CAD system database.

EXPRESS specification:

```
*)
RULE unique_shape_representation FOR (shape_representation);
WHERE
 WR1: SIZEOF( QUERY ( SHAPE <* shape_representation | TRUE)) =1;
END_RULE;
(*
```

EXPRESS specification:

```
*)
END_SCHEMA; -- end API_ABSTRACT schema
(*
```

7 Interface functional specification

NOTE - The logical description of the interface functions and their FORTRAN bindings are given in the normative annex, ANNEX A.

7.1 Notational conventions

7.1.1 Function representation

The heading of each function specifies:

I the function name,

II the minimal interface level for which the function is mandatory

III the minimal interface level for which the function is mandatory.

The parameter list indicates for each entry:

IV whether the entry is an input (in) or an output (out) parameter;

V the name of the parameter;

VI the data type itself, the short form notation for the type is explained in **7.1.2**

VII the meaning of the parameter;

VIII for entity name data, the permitted entity types (e.g. lin or basic,pnt);

either, for enumeration type data (e.g. [TDB,CAD]), the permitted values, or for double and integer data, any restriction on its range (e.g. (EPS ≤ X ≤ MAX));

The FORTRAN binding is specified by:

IX the FORTRAN syntax of FUNCTION - or SUBROUTINE call (the mapping between logical type and FORTRAN type is explained in annex **A** (see **A.2**);

X the effect of the function if none of the errors specified below is detected. For all but the inquire functions and the Reset_Error_State function, this effect is performed only if :

1) the interface is not in an error-state ;

2) no error is detected by the function ;

else the function returns to the application program, and in case 2, sets the *error_variable, error_text* and *error_origin* to the error values;

XI special NOTES, that are important for usage or implementation of function.

The internal reference is given to:

XII a reference to a clause or subclause of this ISO 15384 International Standard, where the concepts or domain entity definition for the function are presented.

The errors that are to be detected by the function, specified by:

XIII an error number or "–" for none error, and

XIV the accompanying error message.

The following example is provided as an illustration of the function representation:

EXAMPLE - Layout of a function representation:

<u>Function name:</u>

(I)

interface level:	(II)
geometrical power level:	(III)

<u>Parameters:</u>

in/ out	name	Data type	Meaning	permitted types/values
IV	**(V)**	**(IV)**	**(VI)**	**(VIII)**

<u>FORTRAN binding:</u>

(IX)

<u>Effects:</u>

(X)

<u>Notes:</u>

(XI)

<u>Internal reference:</u>

(XII)

Errors:

XIII	(XIV)

7.1.2 Data type representation

Data types in the definition of functions are either simple types or combinations of simple types.

Simple types are:

Table 16 — Simple data types

notation	data type	description
I	Integer	whole number
D	Double	biggest floating point number available in the language
N	*entity_name_type*	identification for entities
E	Enumeration	data type comprising an ordered set of values. The ordered set is defined by enumerating the identifiers that denote the values
S	String	a character sequence

A combination of simple types can be one of the following:

— a list of values of one simple type. For example:

nxI	list of integer
nxD	list of double
nxN	list of *entity_name_type*
nxS	list of string

The occurrence of an **n** here merely indicates a variable integer value.

Allowed value ranges or enumeration type values can be specified by :

— a condition. For example: > 0° or (EPS ≤ parameter ≤ MAX) ;

— a standard range of integer values. For example: (1..3) ;

— a range of integer values in which the maximum is determined by implementation or other constraints.

For example: (1..n) ;

— a list of values that constitute an enumeration type. For example: [TDB,CAD]

7.1.3 Entity names and abbreviations

To describe the permitted entity subtypes allowed for interface functions, the following short names are used.

Table 17 — Short names for entity types

Short name	meaning
a1p	Axis1_placement
a2p	Axis2_placement
Afa	Annotation_fill_area
Aps	Api_planar_surface
Arc	Api_circular_arc
Blk	Block
Brs	Boolean_result
Con	Right_circular_cone
Ctr	Api_contour
Cyl	Right_circular_cylinder
Dir	Direction
Eas	Extruded_area_solid
Elc	Api_elliptical_arc
Fsh	Fill_area_style_hatching
Grp	Api_group
Hss	Half_space_solid
Hyp	Api_hyperbolic_arc
lin	Api_line
Par	Api_parabolic_arc
Pln	Polyline
Pnt	Cartesian_point
Ras	Revolved_area_solid
Set	Set
Sph	Sphere
Tor	Torus
Wdg	Right_angular_wedge

Table 18 — Short names for collections of entity types

Short name	Meaning
Basic	an element of the collection (lin, arc)
Conic_arc	an element of the collection (elc, hyp, par)
Curves	an element of the collection (basic, conic_arc, general)
Csg	an element of the collection (blk, con, cyl, sph, tor, wdg, brs)
Fill_area	an element of the collection (afa, fsh)
General	an element of the collection (pln, ctr)
Math	an element of the collection (dir, a1p, a2p)
Solid_model	an element of the collection (csg, sweep)
Solids	an element of the collection (solid_model, hss)
Sweep	an element of the collection (eas, ras)

NOTE - All graphical entities' means all geometric representation items that may be created through the interface functions according Figure 2of this international standard.

7.1.4 Function names

The function names are build up by using the short names for the entity types, that are involved in the function, and sometimes an abbreviation (see **Table 19**). All parts of the function name begins with a capital letter and the parts of the name are broken by an underline character '_' .

For example:

— the function "Duplicate and Shift an Entity defined by a Direction', becomes the name Dup_Shift_Dir_Ent, by using the following abbreviations :

— Dup for duplicate

— Dir for an entity of type **direction**

— Ent for entity

Table 19 — Abbreviations used for function names

Abbreviation	**meaning**
Chg	change
Dup	duplicate
Ent	entity
Gen	generation
Inq	Inquire
Ref_Sys	Reference System
Sld	solid

7.2 Logical description of the interface functions and FORTRAN binding

The normative **Annex A** describes each of the interface functions and its binding for FORTRAN 90. Other language bindings will be specified as separate parts of this multi-part International Standard.

8 Interface tables

This clause specifies all entries from the interface tables and their default values.

8.1 Interface description table

The following table contains all entries, used in this International Standard to describe the capability of this interface. The values for this entries are implementation dependent.

Table 20 — Interface description table

Name	**type**	**meaning**
interface_level	I	Level : 1 to 3
hidden_line_capability	E	hidden line process available [OFF,ON]
max_interpolation_nodes_number	I	number of interpolation points for simulation (=1)
contour_entities	nxS	List of short names of entities permitted for api_contour besides the basic ones (at minimum 'lin' and 'arc')

8.2 Interface status table

The following table described all entries of the interface status table, used in this International Standard. These values are the default values, they may be changed by the initialisation process of a view, performed by the LMS.

Table 21 — Interface status table

Name	type	Value	meaning
error_variable	I	0	no error
error_origin	S	Empty	no error
error_text	S	Empty	no error
geometrical_power_level	I	1	2D view
hidden_line	E	see Table 20	equal to hidden_line_capability
hidden_line_involved	E	TRUE	hidden line involved [TRUE,FALSE]
interpolation_nodes_number	I	see Table 20	equal to max_interpolation_nodes_number
view_length_unit	E	METRE	metre [METRE,INCH]
view_length_scale_factor	D	10^{-3}	scale factor
view_angle_unit	E	DEG	degree [RAD,DEG,GRAD]
point_style	2xS	'asterisk_point', 'ISO_13584_31'	presentation style for points
curve_style	2xS	'plain_solid_line', 'ISO_13584_31'	presentation style for curves
fill_area_style	2xS	'opaque_fill_area', 'ISO_13584_31'	presentation style for fill_areas
surface_style	2xS	'solid_surface', 'ISO_13584_31'	presentation style for surfaces
hatch_curve_font	S	'continuous'	hatching curve font appearance
hatch_width	S	'thin_hatch_line'	hatching curve width appearance
hatch_colour	S	'hatch_line_colour'	hatching curve colour appearance
hidden_line_aspect	S	'hidden_line_invisible'	hidden line style for HLI entities
view_level	D	0.0	relative view level (virtual height)

9 Dimensions of interface implementation

9.1 Minimal dimensions of the different interface buffers and structured data types

This subclause specifies the minimal dimensions of the structured data types and the minimal capabilities of the data storage that shall be supported by an ISO 13584 conforming interface.

Table 22 — Dimensions of Interface implementation

kind of dimension	type	value
number of entities in the TDB	I	10000
number of points per polyline	I	300
number of entities per api_contour	I	300
number of inner boundaries per annotation_fill_area	I	100
number of inner boundaries per api_planar_surface	I	100
number of groups in the TDB	I	200
size of group stack	I	100
size of set stack	I	100
number of characters per string	I	256

Annex A
(normative)

Logical description of the interface functions and FORTRAN bindings

A.1 Introduction

This annex forms an integral part of the standard and is normative. It describes logically each interface function and specifies its FORTRAN binding.

A.2 FORTRAN mapping

A.2.1 Mapping for the interface function

— FORTRAN subroutines and FORTRAN functions correspond one-to-one to the logical functions. Each function has an unique FORTRAN subroutine or FORTRAN function name by which it shall be invoked. In this annex both the logical description and the FORTRAN binding of each interface function are specified.

— All FORTRAN function names and FORTRAN subroutine are max. 31 letters long. These are the same names as the logical functions in capital letters. For example, the logical function name **Dup_Shift_Dir_Ent** of the function 'Duplicate and Shift an Entity defined by a Direction', becomes the FORTRAN name `DUP_SHIFT_DIR_ENT`.

A.2.2 Mapping for the logical data type

Mapping the logical data types onto the FORTRAN data types:

Table A. 1 — Mapping of logical data types

logical data types	FORTRAN data representation
integer	INTEGER
list of integer	INTEGER array(length), where length is given by an INTEGER variable (e.g. N) or INTEGER constant
double	DOUBLE PRECISION
list of double	DOUBLE PRECISION array(length), where length is given by an INTEGER variable or INTEGER constant
enumeration	INTEGER, all values are mapped to the range zero to N-1, where N is the number of enumeration alternatives
entity_name_type	INTEGER; zero = 0; unknown = negative integer
list of entity_name_type	INTEGER array(length) for entity names, where length is given by an INTEGER variable or INTEGER constant
string	CHARACTER (LEN= *), containing the string
list of string	CHARACTER (LEN= *) array(length), where length is given by an INTEGER variable (N) or INTEGER constant

A.2.3 FORTRAN limitations for Parts Supplier Programs

A.2.3.1 Language basis

The basis of the syntax is the programming language FORTRAN (ISO 1539:1991(E)).

A.2.3.2 Excluded statements

The FORTRAN programs, within a part library, are intended to be run in various environments, on different CAD-systems and following various operating techniques (e.g. compiling and linkage, interpretation, translation...).

In order to ensure a maximal portability of these programs, the following FORTRAN statements are forbidden in part programs.

(1) Program architecture statements :

PROGRAM

ENTRY

STOP

BLOCK DATA

(2) Input/output statements :

READ,WRITE,FORMAT

OPEN,CLOSE,INQUIRE

REWIND,BACKSPACE,ENDFILE

(3) FORTRAN specific data organisation statements :

COMMON

EQUIVALENCE

DATA

SAVE

A.2.3.3 Obsolete features

The following features have been declared obsolescent in FORTRAN, i.e. they are still present, but will not appear in the next revision of the FORTRAN standard. Therefore the use of these features should be avoided in new programs, and eventually replaced in old programs.

Arithmetic-IF

Alternate-Return from subroutine

ASSIGN

Assigned FORMAT specifier

Assigned GOTO

DO loop control variables that are not integers

DO loop not ending on CONTINUE

Branch to END IF from outside IF block

H edit descriptor

PAUSE

A.2.3.4 Recommended statements

The use of implicit data typing should be avoided, therefore every program should contain the following directive:

IMPLICIT NONE

A.3 LIST OF INTERFACE FUNCTIONS

This subset of functions are the only interface functions that shall be provided for part supplier library programs and are specified by this normative annex**.**

A.3.1 List of interface functions according interface level 1

This following table shows a sub-set of interface functions that defines the complete sub-set of functions to be implemented for interface level 1 (see chapter **5.1.1** and Table 20 — Interface description table) and that shall be callable on an implementation conform to these interface level (1).

Table A. 2 — List of interface functions according interface level 1

chapter	function name	parameter	power
A.4.1	Data control functions		
A.4.1.1	Clear_Tdb	—	≥ 0
A.4.1.2	Fix_Ent	N,ENTLST	≥ 1
A.4.2	Error control functions		
A.4.2.1	Inq_Error_State	ERRNUM,ERRSRC,ERRTXT,ERR	≥ 0
A.4.2.2	Reset_Error_State	—	≥ 0
A.4.3	Interrogate interface capability functions		
A.4.3.1	Inq_Level	LEVEL,ERR	≥ 0
A.4.3.2	Inq_Hidden_Line_Capability	HLCAPA,ERR	≥ 0
A.4.3.3	Inq_Contour_Ent	N,TYPLST,ERR	≥ 0
A.4.3.4	Inq_Interface_Dimension	NUMLST,ERR	≥ 0
A.4.4.1	Inq_Hidden_Line	HIDMOD,ERR	≥ 0
A.4.4.2	Inq_Hidden_Line_Involvement	HLI,ERR	≥ 0
A.4.4	Interrogate interface sytem entry functions		
A.4.4.3	Inq_Interpolation_Nodes	NODENO,ERR	≥ 0
A.4.4.4	Inq_Geometrical_Power	POWER,ERR	≥ 0
A.4.4.5	Inq_Ovc_Unit	VLUNI,VLSFAC,VAUNI,ERR	≥ 0

Table A. 2 — (continued)

chapter	function name	parameter	power
A.4.5	Set interface sytem entry functions		
A.4.5.1	Set_Hidden_Line_Involvement	HLI	≥ 0
A.5.1.1	Direction		
A.5.1.1.1	Dir_Component	X,Y,Z,KFIX	≥ 1
A.5.1.1.2	Dir_2_Pnt	STAPNT,ENDPNT,KFIX	≥ 1
A.5.1.1.3	Dir_2_Dir_Angle	REFDIR,ZDIR,ANGLE,KFIX	≥ 1
A.5.1.1.4	Dir_A2p_X	A2PNAM,KFIX	≥ 1
A.5.1.1.5	Dir_A2p_Y	A2PNAM,KFIX	≥ 1
A.5.1.3	Axis2_placement (Local Coordinate System)		
A.5.1.3.1	A2p_3_Pnt	CENPNT,AXSPNT,REFPNT,KFIX	≥ 1
A.5.1.3.2	A2p_2_Dir	CENPNT,AXSDIR,REFDIR,KFIX	≥ 1
A.5.1.3.3	A2p_2_Dir_Xy	CENPNT,REFDIR,YAXDIR,KFIX	≥ 1
A.5.1.3.4	A2p_Position_Relative	REFLST,KFIX	≥ 1
A.5.1.3.5	A2p_Ref_Sys	KFIX	≥ 1
A.5.2.1	Points with canonical definition		
A.5.2.1.1	Pnt_Cartesian_Absolute	X,Y,Z,KFIX	≥ 1
A.5.2.1.2	Pnt_Cartesian_Relative	PNTNAM,DX,DY,DZ,KFIX	≥ 1
A.5.2.1.3	Pnt_Polar_Absolute	PHI,THETA,RAD,KFIX	≥ 1
A.5.2.1.4	Pnt_Polar_Relative	PNTNAM,PHI,THETA,RAD,KFIX	≥ 1
A.5.2.2	Points with constrained based definition		
A.5.2.2.1	Pnt_Begin_Ent	ENTNAM,KFIX	≥ 1
A.5.2.2.2	Pnt_End_Ent	ENTNAM,KFIX	≥ 1
A.5.2.2.3	Pnt_Intersection_2_Ent	ENTNM1,ENTNM2,KFIX	≥ 1
A.5.2.2.4	Pnt_Tangential_Arc	ARCNAM,LINNAM,KFIX	≥ 1
A.5.2.2.5	Pnt_Centre_Arc	ARCNAM,KFIX	≥ 1
A.5.2.2.6	Pnt_Middle_Ent	ENTNAM,KFIX	≥ 1
A.5.2.2.7	Pnt_Projection_Ent	PNTNAM,ENTNAM,KFIX	≥ 1
A.5.2.2.8	Pnt_Projection_A2p	PNTNAM,A2PNAM,KFIX	≥ 1
A.5.3.1.1	Line segments (api_line)		
A.5.3.1.1.1	Lin_2_Pnt	STAPNT,ENDPNT,KFIX	≥ 1
A.5.3.1.1.2	Lin_Pnt_Length_Dir	STAPNT,LEN,DIRNAM,KFIX	≥ 1
A.5.3.1.1.3	Lin_Tangential_Arc	STAPNT,ARCNAM,KFIX	≥ 1
A.5.3.1.1.4	Lin_Tangential_2_Arc	ARCNM1,ARCNM2,KFIX	≥ 1
A.5.3.1.1.5	Lin_Chamfer_2_Lin	LEN1,LEN2,LINNM1,LINNM2,KFIX	≥ 1
A.5.3.1.2	Circle and circular arc (api_circular_arc)		
A.5.3.1.2.1	Circle_Rad_A2p	RAD,A2PNAM,SENSE,KFIX	≥ 1
A.5.3.1.2.2	Arc_3_Pnt	STAPNT,INTPNT,ENDPNT,KFIX	≥ 1
A.5.3.1.2.3	Arc_Rad_2_Angle_A2p	RAD,STAANG,ENDANG,A2PNAM, SENSE,KFIX	≥ 1
A.5.3.1.2.4	Arc_Rad_3_Pnt	RAD,STAPNT,ENDPNT,HLPPNT,KFIX	≥ 1
A.5.3.1.2.5	Arc_Rad_2_Pnt_A2p	RAD,PNTNM1,PNTNM2,A2PNAM, SENSE,KFIX	≥ 1

Table A. 2 — (continued)

chapter	function name	parameter	power
A.5.3.1.2.6	Arc_Fillet_2_Ent	ENTNM1,ENTNM2,RAD,KFIX	≥ 1
A.5.3.1.2.7	Arc_Tangential_2_Ent	ENTNM1,ENTNM2,RAD,KFIX	≥ 1
A.5.3.1.2.8	Arc_Rad_2_Ent	RAD,ENTNM1,ENTNM2,IN1,IN2, MINLEN,KFIX	≥ 1
A.5.3.1.2.9	Arc_3_Ent	ENTNM1,ENTNM2,ENTNM3,IN1,IN2,IN3, KFIX	≥ 1
A.5.3.2.1	Ellipse and elliptical arc (api_elliptical_arc)		
A.5.3.2.1.1	Ellipse_2_Diameter_A2p	SEMI1,SEMI2,A2PNAM,SENSE,KFIX	≥ 1
A.5.3.2.1.2	Elc_Gen	SEMI1,SEMI2,STAANG,ENDANG,A2PNAM ,SENSE,KFIX	≥ 1
A.5.3.2.2	Hyperbolical arc (api_hyperbolic_arc)		
A.5.3.2.2.1	Hyp_Gen	SEMAXI,SEMIMG,STAANG,ENDANG, A2PNAM,KFIX	≥ 1
A.5.3.2.2	Hyperbolical arc (api_parabolic_arc)		
A.5.3.2.3.1	Par_Gen	FOCLEN,STAANG,ENDANG,A2PNAM, KFIX	≥ 1
A.5.3.3.1	Polyline		
A.5.3.3.1.1	Pln_Cartesian_Coordinate	N,XLST,YLST,ZLST,KFIX	≥ 1
A.5.3.3.1.2	Pln_Pnt_List	N,PNTLST,KFIX	≥ 1
A.5.3.3.2	Planar contour (api_contour)		
A.5.3.3.2.1	Ctr_Gen	N,ENTLST,KFIX	≥ 1
A.5.4	Fill area		
A.5.4.1	Afa_Gen	CTRNAM,N,CTRLST,KFIX	= 1
A.5.4.2	Fsh_Gen	REFPNT,DIST1,DIST2,ANGLE	= 1
A.5.4.3	Hatch_Afa	HATCH,AFA,REFPNT	= 1
A.6.1	**Structure entities in the TDB**		
A.6.1.1	Create_Grp	—	≥ 1
A.6.1.2	Close_Grp	—	≥ 1
A.6.1.3	Reopen_Grp	GRPNAM	≥ 1
A.6.1.4	Remove_Ent_Grp	ENTNAM	≥ 1
A.6.1.5	Gather_Ent_Grp	N,ENTLST	≥ 1
A.6.1.6	Add_Ent_Grp	GRPNAM,ENTNAM	≥ 1
A.6.2	**Structure entities to sent into CAD sysem**		
A.6.2.1	Open_Set	SETNAM	≥ 1
A.6.2.2	Close_Set		≥ 1
A.7.1	**Duplicating entities**		
A.7.1.1	Dup_Ent	ENTNAM,KFIX	≥ 1
A.7.2	**Mirroring entities**		
A.7.2.1	Mirror_Ent	ENTNAM,LINNAM	≥ 1
A.7.2.2	Dup_Mirror_Ent	ENTNAM,LINNAM,KFIX	≥ 1

Table A. 2 — (continued)

chapter	function name	parameter	power
A.7.3	**Shifting entities**		
A.7.3.1	Shift_Dir_Ent	ENTNAM,DIRNAM,SHFLEN	≥ 1
A.7.3.2	Shift_Displacement_Ent	ENTNAM,DX,DY,DZ	≥ 1
A.7.3.3	Dup_Shift_Dir_Ent	ENTNAM,DIRNAM,SHFLEN,KFIX	≥ 1
A.7.3.4	Dup_Shift_Displacement_Ent	ENTNAM,DX,DY,DZ,KFIX	≥ 1
A.7.4	**Rotating entities**		
A.7.4.1	Rotate_Ent	ENTNAM,PNTNAM,ANG1,ANG2,ANG3	≥ 1
A.7.4.2	Dup_Rotate_Ent	ENTNAM,PNTNAM,ANG1,ANG2,ANG3, KFIX	≥ 1
A.7.5	**Changing entities**		
A.7.5.1	Chg_Orientation_Ent	ENTNAM	≥ 1
A.7.5.2	Chg_Sense_Ent	ENTNAM	≥ 1
chapter	**function name**	**parameter**	**power**
A.7.5.3	Homotetia_Ent	ENTNAM,PNTNAM,K	≥ 1
A.8.1	Utility functions for geometric entities		
A.8.1.1	Pnt_Retrieve_Coordinate	PNTNAM,X,Y,Z	≥ 1
A.8.1.2	Pnt_Retrieve_Component	DIRNAM,X,Y,Z	≥ 1
A.8.1.3	A2p_Retrieve_Location	A2PNAM,PNTNAM	≥ 1
A.8.1.4	Lin_Retrieve_Dir	LINNAM,DIRNAM	≥ 1
A.8.1.6	Arc_Retrieve_A2p	ARCNAM,A2PNAM	≥ 1
A.8.1.7	Arc_Retrieve_Rad	ARCNAM,RADIUS	≥ 1
A.8.1.8	Arc_Retrieve_Sense	ARCNAM,SENSE	≥ 1
A.8.2	Interrogate entity functions		
A.8.2.1	Retrieve_Type_Ent	ENTNAM,TYPE	≥ 1
A.8.2.2	Retrieve_Member_Grp	GRPNAM,N,ENTLST[3)]	≥ 1
A.8.2.3	Retrieve_Ent_Ctr	CTRNAM,N,ENTLST[6]	≥ 1
A.8.3	Calculation utility functions		
A.8.3.1	Distance_2_Pnt	PNTNM1,PNTNM2	≥ 1
A.8.3.2	Start_Angle_Arc	ARCNAM	≥ 1
A.8.3.3	End_Angle_Arc	ARCNAM	≥ 1
A.9.1	Generation and setting of nnew reference system		
A.9.1.1	Ref_Sys_3_Pnt	CENPNT,AXSPNT,REFPNT	≥ 1
A.9.1.2	Ref_Sys_2_Dir	CENPNT,AXSDIR,REFDIR	≥ 1
A.9.1.3	Ref_Sys_2_Dir_Xy	CENPNT,REFDIR,YAXDIR	≥ 1
A.9.1.4	Ref_Sys_Position_Relative	REFLST	≥ 1
A.9.1.5	Ref_Sys_A2p	A2PNAM	≥ 1

3) additional parameter DIMLST for FORTRAN binding

Table A. 2 — (continued)

chapter	**function name**	**parameter**	**power**
A.10.1	Setting of global entries for visualisation attributes		
A.10.1.1	Set_Point_Style	EXTSOU,PNTSTY	≥ 0
A.10.1.2	Set_Curve_Style	EXTSOU,CURSTY	≥ 0
A.10.1.3	Set_Fill_Area_Style	EXTSOU,AFASTY	≥ 0
A.10.1.4	Set_Surface_Style	EXTSOU,SURSTY	≥ 0
A.10.1.5	Set_Hatch_Width	WIDTH,ERR	≥ 0
A.10.1.6	Set_Hatch_Curve_Font	FONT	≥ 0
A.10.1.7	Set_Hatch_Colour	COLOUR	≥ 0
A.10.1.8	Set_Hidden_Line_Aspect	HIDSTY	≥ 0
A.10.1.9	Set_Relative_View_Level	RVL	≥ 0
A.10.2	Inquire of global entries for visualisation attributes		
A.10.2.1	Inq_Point_Style	EXTSOU,PNTSTY,ERR	≥ 0
A.10.2.2	Inq_Curve_Style	EXTSOU,CURSTY,ERR	≥ 0
A.10.2.3	Inq_Fill_Area_Style	EXTSOU,AFASTY,ERR	≥ 0
A.10.2.4	Inq_Surface_Style	EXTSOU,SURSTY,ERR	≥ 0
chapter	**function name**	**parameter**	**power**
A.10.2.5	Inq_Hatch_Width	WIDTH,ERR	≥ 0
A.10.2.6	Inq_Hatch_Curve_Font	FONT,ERR	≥ 0
A.10.2.7	Inq_Hatch_Colour	COLOUR,ERR	≥ 0
A.10.2.8	Inq_Hidden_Line_Aspect	HIDSTY,ERR	≥ 0
A.10.2.9	Inq_Relative_View_Level	RVL,ERR	≥ 0
A.10.3	Changing the visual appearance of entities		
A.10.3.1	Chg_Point_Style	PNTNAM,EXTSOU,PNTSTY	≥ 1
A.10.3.2	Chg_Curve_Style	ENTNAM,EXTSOU,CURSTY	≥ 1
A.10.3.3	Chg_Fill_Area_Style	AFANAM,EXTSOU,AFASTY	≥ 1
A.10.3.4	Chg_Surface_Style	ENTNAM,EXTSOU,SURSTY	≥ 1
A.10.3.5	Chg_Hatch_Width	FSHNAM,WIDTH	≥ 1
A.10.3.6	Chg_Hatch_Curve_Font	FSHNAM,FONT	≥ 1
A.10.3.7	Chg_Hatch_Colour	FSHNAM,COLOUR	≥ 1
A.10.3.8	Chg_Hidden_Line_Aspect	ENTNAM,HIDSTY	≥ 1
A.10.3.9	Chg_Relative_View_Level	ENTNAM,RVL	≥ 1
A.10.4	**Retrieve assigned style from entities**		
A.10.4.1	Retrieve_Point_Style	PNTNAM,EXTSOU,PNTSTY	≥ 1
A.10.4.2	Retrieve_Curve_Style	ENTNAM,EXTSOU,CURSTY	≥ 1
A.10.4.3	Retrieve_Fill_Area_Style	AFANAM,EXTSOU,AFASTY	≥ 1
A.10.4.4	Retrieve_Surface_Style	ENTNAM,EXTSOU,SURSTY	≥ 1
A.10.4.5	Retrieve_Hatch_Width	FSHNAM,WIDTH	≥ 1
A.10.4.6	Retrieve_Hatch_Curve_Font	FSHNAM,FONT	≥ 1
A.10.4.7	Retrieve_Hatch_Colour	FSHNAM,COLOUR	≥ 1
A.10.4.8	Retrieve_Hidden_Line_Aspect	ENTNAM,HIDSTY	≥ 1
A.10.4.9	Retrieve_Relative_View_Level	ENTNAM,RVL	≥ 1

A.3.2 List of interface functions according interface level 2

This following table shows a sub-set of interface functions that defines the complete sub-set of functions to be implemented as ‘ add on’ for interface level 2 (see chapter **5.1.1** and Table 20 — Interface description table). These functions shall be callable on an implementation conform to these interface level (2), additionally to the functions implemented for interface level 1.

Table A. 3 — List of interface functions according interface level 2

chapter	function name	parameter	power
A.5.1.1	Direction		
A.5.1.1.6	DIR_A2P_Z	A2PNAM,KFIX	≥ 2
A.5.1.2	Axis1_placement (single axis)		
A.5.1.2.1	A1P_GEN	PNTNAM,DIRNAM,KFIX	≥ 2
A.5.1.2.2	A1P_2_PNT	STAPNT,ENDPNT,KFIX	≥ 2
A.5.2.1	Points with canonical definition		
A.5.2.1.5	PNT_CYLINDER_ABSOLUTE	PHI,RAD,HEIGHT,KFIX	≥ 2
A.5.2.1.6	PNT_CYLINDER_RELATIVE	PNTNAM,PHI,RAD,HEIGHT,KFIX	≥ 2
A.5.5	Surface entities		
A.5.5.1	APS_GEN	CTRNAM,N,CTRLST,KFIX	≥ 2

A.3.3 List of interface functions according interface level 3

This following table shows a sub-set of interface functions that defines the complete sub-set of functions to be implemented as ‘ add on’ for interface level 3 (see chapter **5.1.1** and Table 20 — Interface description table). These functions shall be callable on an implementation conform to these interface level (3), additionally to the functions implemented for interface level 1 and 2.

Table A. 4 — List of interface functions according interface level 3

chapter	function name	parameter	power
A.5.6.1	CSG primitives		
A.5.6.1.1	Sph_Gen	RAD,CNTPNT,KFIX	= 3
A.5.6.1.2	Con_Gen	ANGLE,HEIGHT,RAD,A1PNAM,KFIX	= 3
A.5.6.1.3	Cyl_Gen	RAD,HEIGHT,A1PNAM,KFIX	= 3
A.5.6.1.4	Tor_Gen	MAJOR,MINOR,A1PNAM,KFIX	= 3
A.5.6.1.5	Blk_Gen	LENX,LENY,LENZ,A2PNAM,KFIX	= 3
A.5.6.1.6	Wdg_Gen	LENX,LENY,LENZ,LTX,A2PNAM,KFIX	= 3
A.5.6.2	CSG regular Boolean operations		
A.5.6.2.1	Union_Sld	BOPNM1,BOPNM2,KFIX	= 3
A.5.6.2.2	Intersection_Sld	BOPNM1,BOPNM2,KFIX	= 3
A.5.6.2.3	Difference_Sld	BOPNM1,BOPNM2,KFIX	= 3
A.5.6.3	sweept_area solids		
A.5.6.3.1	Sld_Extrusion	SRFNAM,DIRNAM,DEPTH,KFIX	= 3
A.5.6.3.2	Sld_Revolution	SRFNAM,ANG,A1PNAM,KFIX	= 3
A.5.6.4	CSG solid pipe entity		
A.5.6.4.1	Sld_Pipe	PLNNAM,SRFNAM,RAD,KFIX	= 3
A.5.6.5	Half space solid entity		
A.5.6.5.1	Hss_Gen	A2PNAM,AGREMF	= 3
A.8.1	Utility functions for geometric entities		
A.8.1.5	Hss_Retrieve_A2p	HSSNAM,A2PNAM	= 3

A.4 INTERFACE CONTROL FUNCTIONS

These functions allow the application program to control the interface (e.g. to send geometrical data to CAD system), to control the error state, and to interrogate entries from the interface tables.

A.4.1 Data control functions

Clear temporary database	Clear_TDB
Fix entities into CAD system	Fix_Ent

A.4.1.1 Clear temporary database

Function name:

Clear_TDB

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
–				

FORTRAN binding:

```
CALL CLEAR_TDB ( )
```

Effects:

The temporary database (TDB) shall be cleared. If an error occurs, no modification of the TDB will be done.

1) If there is an open 2D view and there are entities within the TDB that are virtually sent (assigned with the **api_pre_defined_virtually_sent_style**), these entities shall not be deleted. They remain in the TDB until the hidden line removal process is finished.

Notes:

Internal reference:

5.3.4, 5.3.5, 6.2.5.5

Errors:

–	None		

A.4.1.2 Fix entities into CAD system

Function name:

Fix_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	N	I	length of ENTLST	≥ 1
in	ENTLST	nxN	list of entity names	(pnt,curves,afa,aps,math, solid_model,grp)

FORTRAN binding:

```
CALL FIX_ENT ( N,ENTLST )
```

Effects:

All entities of the given list ENTLST are removed from the group structure they belong to, and are sent from TDB to the CAD system. The access to these entities are no longer available and the names of these entities become unknown. If an error occurs, no modification will be done.

Notes:

1) If the effected entity is an instance of type **api_group**, this group shall be a closed group and all entities referring to this group are sent to the CAD system. The name of this group becomes unknown just like the names of refered entities.

2) If there is an open 2D view and the *hidden_line* entry in the interface status table is equal to ON, all entities of ENTLST that are hidden line involved (HLI) are only virtually sent. That means, these entities becomes an **api_pre_defined_virtually_sent_style**attached and are 'virtually sent'. They remain within the temporary database (TDB) until they shall be send to the CAD system after the hidden line removal process is finished.

3) When an entity is virtually sent, it shall be removed from the **api_group** structure and shall belong to the root group that cannot be used in any group functions.

Internal reference:

5.3.4, 5.3.5, **5.5**, 6.1.19, 6.2.5.5

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
5	integer value out of permitted range	202	error while sending entity to CAD system
204	function not compatible with current power level		

A.4.2 Error control functions

Inquire error state	Inq_Error_State
Reset system error	Reset_Error_State

A.4.2.1 Inquire error state

Function name:

Inq_Error_State

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
out	ERRNUM	I	current error value	0 or *error_variable*
out	ERRSRC	S	name of function that caused the error	
out	ERRTXT	S	message text associated with the error	
out	ERR	E	*error_indicator* for inquire functions	[NOERROR,ERROR]

FORTRAN binding:

```
CALL INQ_ERROR_STATE ( ERRNUM,ERRSRC,ERRTXT,ERR )
```

Effects:
This function provides the current *error_status* of the interface by returning the current settings of *error_variable*, *error_origin* and *error_text*.

If the interface is in an error state (*error_state* = true):

— the current value error_variable and the current strings error_origin and error_text of the interface status table returned for ERRNUM, ERRSRC and ERRTXT respectively.

— the value of ERRNUM is equal to zero and the strings of ERRSRC and ERRTXT are empty strings.

If the interface is in a non error state (*error_state* = false):

The *error_indicator* ERR, reports any difficulty during function performance.

Notes:
–

Internal reference:
5.8

Errors:

–	None		

A.4.2.2 Reset error state

Function name:
Reset_Error_State

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
–				

FORTRAN binding:

```
CALL RESET_ERROR_STATE ( )
```

Effects:
This function resets the current error state of the interface.

If the interface is in an error state (*error_state* = true):

— exits the interface from error state and resets the current value *error_variable* and the current strings *error_origin* and *error_text* of the interface status table, by setting zero for *error_variable* and an empty string for *error_origin* and *error_text* .

If the interface is not in an error state (*error_state* = false):

— returns and no modification will be done.

Notes:
–

Internal reference:
5.8

Errors:

–	None		

A.4.3 Interrogate interface capability functions

Inquire interface level	Inq_Level
Inquire hidden line capability	Inq_Hidden_Line_Capability
Inquire contour entity	Inq_Contour_Ent
Inquire interface dimension	Inq_Interface_Dimension

A.4.3.1 Inquire interface level

Function name:
Inq_Level

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
out	LEVEL	I	value for*interface_level* entry	(1...3)
out	ERR	E	*error_indicator* for inquire functions	[NOERROR,ERROR]

FORTRAN binding:

```
CALL INQ_LEVEL ( LEVEL,ERR )
```

Effects:
This function provides the value of the *interface_level* entry from the interface description table. The *error_indicator* ERR, reports any difficulty during function performance.

Notes:
–

Internal reference:
5.1.1, **8.1**

Errors:

–	None		

A.4.3.2 Inquire hidden line capability

Function name:
Inq_Hidden_Line_Capability

interface level:	**1**

geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
out	HLCAPA	E	value for *hidden_line_capability* entry	[OFF,ON]
out	ERR	E	*error_indicator* for inquire functions	[NOERROR,ERROR]

FORTRAN binding:

```
CALL INQ_HIDDEN_LINE_CAPABILITY ( HLCAPA,ERR )
```

Effects:

This function provides the value of the *hidden_line_capability* entry from the interface description table. The *error_indicator* ERR, reports any difficulty during function performance.

Notes:

–

Internal reference:

5.3.5, **8.1**

Errors:

–	None		

A.4.3.3 Inquire contour entity

Function name:

Inq_Contour_Ent

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
out	N	I	length of TYPLST, that defines the number of entries	(2...6)
out	TYPLST	nxS	list of names for short entity types, that are allowed for **api_contour**	('lin', 'arc', 'elc', 'par', 'hyp', 'pln')
out	ERR	E	*error_indicator* for inquire functions	[NOERROR,ERROR]

FORTRAN binding:

```
CALL INQ_CONTOUR_ENT ( N,TYPLST,ERR )
```

Effects:

This function provides a list of short names of entity types, that are permitted for contour representation, as *contour_entities* entry from the interface description table, added with the two defaults. The minimal number of entity types in TYPLST must be 2 ('lin' for **api_line** and 'arc' for **api_circular_arc,** that are always be presented in every interface implementation). The *error_indicator* ERR, reports any difficulty during function performance.

Notes:

1) The number of character per list element TYPLST(i) shall be equal to 3.

Internal reference:

6.1.14.2, 7.1.3, **8.1**

Errors:

–	None		

A.4.3.4 Inquire interface dimension

Function name:

Inq_Interface_Dimension

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
out	NUMLST	9xI	list of minimal dimensions of interface buffers	see below
out	ERR	E	*error_indicator* for inquire functions	[NOERROR,ERROR]

FORTRAN binding:

```
CALL INQ_INTERFACE_DIMENSION ( NUMLST,ERR )
```

Effects:

This function provides a list of numbers that specifies the minimal dimensions of interface buffers, according Table 21 of this International Standard. The list elements are:

NUMLST(1)	–	number of entities in the TDB
NUMLST(2)	–	number of points per polyline
NUMLST(3)	–	number of entities per api_contour
NUMLST(4)	–	number of inner boundaries per annotation_fill_area
NUMLST(5)	–	number of inner boundaries per api_planar_surface
NUMLST(6)	–	number of groups in the TDB
NUMLST(7)	–	size of group stack
NUMLST(8)	–	size of set stack
NUMLST(9)	–	number of characters per string

The *error_indicator* ERR, reports any difficulty during function performance.

Notes:

–

Internal reference:

9.1

Errors:

–	None		

A.4.4 Interrogate interface system entry functions

Inquire hidden line	Inq_Hidden_Line
Inquire hidden line involvement	Inq_Hidden_Line_Involvement

Inquire interpolation nodes number	Inq_Interpolation_Nodes
Inquire geometrical power	Inq_Geometrical_Power
Inquire OVC units	Inq_OVC_Unit

A.4.4.1 Inquire hidden line

Function name:

Inq_Hidden_Line

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
out	HIDMOD	E	current value for *hidden_line* entry	[ON,OFF]
out	ERR	E	*error_indicator* for inquire functions	[NOERROR,ERROR]

FORTRAN binding:

```
CALL INQ_HIDDEN_LINE ( HIDMOD,ERR )
```

Effects:

This function provides the current values for *hidden_line* entry from the interface status table. The *error_indicator* ERR, reports any difficulty during function performance.

Notes:

–

Internal reference:

5.3.5, 8.2

Errors:

–	None		

A.4.4.2 Inquire hidden line involvement

Function name:

Inq_Hidden_Line_involvement

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
out	HLI	E	current value for *hidden_line_involved* entry	[TRUE,FALSE]
out	ERR	E	*error_indicator* for inquire functions	[NOERROR,ERROR]

FORTRAN binding:

```
CALL INQ_HIDDEN_LINE_INVOLVEMENT ( HLI,ERR )
```

Effects:

This function provides the current values for *hidden_line_involved* entry from the interface status table. The *error_indicator* ERR, reports any difficulty during function performance.

Notes:

–

Internal reference:
5.3.5, 8.2

Errors:

–	None		

A.4.4.3 Inquire interpolation nodes number

Function name:
Inq_Interpolation_Nodes

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
out	NODENO	I	current value for *interpolation_nodes_number* entry	≥ 1
out	ERR	E	*error_indicator* for inquire functions	[NOERROR,ERROR]

FORTRAN binding:

```
CALL INQ_INTERPOLATION_NODES ( NODENO,ERR )
```

Effects:
This function provides the implemented and available value of the *interpolation_nodes_number* entry from the interface status table. The *error_indicator* ERR, reports any difficulty during function performance.

Notes:
–

Internal reference:
6.1.13, **8.1**

Errors:

–	None		

A.4.4.4 Inquire geometrical power

Function name:
Inq_Geometrical_Power

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
out	POWER	I	current value for *geometrical_power_level* entry	(0...3)
out	ERR	E	*error_indicator* for inquire functions	[NOERROR,ERROR]

FORTRAN binding:

```
CALL INQ_GEOMETRICAL_POWER ( POWER,ERR )
```

Effects:
This function provides the current values for *geometrical_power_level* entry from the interface status table. The *error_indicator* ERR, reports any difficulty during function performance.

Notes:
–

Internal reference:
5.1.1, 8.2

Errors:

–	None		

A.4.4.5 Inquire OVC units

Function name:
Inq_OVC_Unit

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
out	VLUNI	E	current value for *view_length_unit* entry	[METRE,INCH]
out	VLSFAC	D	current value for *view_length_scale_factor* entry	
out	VAUNI	E	current value for *view_angle_unit* entry	[RAD,DEG,GRAD]
out	ERR	E	*error_indicator* for inquire functions	[NOERROR,ERROR]

FORTRAN binding:

```
CALL INQ_OVC_UNIT ( VLUNI,VLSFAC,VAUNI,ERR )
```

Effects:
This function provides the settings for *OVC_length_unit* and *OVC_angle_unit* by returning the current values for *view_length_unit*, *view_length_scale_factor* and *view_angle_unit* entries from the interface status table.

OVC_length_unit refers to *view_length_unit* scaled by *view_length_scale_factor* and *OVC_angle_unit* refers to *view_angle_unit* .

The *error_indicator* ERR, reports any difficulty during function performance.

Notes:
–

Internal reference:
5.3.2, **8.2**

Errors:

–	None		

A.4.5 Set interface system entry functions

Set hidden line involvement	Set_Hidden_Line_Involvement

A.4.5.1 Set hidden line involvement

Function name:

Set_Hidden_Line_Involvement

interface level:	**1**
geometrical power level:	**0,1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	HLI	E	value for *hidden_line_involved* entry	[TRUE,FALSE]

FORTRAN binding:

```
CALL SET_HIDDEN_LINE_INVOLVEMENT ( HLI )
```

Effects:

Sets the new current value for *hidden_line_involved* entry from the interface status table, defined by the given value HLI equals TRUE. Each curve and fill area entity created through an interface function shall be involved in the hidden line removal process; or HLI equals FALSE, that means that these created entities are not involved. If an error occurs, no modification shall be done.

Notes:

–

Internal reference:

5.3.5, **8.2**

Errors:

1001	enumerated value out of range		

A.5 FUNCTIONS FOR GEOMETRIC DATA

A.5.1 Mathematical entities

A.5.1.1 Direction

Direction vector defined by components	Dir_Component
Direction vector defined by two points	Dir_2_Pnt
Direction vector defined by two directions and angle	Dir_2_Dir_Angle
X direction from an axis2_placement	Dir_A2p_X
Y direction from an axis2_placement	Dir_A2p_Y
Z direction from an axis2_placement	Dir_A2p_Z

A.5.1.1.1 Direction vector defined by components

Function name:

Dir_Component

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	X	D	x component, in (Ox) direction of the current OVC	see NOTE (1)
in	Y	D	y component, in (Oy) direction of the current OVC	see NOTE (1)
in	Z	D	z component, in (Oz) direction of the current OVC	see NOTE (1)
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **direction**	dir

FORTRAN binding:

```
NAME = DIR_COMPONENT ( X,Y,Z,KFIX )
```

Effects:

Creates a **direction** entity with **direction_ratio**s **x**,**y** and **z** derived from the given parameters X, Y and Z respectively. The name of the created **direction** is returned. This **direction** has a **null_style** assigned to it.

The magnitude of the direction vector shall be in range [EPS,MAX]. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) No value of component, neither for |X| nor for |Y| and nor for |Z|, shall be between ZERO_VALUE and EPS.

2) When the current open view is defined as a 2D view (the *geometrical_power_level* entry of the interface status table equals 1), the value for **z** will subsequently be ignored by the interface.

Internal reference:

6.1.9.3, 6.2.1.2

Errors:

7	real value out of permitted range	101	attempt to create a degenerated entity
102	magnitude of direction vector out of range [EPS,MAX]	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.1.1.2 Direction vector defined by two points

Function name:

Dir_2_Pnt

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	STAPNT	N	name of start **cartesian_point**	pnt
in	ENDPNT	N	name of end **cartesian_point**	pnt
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **direction**	dir

FORTRAN binding:

```
NAME = DIR_2_PNT ( STAPNT,ENDPNT,KFIX )
```

Effects:

Creates a **direction** entity with **direction_ratio**s **x**,**y** and **z**.

In the following, we will consider **P1** and **P2** as synonym for the two given **cartesian_point**s STAPNT

— the difference **P2-P1** is calculated and the three derived components are stored in the **direction_ratio**s **x**, **y** and **z**. The name of the created **direction** is returned. This **direction** has a **null_style** assigned to it.

and ENDPNT, respectively. Then:

The distance between the two **cartesian_point**s shall be in range [EPS,MAX]. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

When the current open view is defined as a 2D view (that means that the *geometrical_power_level* entry of the interface status table equals 1), the value for **z** will subsequently be ignored by the interface.

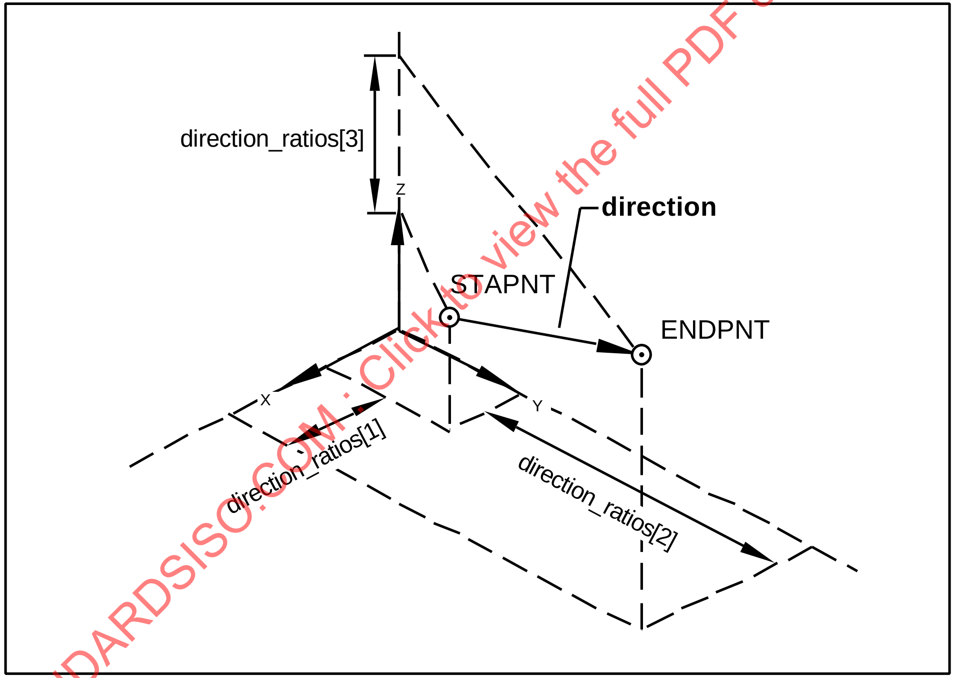


Figure A. 1 — Function: Dir_2_Pnt

Internal reference:

6.1.9, **6.1.9.3**, 6.2.1.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
103	distance between two points out of range [EPS,MAX]	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.1.1.3 Direction vector defined by two directions and an angle

Function name:

Dir_2_Dir_Angle

interface level:	1
geometrical power level:	1,2,3

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	REFDIR	N	name of reference **direction**	dir
in	ZDIR	N	name of z-**direction**	dir
in	ANGLE	D	angle value	$(0° \leq \text{ANGLE} \leq 360°)$
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **direction**	dir

FORTRAN binding:

```
NAME = DIR_2_DIR_ANGLE ( REFDIR,ZDIR,ANGLE,KFIX )
```

Effects:

Creates a normalised **direction** entity with **direction_ratio**s **x**,**y** and **z**, derived from a calculation process dependent on the current view initialisation.

— a NAME normalised **direction** entity is created, such that the value of the oriented angle from REFDIR to NAME is ANGLE.

In the case of an open 2D view:

In the case of an open 3D view:

— let **P** be the plane normal to the ZDIR **direction**. Then a normalised NAME **direction** entity is created, such that the value of the oriented angle from the orthogonal projection of REFDIR onto **P** to NAME is ANGLE.

In both cases, the name of the created **direction** is returned. This **direction** has a **null_style** assigned to it.

If the given value of ANGLE that is measured in *OVC_angle_units*, is less than ZERO_VALUE, then the created **direction** entity NAME is equal to REFDIR. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

When the current open view is defined as a 2D view (that means that the *geometrical_power_level* entry of the interface status table equals 1), the given parameter ZDIR will subsequently be ignored by the interface.

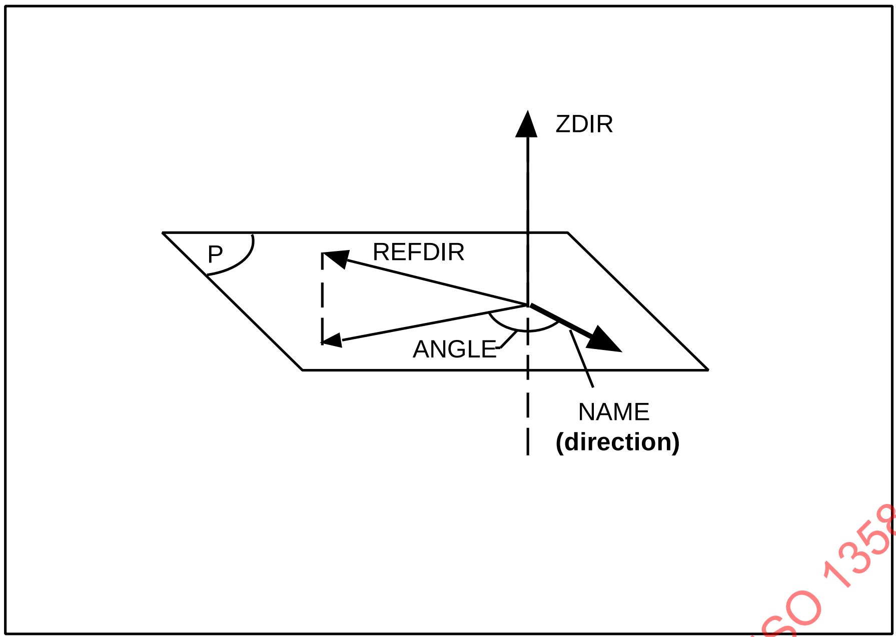


Figure A. 2 — Function: Dir_2_Dir_Angle

Internal reference:
6.1.9, **6.1.9.3**, 6.2.1.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
4	value of plane angle measure out of permitted range	101	attempt to create a degenerated entity
117	given directions are parallel	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.1.1.4 X direction from an axis2_placement

Function name:
Dir_A2p_X

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	A2PNAM	N	Name of **axis2_placement**	a2p
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **direction**	dir

FORTRAN binding:

```
NAME = DIR_A2P_X ( A2PNAM,KFIX )
```

Effects:
Creates a normalised **direction** entity collinear to the (Ox) axis of the given **axis2_placement**. The name of the created **direction** is returned. This **direction** has a **null_style** assigned to it.

If an error occurs, no entity is created and the entity name is returned as zero.

Notes:
–

Internal reference:
6.1.9.3, 6.1.9.7, 6.1.9.8

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.1.1.5 Y direction from an axis2_placement

Function name:

Dir_A2p_Y

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	A2PNAM	N	Name of **axis2_placement**	a2p
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **direction**	dir

FORTRAN binding:

```
NAME = DIR_A2P_Y( A2PNAM,KFIX )
```

Effects:
Creates a normalised **direction** entity collinear to the (Oy) axis of the given **axis2_placement**. The name of the created **direction** is returned. This **direction** has a **null_style** assigned to it.

If an error occurs, no entity is created and the entity name is returned as zero.

Notes:
–

Internal reference:
6.1.9.3, 6.1.9.7, 6.1.9.8

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.1.1.6 Z direction from an axis2_placement

Function name:

Dir_A2p_Z

interface level:	**2**
geometrical power level:	**2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	A2PNAM	N	Name of **axis2_placement**	a2p
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **direction**	dir

FORTRAN binding:

```
NAME = DIR_A2P_Z ( A2PNAM,KFIX )
```

Effects:

Creates a normalised **direction** entity collinear to the (Oz) axis of the given **axis2_placement**. The name of the created **direction** is returned. This **direction** has a **null_style** assigned to it.

If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) If the current open view is defined as a 2D view (that means that the *geometrical_power_level* entry of the interface status table equals 1), an error occurs, and the entity name is returned as zero.

Internal reference:

6.1.9.3, 6.1.9.8

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.1.2 Axis1_placement (single axis)

Axis1_placement Generation	A1p_Gen
Axis1_placement Between two Points	A1p_2_Pnt

A.5.1.2.1 Axis1_placement generation

Function name:

A1p_Gen

interface level:	**2**
geometrical power level:	**2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	PNTNAM	N	Name of **cartesian_point**	pnt
in	DIRNAM	N	Name of **direction**	dir
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **axis1_placement**	a1p

FORTRAN binding:

```
NAME = A1P_GEN ( PNTNAM,DIRNAM,KFIX )
```

Effects:
Creates an **axis1_placement** entity that is a single axis in 3D space.

— the given **cartesian_point** PNTNAM is duplicated as **p1**. This **cartesian_point** has a **null_style** assigned to.

— the given **direction** DIRNAM is duplicated as **d**. This **direction** has a **null_style** assigned to.

— an **axis1_placement** is instantiated with the **location p1** and the **axis d**. The name of the **axis1_placement** is returned and it has a **null_style** assigned to it.

All input parameters are mandatory; if an error occurs, the entity name is returned as zero.

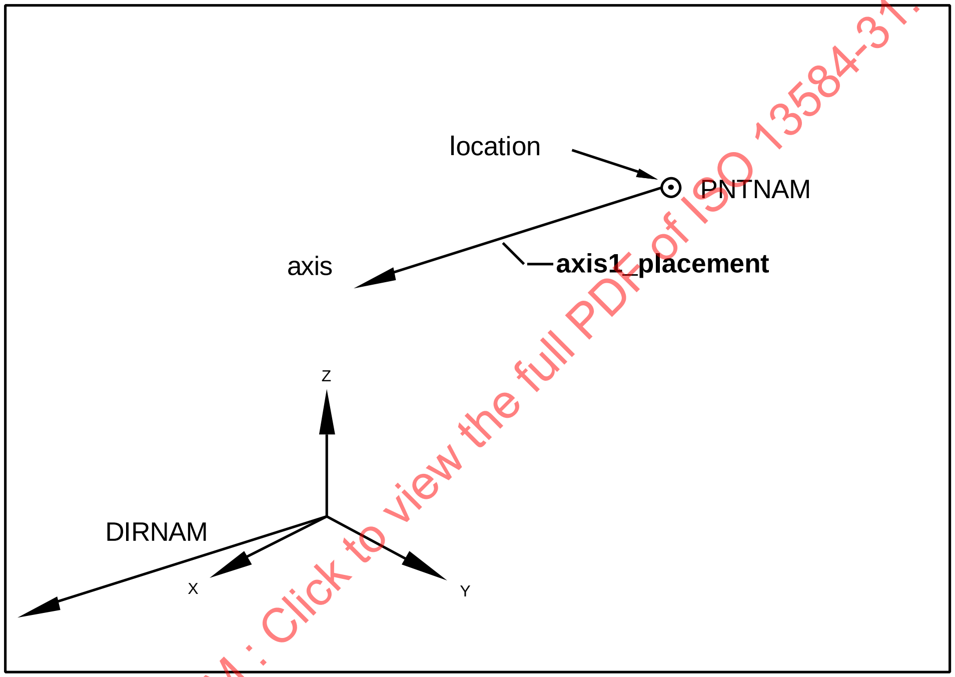


Figure A. 3 — Function: A1p_Gen

Notes:
1) If the current open view is defined as a 2D view (the *geometrical_power_level* entry of the interface status table equals 1), an error occurs, and the entity name is returned as zero.

Internal reference:
6.1.9, **6.1.9.6**, 6.2.1.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.1.2.2 Axis1_placement between two points

Function name:

A1p_2_Pnt

interface level:	**2**
geometrical power level:	**2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	STAPNT	N	name of start **cartesian_point**	pnt
in	ENDPNT	N	name of end **cartesian_point**	pnt
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	name of created **axis1_placement**	a1p

FORTRAN binding:

```
NAME = A1P_2_PNT ( STAPNT,ENDPNT,KFIX )
```

Effects:

Creates an **axis1_placement** entity that is a single axis in 3D space. The given **cartesian_point** STAPNT is duplicated as **p1** and it has a **null_style** assigned to. Let **P2** be a synonym for the second

— a **direction d** is instantiated with **direction_ratio**s derived from **P2-p1**. This **direction** has a **null_style** assigned to it.

— an **axis1_placement** is instantiated with **location p1** and **axis d**. The name of the **axis1_placement** is returned and it has a **null_style** assigned to it.

given **cartesian_point** ENDPNT. Then:

The distance between the two **cartesian_point**s shall be in range [EPS,MAX]. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) If the current open view is defined as a 2D view (the *geometrical_power_level* entry of the interface status table equals 1), an error occurs, and the entity name is returned as zero.

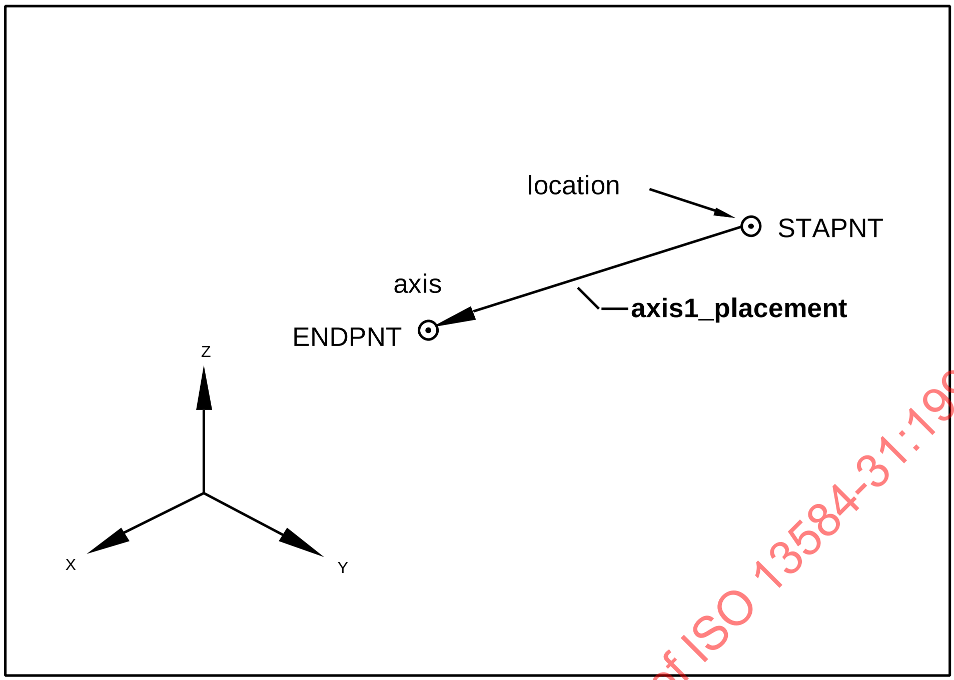


Figure A. 4 — Function: A1p_Pnt

Internal reference:
6.1.9, **6.1.9.6**, 6.2.1.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
103	distance between two points out of range [EPS,MAX]	201	temporary database overflow
202	error while sending entity to CAD system	203	function not compatible with implemented interface level
204	function not compatible with current power level	1001	enumerated value out of range

A.5.1.3 Axis2_placement (Local Coordinate System)

Axis2_placement by 3 point	A2p_3_Pnt
Axis2_placement by 2 direction	A2p_2_Dir
Axis2_placement by 2 direction (Ox) and (Oy)	A2p_2_Dir_Xy
Axis2_placement positioning relative	A2p_Position_Relative
Axis2_placement by reference system	A2p_Ref_Sys

A.5.1.3.1 Axis2_placement by 3 points

Function name:
A2p_3_Pnt

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	CENPNT	N	name of **cartesian_point**, defining the origin	pnt
in	AXSPNT	N	name of **cartesian_point** in direction of Z axis (shall be ignored in case of 2D view)	pnt
in	REFPNT	N	name of **cartesian_point** either in direction of an approximation to X axis, or in direction of exact X axis in case of 2D view	pnt
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	name of created **axis2_placement**	a2p

FORTRAN binding:

```
NAME = A2P_3_PNT ( CENPNT,AXSPNT,REFPNT,KFIX )
```

Effects:

Creates an **axis2_placement** entity that is an orthogonal Local Coordinate System (LCS) in the current OVC Reference System. The type of **axis2_placement** created is dependent upon the initialisation of the open view, i.e. an **axis2_placement_2d** will be instantiated in the case of a 2D view, or an **axis2_placement_3d** in the case of a 3D view. For creating an **axis2_placement_3d**, the three given points CENPNT, AXPNT and REFPNT shall be used to create the origin (O) and the two axes (Oz and Ox) of the placement coordinate system. For creating an **axis2_placement_2d**, only two of the given three points (CENPNT and REFPNT) are used to create the origin (O) and the (Ox) axis of the placement coordinate system.

The given **cartesian_point** CENPNT is duplicated as **p1**. This **cartesian_point** is used to define the origin of the placement, and has a **null_style** assigned to it. Then:

In the case of the instanciation of an **axis2_placement_3d**:

— let **P2** and **P3** be a synonym for the two given **cartesian_point**s, AXSPNT and REFPNT, respectively.

— a **direction d1** is instantiated with **direction_ratio**s defined by **P2-p1**. This **direction** is used to define the exact direction of placement Z axis, and has a **null_style** assigned to it. Where distance between the two **cartesian_point**s shall be in range [EPS,MAX].

— a **direction d2** is instantiated with **direction_ratio**s defined by **P3-p1**. This **direction** is used to define an approximation to the placement X axis direction, and has a **null_style** assigned to it. The distance between the two **cartesian_point**s shall be in range [EPS,MAX].

— an **axis2_placement_3d** is instantiated with **location p1** and **axis d1** and **ref_direction d2**. This **axis2_placement_3d** has a **null_style** assigned to it. The name of the **axis2_placement_3d** is returned.

In the case of the instanciation of an **axis2_placement_2d**:

— let **P3** be a synonym for the one given **cartesian_point**, REFPNT

— a **direction d2** is instantiated with **direction_ratios** defined by **P3-p1**. This **direction** is used to define the exact direction of placement X axis, and has a **null_style** assigned to it. The distance between the two **cartesian_points** shall be in range [EPS,MAX].

— an **axis2_placement_2d** is instantiated with **location p1** and **ref_direction d2**. This **axis2_placement_2d** has a **null_style** assigned to it. The name of the **axis2_placement_2d** is returned.

If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) If necessary an adjustment of **ref_direction** is made to maintain orthogonality to the axis direction, performed by projecting **ref_direction** onto a plane normal to **axis**.

2) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the given parameter AXSPNT will subsequently be ignored by the interface.

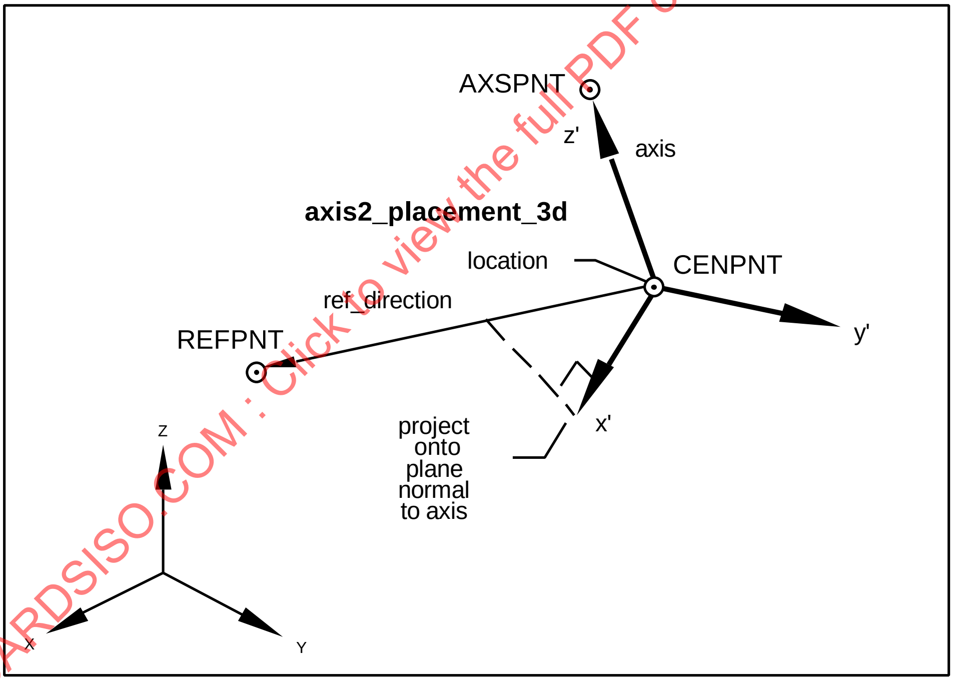


Figure A. 5 — Function: A2p_3_Pnt

Internal reference:

6.1.9, **6.1.9.7**, **6.1.9.8,** 6.2.1.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
103	distance between two points out of range [EPS,MAX]	105	attempt to create a degenerated direction during entity creation
116	given points are linear dependent	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.1.3.2 Axis2_placement by 2 directions

Function name:

A2p_2_Dir

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	meaning	permitted types/values
in	CENPNT	N	name of **cartesian_point**, defining the origin	pnt
in	AXSDIR	N	name of Z axis direction (shall be ignored in case of 2D view)	dir
in	REFDIR	N	name of either an approximated X axis direction, or in exact direction of X axis in case of 2D view	dir
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **axis2_placement**	a2p

FORTRAN binding:

```
NAME = A2P_2_DIR ( CENPNT,AXSDIR,REFDIR,KFIX )
```

Effects:

Creates an **axis2_placement** entity, which is an orthogonal Local Coordinate System (LCS) in the current OVC Reference System. The type of **axis2_placement** created is dependent upon the initialisation of the open view, i.e. an **axis2_placement_2d** will be instantiated in the case of a 2D view, or an **axis2_placement_3d** in the case of a 3D view. For creating an **axis2_placement_3d**, the three given parameters CENPNT, AXDIR and REFDIR shall be used to create the origin (O) and the two axes (Oz and Ox) of the placement coordinate system. For creating an **axis2_placement_2d**, only two of the given three parameters (CENPNT and REFDIR) are used to create the origin (O) and the (Ox) axis of the placement coordinate system.

The given **cartesian_point** CENPNT is duplicated as **p1**. This **cartesian_point** is used to define the origin of the placement, and has a **null_style** assigned to it. Then:

— the two **directions**, AXSDIR and REFDIR, are duplicated as **d1** and **d2** respectively. This directions defines the exact direction of placement Z axis and an approximated direction to the placement X axis. This **directions** have a **null_style** assigned to them.

— an **axis2_placement_3d** is instantiated with **location p1** and **axis d1** and **ref_direction d2**. The name of the **axis2_placement_3d** is returned and it has a **null_style** assigned to it.

In the case of the instanciation of an **axis2_placement_3d**.

In the case of the instanciation of an **axis2_placement_2d**.

— the **direction** REFDIR is duplicated as **d2** and it has a **null_style** assigned to it. This direction is used to define the exact direction of placement X axis.

— an **axis2_placement_2d** is instantiated with **location p1** and **ref_direction d2**. The name of the **axis2_placement_2d** is returned and it has a **null_style** assigned to it.

If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) If necessary an adjustment of **ref_direction** is made to maintain orthogonality to the axis direction, performed by projecting **ref_direction** onto a plane normal to **axis**.

2) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the given parameter AXSDIR will subsequently be ignored by the interface.

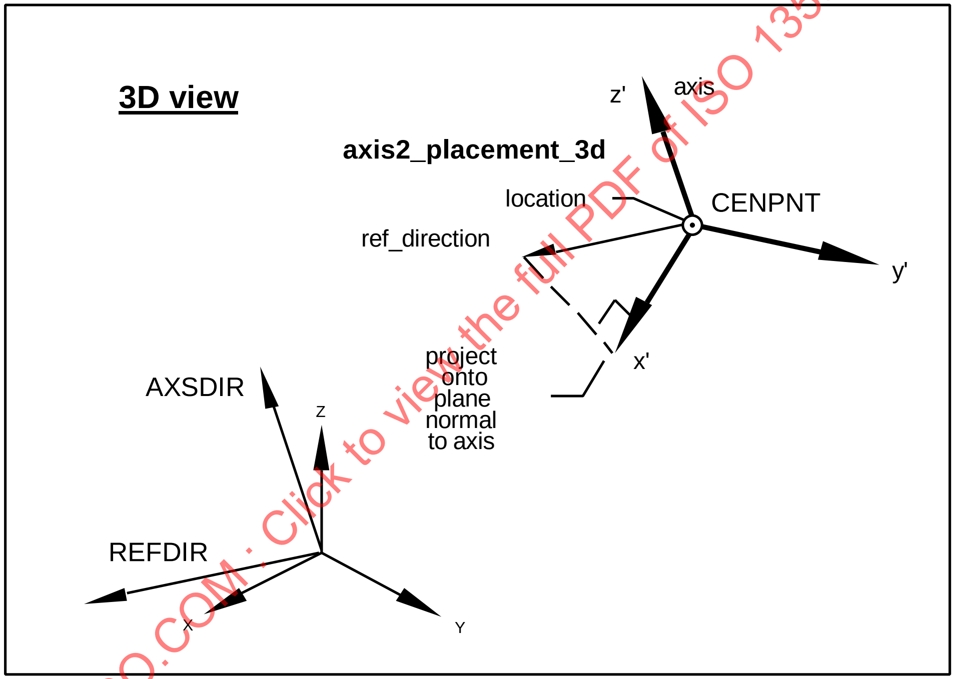


Figure A. 6 — Function: A2p_2_Dir (in a 3D view)

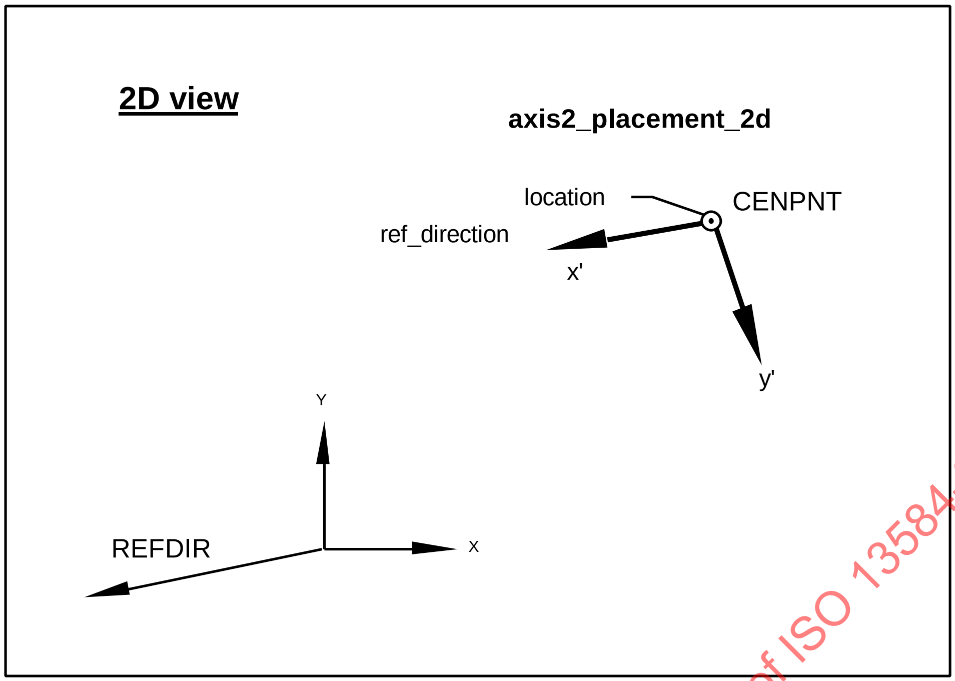


Figure A. 7 — Function: A2p_2_Dir (in a 2D view)

Internal reference:
6.1.9, **6.1.9.7**, **6.1.9.8,** 6.2.1.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
101	attempt to create a degenerated entity	117	given directions are parallel
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.1.3.3 Axis2_placement by 2 directions (Ox) and (Oy)

Function name:
A2p_2_Dir_Xy

Interface level:	**1**
Geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	CENPNT	N	name of **cartesian_point**, defining the origin	pnt
in	REFDIR	N	name of exact X axis direction	dir
in	YAXDIR	N	name of an approximated Y axis direction (shall be ignored in case of 2D view)	dir
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **axis2_placement**	a2p

FORTRAN binding:

```
NAME = A2P_2_DIR_XY ( CENPNT,REFDIR,YAXDIR,KFIX )
```

Effects:
Creates an **axis2_placement** entity that is an orthogonal Local Coordinate System (LCS) in the current OVC Reference System. The type of **axis2_placement** created is dependent upon the

initialisation of the open view, i.e. an **axis2_placement_2d** will be instantiated in the case of a 2D view, or an **axis2_placement_3d** in the case of a 3D view. For creating an **axis2_placement_3d**, the three given parameters CENPNT, REFDIR and YAXDIR shall be used to create the origin (O) and the two axes (Ox and Oy) of the placement coordinate system. For creating an **axis2_placement_2d**, only two of the given three parameters (CENPNT and REFDIR) are used to create the origin (O) and the (Ox) axis of the placement coordinate system.

The given **cartesian_point** CENPNT is duplicated as **p1**, used to define the origin of the placement and the given **direction**, REFDIR, is duplicated as **d1**, used to define the exact direction of placement X axis. These two entities have a **null_style** assigned to them. Then:

In the case of the instanciation of an **axis2_placement_3d**:

— a **direction d2** is created by computing a projection of the normalised direction of YAXDIR onto a plane normal to **d1**. This **direction** has a **null_style** assigned to it.

— a **direction d3** is instantiated, with attributes derived from a cross product of **d1** and **d2.** This **direction** define the exact direction of placement Z axis and it has a **null_style** assigned to it.

— an **axis2_placement_3d** is instantiated with **location p1** and **axis d3** and **ref_direction d1**. The name of the **axis2_placement_3d** is returned and it has a **null_style** assigned to it.

In the case of the instanciation of an **axis2_placement_2d**:

— an **axis2_placement_2d** is instantiated with **location p1** and **ref_direction d1**. The name of the **axis2_placement_2d** is returned and it has a **null_style** assigned to it.

If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the given parameter YAXDIR will subsequently be ignored by the interface.

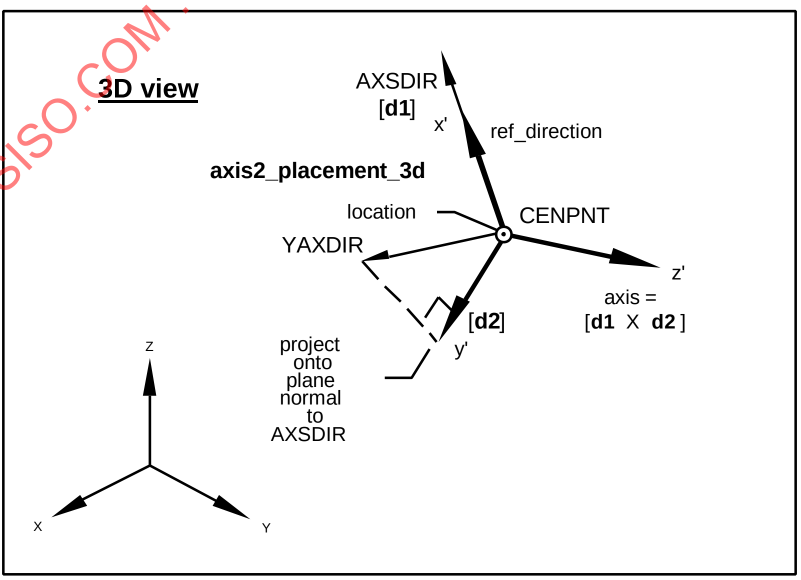


Figure A. 8 — Function: A2p_2_Dir_Xy

Internal reference:
6.1.9, **6.1.9.7**, **6.1.9.8,** 6.2.1.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
101	attempt to create a degenerated entity	117	given directions are parallel
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.1.3.4 Axis2_placement positioning relative

Function name:
A2p_Position_Relative

Interface level:	**1**
Geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	REFLST	6xD	list of length 6, containing specification of successive rotations and relative position (1): relative rotation angle in (Oxy) plane around Z axis of the OVC Reference System (2): relative rotation angle in (Ozy) plane around X axis of the OVC Reference System (3): relative rotation angle in (Ozx) plane around Y axis of the OVC Reference System (4): relative displacement in (Ox) direction of the current OVC Reference System (5): relative displacement in (Oy) direction of the current OVC Reference System (6): relative displacement in (Oz) direction of the current OVC Reference System	(-360°≤REFLST(1)≤360°) (-360°≤REFLST(2)≤360°) (-360°≤REFLST(3)≤360°) (0.0 OR (EPS≤|REFLST(4)|≤MAX)) (0.0 OR (EPS≤|REFLST(5)|≤MAX)) (0.0 OR (EPS≤|REFLST(6)|≤MAX))
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **axis2_placement**	a2p

FORTRAN binding:

```
NAME = A2P_POSITION_RELATIVE ( REFLST,KFIX )
```

Effects:
Creates an **axis2_placement** entity that is an orthogonal Local Coordinate System (LCS) in the current OVC Reference System, positioning relative to the current OVC Reference System. The type of **axis2_placement** created is dependent upon the initialisation of the open view, i.e. an **axis2_placement_2d** will be instantiated in the case of a 2D view, or an **axis2_placement_3d** in the case of a 3D view.

In the case of the instanciation of an **axis2_placement_3d**:

— an **axis2_placement_3d** is instantiated as a copy of the current OVC Reference System. All implicit entities of the instantiated **axis2_placement_3d** and the entity itself have a **null_style** assigned to them.

— the transformation matrices contained within the given parameter REFLIST are applied to the new **axis2_placement_3d** in the following sequence:

1) rotate around the Z axis of the current OVC Reference System

2) rotate around the X axis of the current OVC Reference System

3) rotate around the Y axis of the current OVC Reference System

4) displace the origin of the new **axis2_placement_3d** in the X, Y and Z axis direction of the current OVC Reference System.

— the name of the **axis2_placement_3d** is returned and it has a **null_style** assigned to it.

In the case of the instanciation of an **axis2_placement_2d**:

— an **axis2_placement_2d** is instantiated as a copy of the current OVC Reference System.

— the transformation matrices contained within the given parameter REFLIST are applied to the new **axis2_placement_2d** in the following sequence:

5) rotate in the (Oxy) plane of the current OVC Reference System

6) displace the origin of the new **axis2_placement_2d** in the X and Y axis direction of the current OVC Reference System

— the name of the **axis2_placement_2d** is returned.

The values of rotation angles are measured in *OVC_angle_units* and the displacement values are measured in *OVC_length_units* , and they shall either be equal to zero or in range [EPS,MAX]. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) If the calculated Euclidean norm of displacement is in range [ZERO_VALUE,EPS], no translation shall be performed and no error occurs.

2) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the REFLST(2:3) values for axis rotation angles and the REFLST(6) value for displacement in Z axis direction will subsequently be ignored by the interface.

Internal reference:

5.1.3, 6.1.9, **6.1.9.7**, **6.1.9.8,** 6.2.1.2

Errors:

3	value for length measure out of permitted range	4	value for plane angle measure out of permitted range
101	attempt to create a degenerated entity	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.1.3.5 Axis2_placement by reference system

Function name:

A2P_REF_SYS

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **axis2_placement**	a2p

FORTRAN binding:

```
NAME = A2P_REF_SYS ( KFIX )
```

Effects:

Creates an **axis2_placement** entity that is an orthogonal Local Coordinate System (LCS) in the current OVC Reference System, with location and orientation identical to the location and orientation of the current OVC Reference System.

The type of an **axis2_placement** created is dependent upon the initialisation of the open view, i.e. an **axis2_placement_2d** will be instantiated in the case of a 2D view, or an **axis2_placement_3d** in the

— an **axis2_placement_3d** is instantiated with **location, axis** and **ref_direction** derived from the current Reference System. The name of the **axis2_placement_3d** is returned and it has a **null_style** assigned to it.

case of a 3D view.

— an **axis2_placement_2d** is instantiated with **location** and **ref_direction** derived from the current OVC Reference System. The name of the **axis2_placement_2d** is returned and it has a **null_style** assigned to it.

In the case of the instanciation of an **axis2_placement_2d**.

If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

–

Internal reference:

5.3.1, **6.1.9.7**, **6.1.9.8,** 6.2.1.2

Errors:

201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.2 Point entities

A.5.2.1 Points with canonical definition

Point cartesian absolute	Pnt_Cartesian_Absolute
Point cartesian relative	Pnt_Cartesian_Relative
Point polar absolute	Pnt_Polar_Absolute
Point polar relative	Pnt_Polar_Relative
Point cylinder absolute	Pnt_Cylinder_Absolute
Point cylinder relative	Pnt_Cylinder_Relative

A.5.2.1.1 Point cartesian absolute

Function name:

Pnt_Cartesian_Absolute

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	X	D	x-coordinate of **cartesian_point**	(0.0 OR (EPS≤ |X| ≤MAX))
in	Y	D	y-coordinate of **cartesian_point**	(0.0 OR (EPS≤ |Y| ≤MAX))
in	Z	D	z-coordinate of **cartesian_point**	(0.0 OR (EPS≤ |Z| ≤MAX))
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_CARTESIAN_ABSOLUTE ( X,Y,Z,KFIX )
```

Effects:

Creates a **cartesian_point** entity with **coordinates x**, **y** and **z**. These coordinates are derived from the given parameters X, Y and Z respectively, where these coordinates are cartesian coordinates in the current OVC reference system. This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned.

The given coordinates are measured in *OVC_length_units*. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the given parameter Z will subsequently be ignored by the interface.

Internal Reference:

6.1.9.2, 6.2.4, 8.2

Errors:

3	value for length measure out of permitted range	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.2.1.2 Point cartesian relative

Function name:

Pnt_Cartesian_Relative

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	PNTNAM	N	name of reference **cartesian_point**	pnt
in	DX	D	x-coordinate of **cartesian_point**	(0.0 OR (EPS≤|DX|≤MAX))
in	DY	D	y-coordinate of **cartesian_point**	(0.0 OR (EPS≤|DY|≤MAX))
in	DZ	D	z-coordinate of **cartesian_point**	(0.0 OR (EPS≤|DZ|≤MAX))
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_CARTESIAN_RELATIVE ( PNTNAM,DX,DY,DZ,KFIX )
```

Effects:

Creates a **cartesian_point** entity with **coordinates x**, **y** and **z**. These coordinates are derived from the given parameters DX, DY and DZ, relative to the given reference **cartesian_point** PNTNAM, where these coordinates are cartesian coordinates in the current OVC reference system. This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned.

The given coordinates are measured in *OVC_length_units*. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) If the calculated distance between PNTNAM and the new created **cartesian_point** is in range [ZERO_VALUE,EPS], a copy of the coordinates from PNTNAM is used to create this new **cartesian_point**.

2) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the given parameter DZ will subsequently be ignored by the interface.

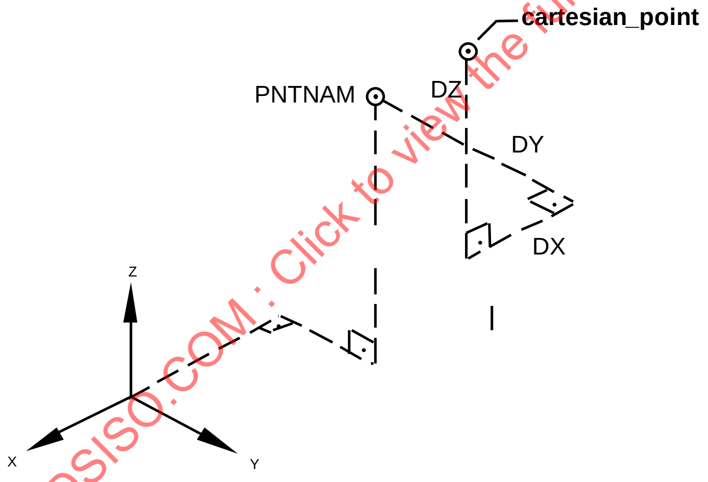


Figure A. 9 — Function: Pnt_Cartesian_Relative

Internal Reference:

6.1.9.2, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.2.1.3 Point polar absolute

Function name:

Pnt_Polar_Absolute

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	PHI	D	angle in (Oxy) plane relative to (Ox) axis of the current OVC	(-360° ≤ PHI ≤ 360°)
in	THETA	D	angle from (Oz) axis to (Oxy) plane of the current OVC	(-360° ≤ THETA ≤ 360°)
in	RAD	D	distance of **cartesian_point** from origin	(0.0 OR (EPS ≤ RAD ≤ MAX))
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_POLAR_ABSOLUTE ( PHI,THETA,RAD,KFIX )
```

Effects:

Creates a **cartesian_point** entity with **coordinates x**, **y** and **z**. These coordinates are calculated from the polar coordinates derived from the given parameters PHI, THETA and RAD, where these coordinates are coordinates in the current OVC reference system. This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned.

The given angles are counted in *OVC_angle_units*, the distance is measured in *OVC_length_units* and shall either be zero or in range [EPS,MAX]. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the given parameter THETA that defines implicit the value for **z**, will subsequently be ignored by the interface.

Internal Reference:

6.1.9.2, 6.2.4, 8.2

Errors:

3	value for length measure out of permitted range	4	value for plane angle measure out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.2.1.4 Point polar relative

Function name:

Pnt_Polar_Relative

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	PNTNAM	N	name of reference **cartesian_point**	pnt
in	PHI	D	angle in (Oxy) plane relative to (Ox) axis of a virtual reference system	(-360° ≤ PHI ≤ 360°)
in	THETA	D	angle from (Oz) axis to (Oxy) plane of a virtual reference system	(-360° ≤ THETA ≤ 360°)
in	RAD	D	distance of **cartesian_point** from origin	(0.0 OR (EPS ≤ RAD ≤ MAX))
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_POLAR_RELATIVE ( PNTNAM,PHI,THETA,RAD,KFIX )
```

Effects:

Creates a **cartesian_point** entity with **coordinates x**, **y** and **z**. These coordinates are calculated from the polar coordinates derived from the given parameters PHI, THETA and RAD, relative to the given reference **cartesian_point** PNTNAM where these coordinates are coordinates in the current OVC reference system. This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned.

The given angles are counted in *OVC_angle_unit*s*,* and the distance is measured in *OVC_length_units* and shall either be zero or in range [EPS,MAX]. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) If the calculated distance between PNTNAM and the new created **cartesian_point** is in range [ZERO_VALUE,EPS], a copy of the coordinates from PNTNAM is used to create this new **cartesian_point**.

2) When the current open view is defined as a 2D view (the *geometrical_power_level* entry of the interface status table equals 1), the given parameter THETA that implicit defines the value for **z**, will subsequently be ignored by the interface.

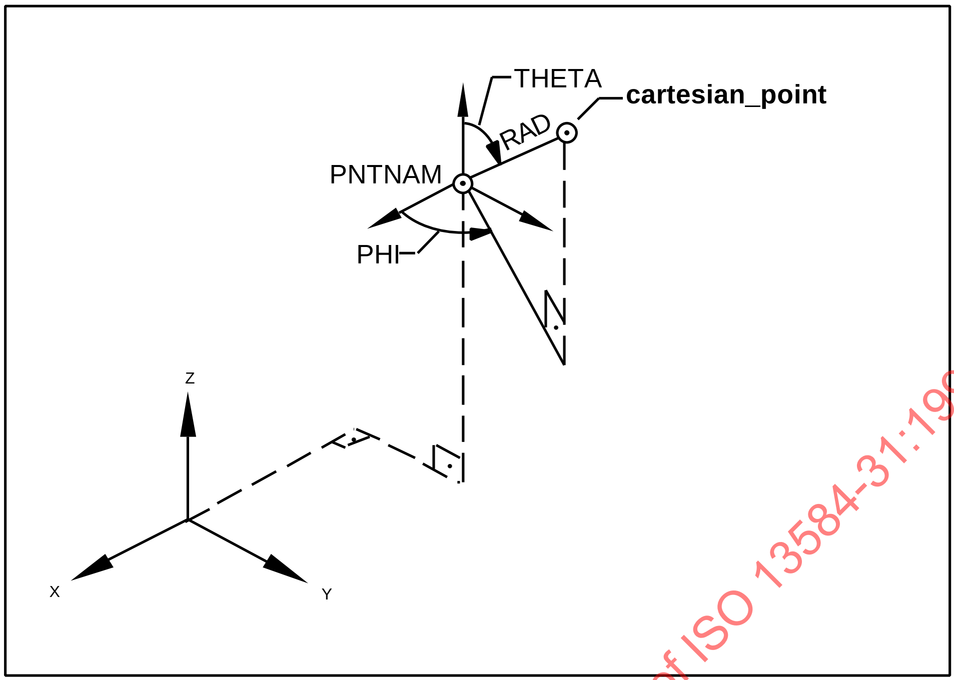


Figure A. 10 — Function: Pnt_Polar_Relative

Internal Reference:
6.1.9.2, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	4	value for plane angle measure out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.2.1.5 Point cylinder absolute

Function name:
Pnt_Cylinder_Absolute

interface level:	**2**
geometrical power level:	**2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	PHI	D	angle in (Oxy) plane relative to (Ox) axis of the current OVC	(-360° ≤ PHI ≤ 360°)
in	RAD	D	length of projection in (Oxy) plane of the current OVC	(0.0 OR (EPS ≤ RAD ≤ MAX))
in	HEIGHT	D	distance of **cartesian_point** from (Oxy) plane of the current OVC	(EPS ≤ |HEIGHT| ≤ MAX)
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_CYLINDER_ABSOLUTE ( PHI,RAD,HEIGHT,KFIX )
```

Effects:

Creates a **cartesian_point** entity with **coordinates x**, **y** and **z**. These coordinates are calculated from the cylindrical coordinates derived from the given parameters PHI, RAD and HEIGHT, where these coordinates are coordinates in the current OVC reference system. This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it, and the name of the **cartesian_point** is returned.

The given angle is counted in *OVC_angle_units*, Rad and HEIGHT are measured in *OVC_length_units*. The length of projection in (Oxy) plane (RAD) shall either be zero or in range [EPS,MAX]. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) If the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), an error occurs and the entity name is returned as zero.

Internal Reference:

6.1.9.2, 6.2.4, 8.2

Errors:

3	value for length measure out of permitted range	4	value for plane angle measure out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.2.1.6 Point cylinder relative

Function name:

Pnt_Cylinder_Relative

interface level:	**2**
geometrical power level:	**2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	PNTNAM	N	name of reference **cartesian_point**	pnt
in	PHI	D	angle in (Oxy) plane relative to (Ox) axis of a virtual reference system	(-360° ≤ PHI ≤ 360°)
in	RAD	D	length of projection in (Oxy) plane of a virtual reference system	(0.0 OR (EPS ≤ RAD ≤ MAX))
in	HEIGHT	D	vertical distance of **cartesian_point** from (Oxy) plane of a virtual reference system	(EPS ≤ |HEIGHT| ≤ MAX)
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_CYLINDER_RELATIVE ( PNTNAM,PHI,RAD,HEIGHT,KFIX )
```

Effects:

Creates a **cartesian_point** entity with **coordinates x**, **y** and **z**. These coordinates are calculated from the cylindrical coordinates derived from the given parameters PHI, RAD and HEIGHT, relative to the given reference **cartesian_point** PNTNAM where these coordinates are coordinates in the current OVC reference system. This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it, and the name of the **cartesian_point** is returned.

The given angle is counted in *OVC_angle_units,* RAD and HEIGHT are measured in *OVC_length_units*. The length of projection, RAD, shall be either zero or in range [EPS,MAX]. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) If the calculated distance between PNTNAM and the new created **cartesian_point** is in range [ZERO_VALUE,EPS], a copy of the coordinates from PNTNAM is used to create this new **cartesian_point**.

2) If the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), an error occurs and the entity name is returned as zero.

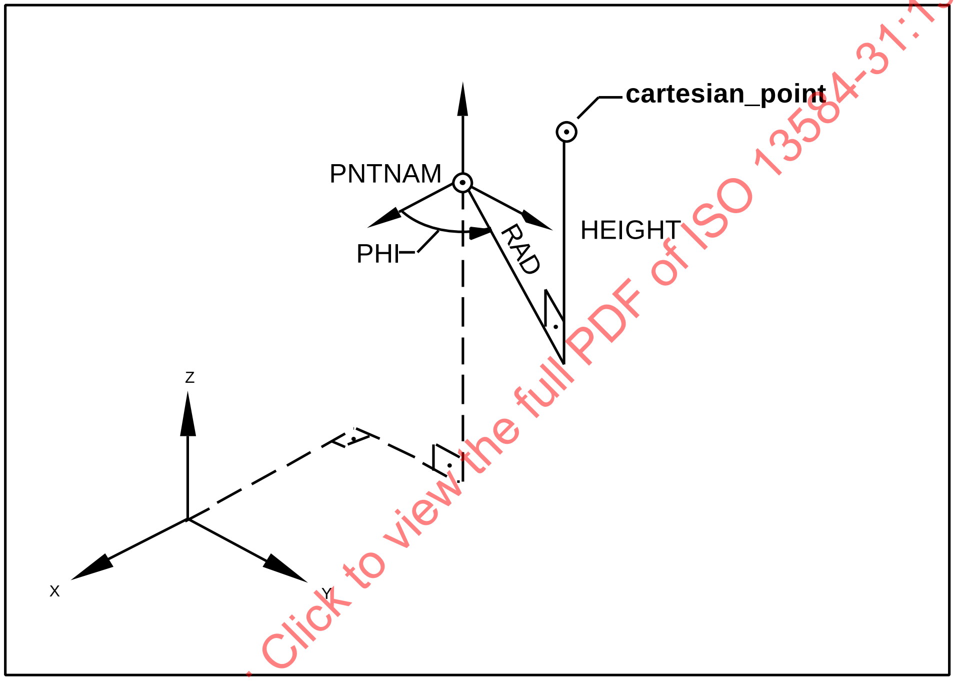


Figure A. 11 — Function: Pnt_Cylinder_Relative

Internal Reference:
6.1.9.2, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	4	value for plane angle measure out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.2.2 Points with constrained based definition

Point at begin of a curve entity	Pnt_Begin_Ent
Point at end of a curve entity	Pnt_End_Ent

Point at intersection of two basic entities	Pnt_Intersection_2_Ent
Point tangential to a circular arc	Pnt_Tangential_Arc
Point at centre of a circular arc	Pnt_Centre_Arc
Point in the middle of a basic entity	Pnt_Middle_Ent
Point as a projection on a basic entity	Pnt_Projection_Ent
Point as a projection on an a2p entity	Pnt_Projection_A2p

A.5.2.2.1 Point at begin of a curve entity

Function name:

Pnt_Begin_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	ENTNAM	N	name of entity	curves
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_BEGIN_ENT ( ENTNAM,KFIX )
```

Effects:

Creates a **cartesian_point** entity at the beginning of given curves entity ENTNAM. The **coordinates x**, **y** and **z** of the created **cartesian_point** are derived either from the parameter value (**parameter_1**) or from the **cartesian_point** reference (**point_1**) of **trim_1** attribute of the given basic or conic_arc

— If the given general curve entity ENTNAM is an instance of type **polyline**, the function creates a **cartesian_point** with **coordinates x**, **y** and **z** derived from the first **cartesian_point** of points attribute list.

— If the given general curve entity ENTNAM is an instance of type **api_contour**, the function creates a **cartesian_point** with **coordinates x**, **y** and **z** derived from the first point of the first segment of **composite_curve_segment**s, that is the beginning point of an **api_contour**.

entity ENTNAM.

This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the value for **z** will subsequently be ignored by the interface.

Internal reference:

6.1.9.2, 6.1.12, 6.1.13, 6.1.14, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.2.2.2 Point at end of a curve entity

Function name:

Pnt_End_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	ENTNAM	N	name of entity	curves
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_END_ENT ( ENTNAM,KFIX )
```

Effects:

Creates a **cartesian_point** entity at the end of given curves entity ENTNAM. The **coordinates x**, **y** and **z** of the created **cartesian_point** are derived either from the parameter value (**parameter_2**) or from the **cartesian_point** reference (**point_2**) of **trim_2** attribute of the given basic or conic_arc entity ENTNAM.

— If the given general curve entity ENTNAM is an instance of type **polyline**, the function creates a **cartesian_point** with **coordinates x**, **y** and **z** derived from the last **cartesian_point** of points attribute list.

— If the given general curve entity ENTNAM is an instance of type **api_contour**, the function creates a **cartesian_point** with coordinates **x**, **y** and **z** derived from the first point of the first segment of **composite_curve_segments**, that is the end point of an **api_contour**.

This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the value for **z** will subsequently be ignored by the interface.

Internal reference:

6.1.9.2, 6.1.12, 6.1.13, 6.1.14, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.2.2.3 Point at intersection of two basic entities

Function name:

Pnt_Intersection_2_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	ENTNM1	N	name of first entity	basic
in	ENTNM2	N	name of second entity	basic
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_INTERSECTION_2_ENT ( ENTNM1,ENTNM2,KFIX )
```

Effects:

Creates a **cartesian_point** entity as a point belonging to both given basic entities ENTNM1 and ENTNM2. The **coordinates x**, **y** and **z** of the created **cartesian_point** are calculated through an intersection of ENTNM1 with ENTNM2, in this order. If the result of the intersection is not unique, then two intersection points are calculated and the following selection process will be used to define one of them.

The intersection point is chosen, that is placed closer to the first trimming point (**trim_1**) of the given **api_line** entity.

If one given entity ENTNM1 or ENTNM2 is an instance of type **api_line**, then:

If of both given entities, ENTNM1 and ENTNM2, are instances of type **api_circular_arc**, then:

1st: a vector $\mathbf{v_n}$ will be created by a cross product of the vector $\mathbf{v_1}$, that is defined as a vector from the centre of the **basis_curve** of the first **api_circular_arc** to the centre of the **basis_curve** of the second **api_circular_arc**, and the vector $\mathbf{v_2}$, that is either a vector normal to the (Oxy) plane of the first **api_circular_arc** entity if the **sense_agreement** flag is equal to true or a vector in opposite direction of the vector normal, if **sense_agreement** flag is equal to false.

2nd: the vector $\mathbf{v_{res}}$ will be created, as a vector from the centre of the **basis_curve** of the first **api_circular_arc** to an intersection point.

3rd: the intersection point that ensures $\mathbf{v_{res}} \times \mathbf{v_n}$ ³ **0** will be chosen.

This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) An intersection occurs when the minimal distance between the two joined entities (ENTNM1 and ENTNM2) is ≤ |ZERO_VALUE|, and there are not more than two points with this quality belonging to the two distinct entities.

2) The **cartesian_point** is created, only if it is lying within the parametric range [**trim_1**, **trim_2**] of both given basic curve entities. Otherwise, an error occurs.

3) If the distance between the chosen **cartesian_point** and one of start- or ending point (**trim_1** or **trim_2**) of the given basic curve entities is in range [ZERO_VALUE,EPS], no error occurs and the coordinates from this trimming point will be used to create the new **cartesian_point**.

4) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the selection process for the case ENTNM1 and ENTNM2 are of type **api_circular_arc**, calculates in a virtual 3D-space, and the value for **z** will subsequently be ignored by the interface.

5) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is greater equal 2), both given entities ENTNM1 and ENTNM2 shall be in the same plane.

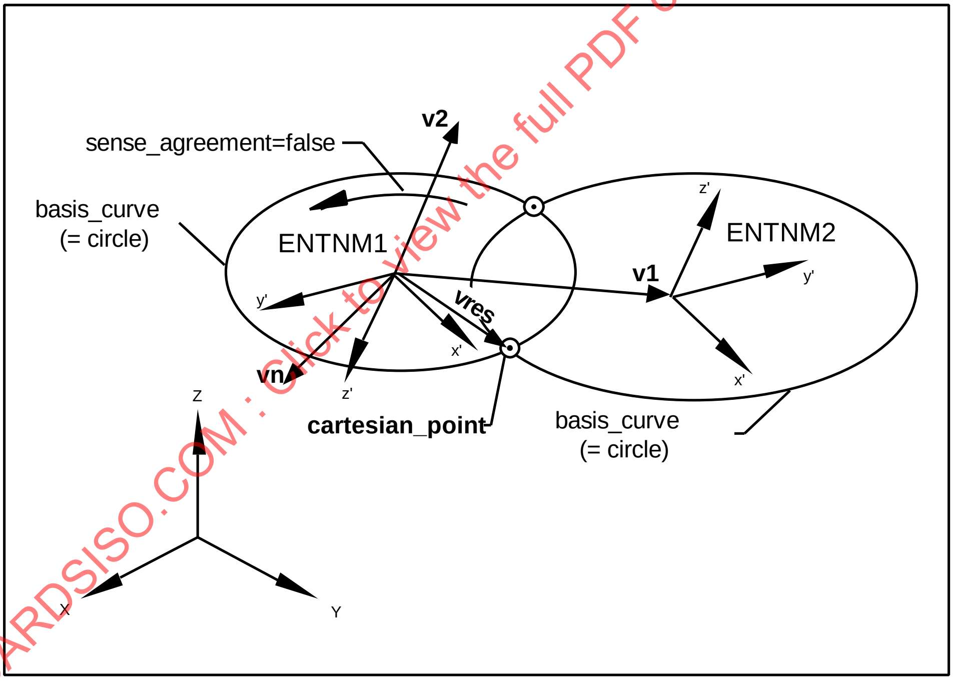


Figure A. 12 — Function: Pnt_Intersection_2_Ent (in a 3D-view)

Internal Reference:

6.1.9.2, 6.1.12, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
110	attempt to create a point outside the parametric range of curves entity	115	given entities are identical
118	given curves entities are parallel/concentric	119	given entities are not in the same plane
122	no intersection of given curves entities	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.2.2.4 Point tangential to a circular arc

Function name:

Pnt_Tangential_Arc

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	ARCNAM	N	Name of **api_circular_arc**	arc
in	LINNAM	N	Name of reference **api_line**	lin
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_TANGENTIAL_ARC ( ARCNAM,LINNAM,KFIX )
```

Effects:

Creates a **cartesian_point** entity as a tangential point on the given **api_circular_arc** entity ARCNAM in conjunction to the given **api_line** entity LINNAM as reference line. This reference line shall not contain the centre of the **basic_curve** of the given **api_circular_arc** ARCNAM. The **coordinates x**, **y** and **z** of the created **cartesian_point** are calculated through a point lying on the ARCNAM entity and that is

— on a parallel line to the reference line that is tangential to the **basis_curve** of the given **api_circular_arc** ARCNAM and

— which has moreover the shortest distance to the infinite straight line defined by the reference line, LINNAM.

This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) The **cartesian_point** is created, only if it is lying within the parametric range [**trim_1**, **trim_2**] of given basic curve ARCNAM. Otherwise, an error occurs.

2) If the distance between the chosen **cartesian_point** and one of start- or ending point (**trim_1** or **trim_2**) of the given basic curve ARCNAM is in range [ZERO_VALUE,EPS], no error occurs and the coordinates from this trimming point will be used to create the new **cartesian_point**.

3) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the value for **z** will subsequently be ignored by the interface.

4) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is greater equal 2), both given entities ARCNAM and LINNAM shall be in the same plane.

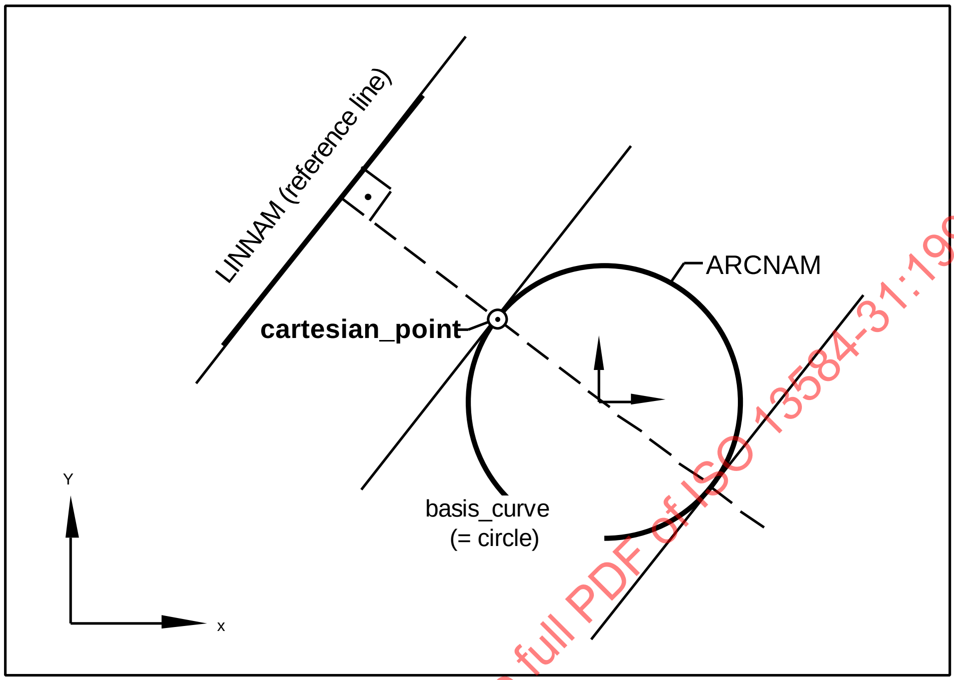


Figure A. 13 — Function: Pnt_Tangential_Arc

Internal Reference:
6.1.9.2, 6.1.12, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
110	attempt to create a point outside the parametric range of curves entity	119	given entities are not in the same plane
127	geometrical design is not feasable	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.2.2.5 Point at centre of a circular arc

Function name:
Pnt_Centre_Arc

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	ARCNAM	N	name of **api_circular_arc**	arc
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME =PNT_CENTRE_ARC ( ARCNAM,KFIX )
```

Effects:

Creates a **cartesian_point** entity at the centre of given **api_circular_arc** entity ARCNAM. The **coordinates x**, **y** and **z** of the created **cartesian_point** are derived from the **position.location** of the **basis_curve** of the **api_circular_arc** ARCNAM. This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the value for **z** will subsequently be ignored by the interface.

Internal Reference:

6.1.9.2, 6.1.12, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.2.2.6 Point in the middle of a basic entity

Function name:

Pnt_Middle_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	ENTNAM	N	name of entity	basic
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_MIDDLE_ENT ( ENTNAM,KFIX )
```

Effects:

Creates a **cartesian_point** entity at the middle of given basic entity ENTNAM. The **coordinates x**, **y** and **z** of the created **cartesian_point** are derived by a calculation of a point lying at the middle of the parametric range between first and second trimming point of the given entity ENTNAM. This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) The **cartesian_point** is created, only if the calculated distance between this **cartesian_point** and both of start- and ending point (**trim_1** or **trim_2**) of the given basic curve ENTNAM is greater than EPS. Otherwise an error occurs.

2) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the value for **z** will subsequently be ignored by the interface.

Internal Reference:

6.1.9.2, 6.1.12, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
110	attempt to create a point outside the parametric range of curves entity	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.2.2.7 Point as a projection on a basic entity

Function name:

Pnt_Projection_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	PNTNAM	N	name of **cartesian_point** to be projected	pnt
in	ENTNAM	N	name of entity	basic
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_PROJECTION_ENT ( PNTNAM,ENTNAM,KFIX )
```

Effects:

Creates a **cartesian_point** entity as a point on the given basic entity ENTNAM by a projection of a given **cartesian_point** PNTNAM.

1) If the given basic entity ENTNAM is an instance of type **api_line**, the function creates a **cartesian_point** with **coordinates x**, **y** and **z** derived from a calculation of a point that results from an orthogonal projection of the point PNTNAM onto the **line** that is the basis_curve of the **api_line**.

2) If the given basic entity ENTNAM is an instance of type **api_circular_arc**, the function creates a **cartesian_point** with **coordinates x**, **y** and **z** derived from an intersection of an infinite straight line that is built up from the point PNTNAM and the centre of the given **api_circular_arc** ENTNAM and the **api_circular_arc** itself. If two intersection points are possible, the point that is located nearer to the point PNTNAM is chosen.

The newly created **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) The **cartesian_point** is created, only if it is lying within the parametric range [**trim_1**, **trim_2**] of given basic curve ENTNAM. Otherwise, an error occurs.

2) If the distance between the newly **cartesian_point** and one of start- or ending point (**trim_1** or **trim_2**) of the given basic curve ENTAM is in range [ZERO_VALUE,EPS], no error occurs and the coordinates derived from this trimming point will be used to adjust the new created **cartesian_point**.

3) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the value for **z** will subsequently be ignored by the interface.

4) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is grater equal 2), both given entities PNTNAM and ENTNAM shall be in the same plane.

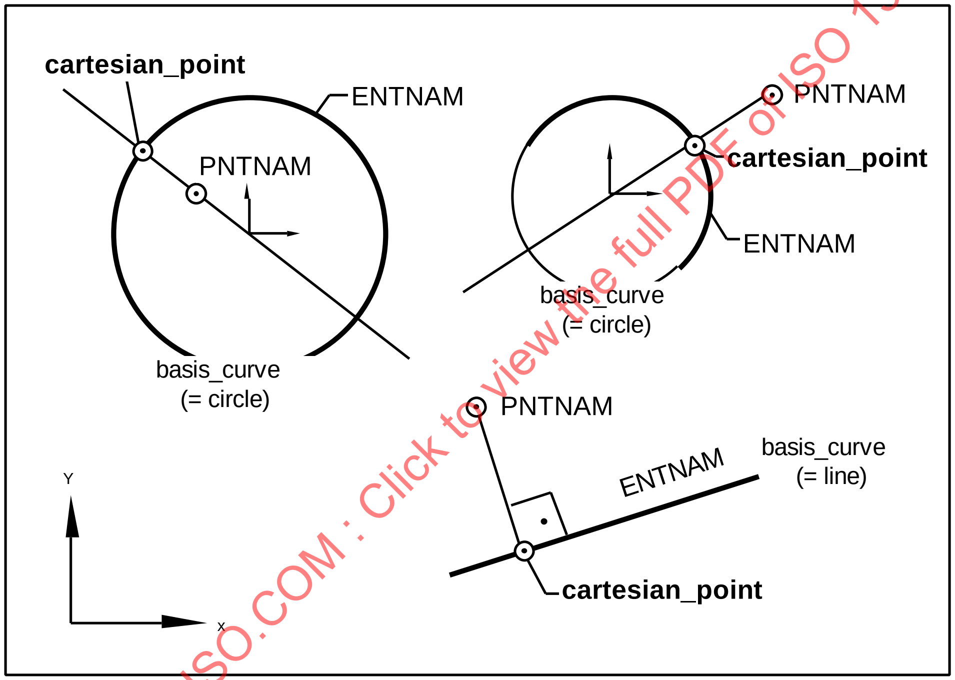


Figure A. 14 — Function: Pnt_Projection_Ent

Internal Reference:
6.1.9.2, 6.1.12, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
110	attempt to create a point outside the parametric range of curves entity	119	given entities are not in the same plane
127	geometrical design is not feasible	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.2.2.8 Point as a projection on an axis2_placement entity

Function name:

Pnt_Projection_A2p

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	PNTNAM	N	Name of **cartesian_point** to be projected	pnt
in	A2PNAM	N	Name of **axis2_placement**	a2p
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **cartesian_point**	pnt

FORTRAN binding:

```
NAME = PNT_PROJECTION_A2P ( PNTNAM,A2PNAM,KFIX )
```

Effects:

Creates a **cartesian_point** entity by a projection of a given **cartesian_point** PNTNAM onto the (Oxy) plane of **axis2_placement** A2PNAM. The **coordinates x, y** and **z** of the created **cartesian_point** are derived from a calculation of a point at the (Oxy) plane of the **axis2_placement** A2PNAM, that have the shortest distance to PNTNAM. This **cartesian_point** has the current *point_style* entry from the interface status table assigned to it. The name of the **cartesian_point** is returned. If an error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) If the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), no error occurs and the **coordinates** of the newly created **cartesian_point** are copied from the give one PNTNAM.

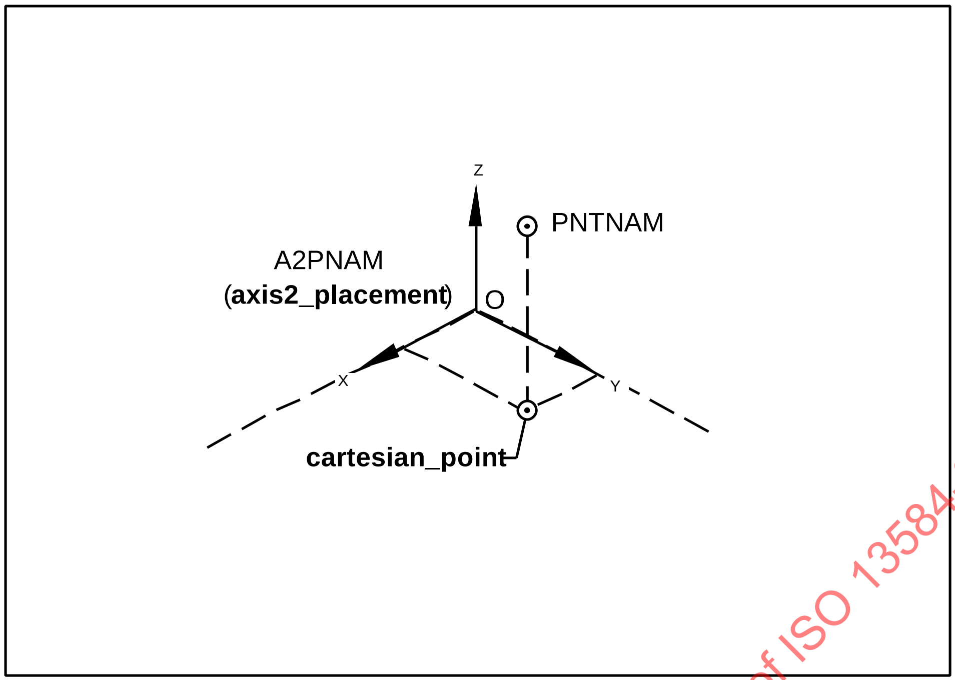


Figure A. 15 — Function: Pnt_Projection_A2p

Internal Reference:
6.1.9.2, 6.1.9.7, 6.1.9.8, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3 Curve entities

A.5.3.1 Basic curve entities

A.5.3.1.1 Line segments (api_line)

Line segment between two points	Lin_2_Pnt
Line segment by start point, length and direction	Lin_Pnt_Length_Dir
Line segment tangential to circular arc	Lin_Tangential_Arc
Line segment tangential to two Circular arc	Lin_Tangential_2_Arc
Line segment as chamfer of two line segments	Lin_Chamfer_2_Lin

A.5.3.1.1.1 Line segment between two points

Function name:
Lin_2_Pnt

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	STAPNT	N	Name of start **cartesian_point**	pnt
in	ENDPNT	N	Name of end **cartesian_point**	pnt
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **api_line**	lin

FORTRAN binding:

```
NAME =  ( STAPNT,ENDPNT,KFIX )
```

Effects:

Creates an **api_line** entity between two existing **cartesian_point**s, where PNTNM1 is the name of the start, and PNTNM2 is the name of the end **cartesian_point**.

The start and end **cartesian_point**s are duplicated as **p1** and **p2** respectively and they have a **null_style** assigned to them. Then:

— a **direction d** is instantiated with **direction_ratios** defined by **p2-p1**. This **direction** has a **null_style** assigned to it.

— a **vector v** is instantiated that **orientation** refers to the **direction d** and with **magnitude** equals **||p2-p1||**. This **vector** has a **null_style** assigned to it.

— a **line l** is instantiated with **pnt p1** and **dir v**. This **line** has a **null_style** assigned to.

— an **api_line** is instantiated with **l** as **basis_curve**, **p1** and **p2** as **trim_1** and **trim_2** respectively, **sense_agreement** equals true and **master_representation** shall be implementation dependent. This **api_line** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_line** is returned.

The distance between these two points shall be in range [EPS,MAX]; if an error occurs, no entity is created and the element name is returned as zero.

Notes:

–

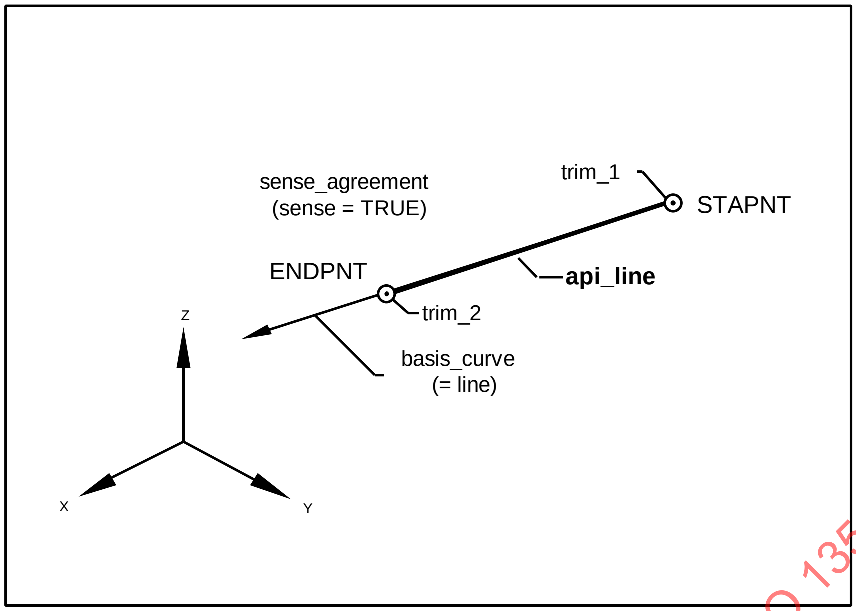


Figure A. 16 — Function: Lin_2_Pnt

Internal Reference:
6.1.9, **6.1.12.1**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
101	attempt to create a degenerated entity	103	distance between two points out of range [EPS,MAX]
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.1.1.2 Line segment by start point, length and direction

Function name:
Lin_Pnt_Length_Dir

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	STAPNT	N	Name of start **cartesian_point**	pnt
in	LEN	D	Length of the **api_line**	(EPS ≤ LEN ≤MAX)
in	DIRNAM	N	Name of **direction**	dir
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **api_line**	lin

FORTRAN binding:

```
NAME = LIN_PNT_LENGTH_DIR ( STAPNT,LEN,DIRNAM,KFIX )
```

Effects:
Creates an **api_line** entity by start **cartesian_point**, length and **direction**.

The start **cartesian_point** STAPNT is duplicated as **p1** and the **direction** DIRNAM is duplicated as **d**. Both entities have a **null_style** assigned to them. Then:

— a **vector v** is instantiated with the **orientation d** and the **magnitude** equals one (normalised vector). This **vector** has a **null_style** assigned to it.

— a **line l** is instantiated with **pnt p1** and **dir v**. This **line** has a **null_style** assigned to it.

— an **api_line** is instantiated with **l** as **basis_curve**, **p1** as **trim_1** and the parameter value LEN as **trim_2**, **sense_agreement** equals true and **master_representation** shall be implementation dependent. This **api_line** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_line** is returned.

The value of LEN shall be in range [EPS,MAX] and is measured in *OVC_length_units* ; if an error occurs, no entity is created and the element name is returned as zero.

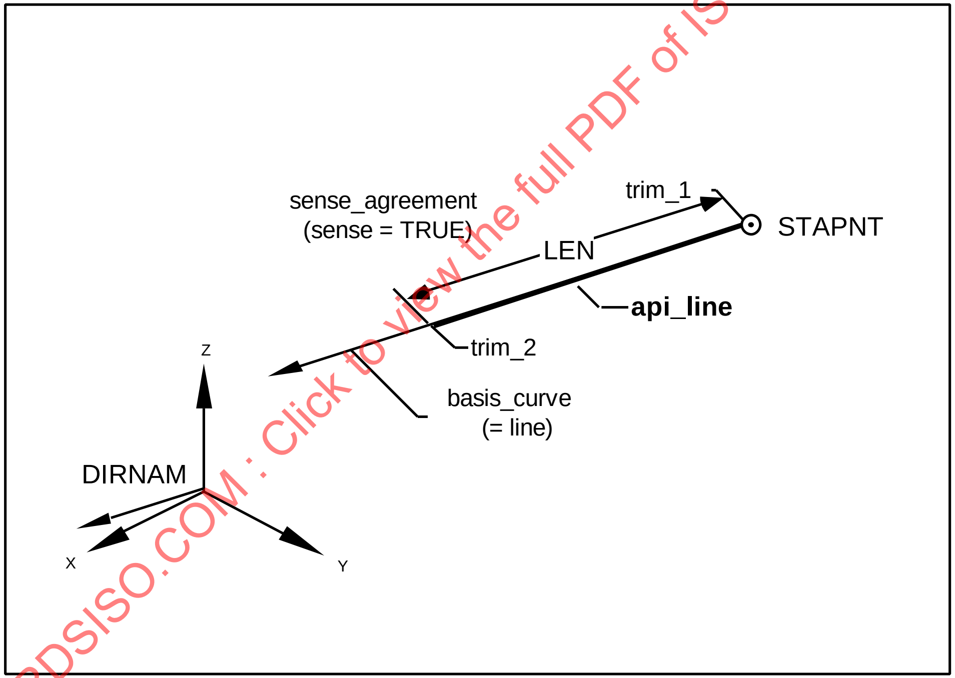


Figure A. 17 — Function: Lin_Pnt_Length_Dir

Notes:
–

Internal Reference:
6.1.9, **6.1.12.1**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	101	attempt to create a degenerated entity
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.1.1.3 Line segment tangential to circular arc

Function name:

Lin_Tangential_Arc

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	STAPNT	N	Name of start **cartesian_point**	pnt
in	ARCNAM	N	Name of **api_circular_arc**	arc
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **api_line**	lin

FORTRAN binding:

```
NAME = LIN_TANGENTIAL_ARC ( STAPNT,ARCNAM,KFIX )
```

Effects:

Creates an **api_line** entity tangential to an existing **api_circular_arc**, ARCNAM, starting from an existing **cartesian_point**, STAPNT. The start **cartesian_point** is duplicated as **p1** and it has a **null_style** assigned to it. Then:

— two **cartesian_point**s, **p2** and **p3**, are instantiated as the two possible tangential points on the **basis_curve** of the **api_circular_arc** ARCNAM. The **cartesian_point**s have a **null_style** assigned to them.

— from the two **cartesian_point**s **p2** and **p3**, the point is chosen, where the direction of a tangent at this point (with respect to the sense agreement flag of ARCNAM) is equal to a direction from the start point **p1** to this tangential point. Let be **p4** a synonym for this chosen point.

— a **direction d** is instantiated with **direction_ratios** defined by **p4-p1**. This **direction** has a **null_style** assigned to it.

— a **vector v** is instantiated that **orientation** refers to the **direction d** with **magnitude** equals ||**p4-p1**||. This **vector** has a **null_style** assigned to it.

— a **line l** is instantiated with **pnt p1** and **dir v**. This **line** has a **null_style** assigned to it.

— an **api_line** is instantiated with **l** as **basis_curve**, **p1** and **p4** as **trim_1** and **trim_2** respectively, **sense_agreement** equals true and **master_representation** shall be implementation dependent. This **api_line** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style**with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_line** is returned. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) The given start **cartesian_point p1** shall lie outside of the **basic_curve** of the **api_circular_arc** ARCNAM.

2) If the distance between the calculated tangential point **p4** and one of trimming points (**trim_1** or **trim_2**) of the given **api_circular_arc** is in range [ZERO_VALUE,EPS], no error occurs and the coordinates derived from this trimming point will be used instead of the calculated one.

3) The **api_line** is created, only if the chosen tangential point **p4** lies within the parametric range [**trim_1**, **trim_2**] of **api_circular_arc** ARCNAM, and the length of this **api_line** is in range [EPS,MAX]. Otherwise an error occurs.

4) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is greater equal 2), the **cartesian_point** PNTNAM and the **api_circular_arc** ARCNAM shall be in the same plane.

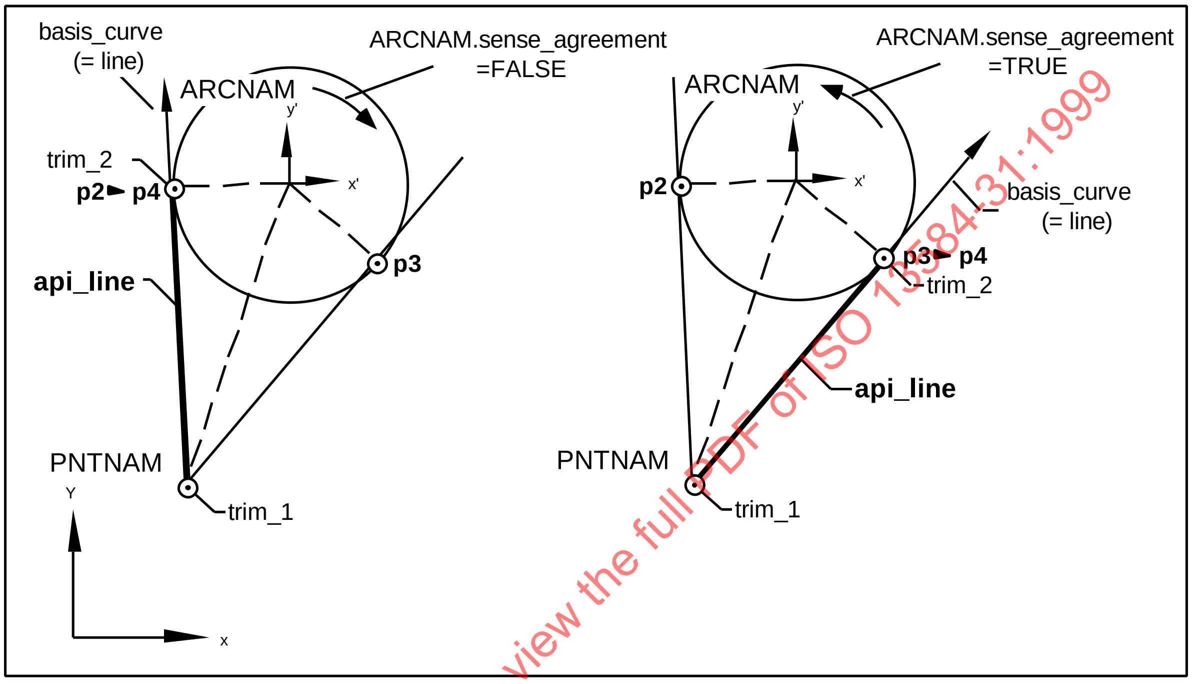


Figure A. 18 — Function: Lin_Tangential_Arc

Internal Reference:
6.1.9, 6.1.12, **6.1.12.1**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
101	attempt to create a degenerated entity	105	attempt to create a degenerated direction during entity creation
110	attempt to create a point outside the parametric range of curves entity	119	given entities are not in the same plane
127	geometrical design is not feasible	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.3.1.1.4 Line segment tangential to two circular arc

Function name:

Lin_Tangential_2_Arc

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	ARCNM1	N	Name of first **api_circular_arc**	arc
in	ARCNM2	N	Name of second **api_circular_arc**	arc
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **api_line**	lin

FORTRAN binding:

```
NAME = LIN_TANGENTIAL_2_ARC ( ARCNM1,ARCNM2,KFIX )
```

Effects:

Creates an **api_line** entity that is an oriented line tangential to two existing **api_circular_arc**s, starting from a tangential point at the first **api_circular_arc**, ARCNM1, and ending at a tangential point at the second **api_circular_arc**, ARCNM2. The directions of the **api_circular_arc**s, implicitly given by their sense agreement flags and their trimming points, determines the orientation of the new **api_line.** In the following, we will consider **C1** and **C2** as synonym for ARCNM1 and ARCNM2 respectively. Then:

— all possible **cartesian_point**s, that are the tangential points at the **basis_curves** of **C1** and **C2**, are instantiated. All these **cartesian_point**s have a **null_style** assigned to them.

— a selection process is initialised to choose a tangential point at **C1** and a tangential point at **C2** . Let **p1** and **p2** be synonym for these points respectively. This process ensures, that the direction of a tangent at this tangential point at **C1** is equal to a tangential point at **C2** (with respect to the sense agreement flags of the **api_circular_arc**s **C1** and **C2**).

— a **direction d** is instantiated with **direction_ratios** defined by **p2-p1**. This **direction** has a **null_style** assigned to it.

— a **vector v** is instantiated that **orientation** refers to the **direction d** with **magnitude** equals ||**p2-p1**||. This **vector** has a **null_style** assigned to it.

— a **line l** is instantiated with **pnt** equals **p1** and **dir** equals **v**. This **line** has a **null_style** assigned to it.

— an **api_line** is instantiated with **l** as **basis_curve**, **p1** and **p2** as **trim_1** and **trim_2** respectively, **sense_agreement** equals true and **master_representation** shall be implementation dependent. This **api_line** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_line** is returned. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) An **api_line** is created, only if the chosen tangential points **p1** and **p2** are lying within the parametric range [**trim_1**, **trim_2**] of their **api_circular_arc**s **C1** and **C2**, respectively, and the distance between these two points, **p1** and **p2**, shall be in range [EPS,MAX]. Otherwise an error occurs.

2) If the distance between the calculated tangential points **p1** and **p2** and one of trimming points (**trim_1** or **trim_2**) of the corresponding given **api_circular_arc**s is in range [ZERO_VALUE,EPS], no error occurs and the coordinates derived from this trimming point will be used instead of the calculated one.

3) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is greater equal 2), both given **api_circular_arc**s shall be in the same plane.

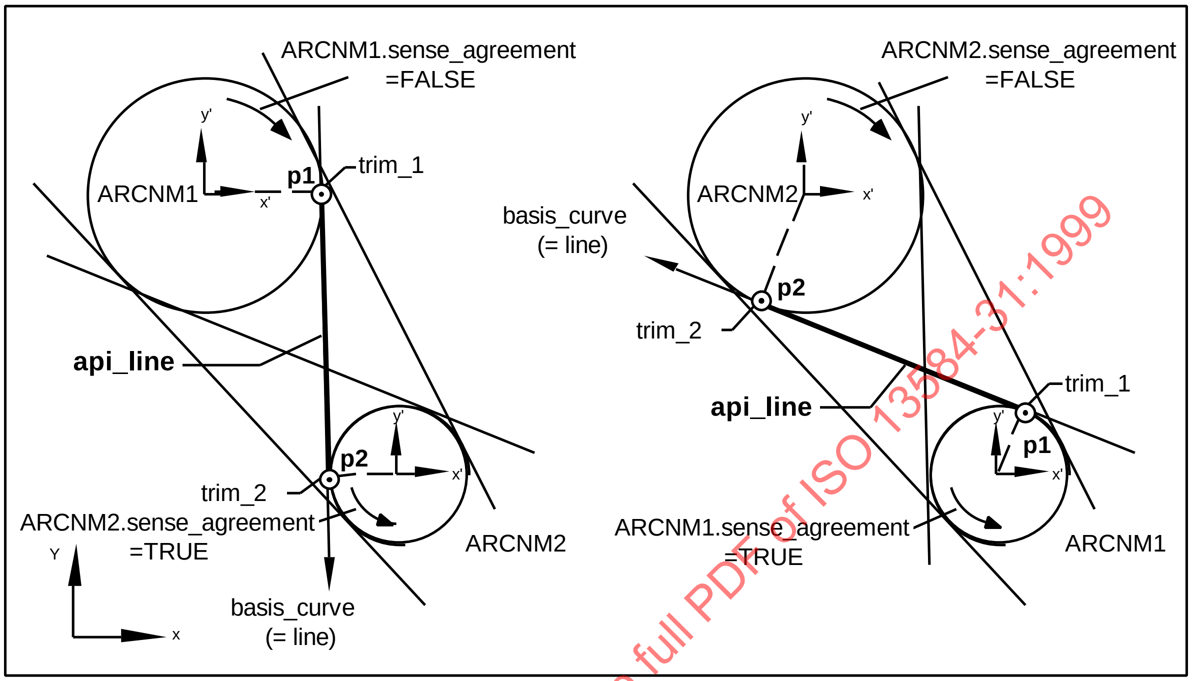


Figure A. 19 — Function: Lin_Tangential_2_Arc

Internal Reference:
6.1.9, 6.1.12, **6.1.12.1**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
101	attempt to create a degenerated entity	105	attempt to create a degenerated direction during entity creation
110	attempt to create a point outside the parametric range of curves entity	115	given entities are identical
118	given curves entities are parallel/concentric	119	given entities are not in the same plane
127	geometrical design is not feasible	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.3.1.1.5 Line segment as chamfer of two lines

Function name:

Lin_Chamfer_2_Lin

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	LEN1	D	cut length on first **api_line**	(EPS ≤ LEN1 ≤ MAX)
in	LEN2	D	cut length on second **api_line**	(EPS ≤ LEN2 ≤ MAX)
in	LINNM1	N	Name of first **api_line**	lin

in	LINNM2	N	Name of second **api_line**	lin
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **api_line**	lin

FORTRAN binding:

```
NAME = LIN_CHAMFER_2_LIN ( LEN1,LEN2,LINNM1,LINNM2,KFIX )
```

Effects:

Creates an **api_line** entity as chamfer between two **api_line**s. Both existing **api_line**s, LINNM1 and LINNM2, shall have an intersection point that lies on each given **api_line**. These **api_line**s will be shortened and the names of the **api_line**s are not changed. They remain in the temporary database until they are explicitly fixed. The orientation of the new created **api_line** (chamfer_line) starts at the new end of the first **api_line**, that is the new trimming point **trim_2** of LINNM1 and ends at the new beginning of the second **api_line**, that is the new trimming point **trim_1** of LINNM2. After shortening, the first and second **api_line** shall no longer intersect.

— a **cartesian_point p1** is instantiated with **coordinates** calculated by an intersection of LINNM1 with LINNM2. This **cartesian_point** has a **null_style** assigned to it.

— a **cartesian_point p2** is instantiated at the distance LEN1 of the intersection point **p1** on the **api_line** LINNM1 and that lies in the opposite direction of the given first **api_line** LINNM1. This **cartesian_point** has a **null_style** assigned to it.

— a **cartesian_point p3** is instantiated at the distance LEN2 of the intersection point **p1** on the **api_line** LINNM2 and that lies in the direction of the given **api_line** LINNM2. This **cartesian_point** has a **null_style** assigned to it.

— a **direction d** is instantiated with **direction_ratios** defined by **p3-p2**. This **direction** has a **null_style** assigned to it.

— a **vector v** is instantiated that **orientation** refers to the **direction d** with **magnitude** equals **||p3-p2||**. This **vector** has a **null_style** assigned to it.

— a **line l** is instantiated with **pnt p2** and **dir v**. This **line** has a **null_style** assigned to it.

— an **api_line** is instantiated with **l** as **basis_curve**, **p2** and **p3** as **trim_1** and **trim_2** respectively, **sense_agreement** equals true and **master_representation** shall be implementation dependent. This **api_line** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style**with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_line** is returned.

— the given **api_line** LINNM1 is redefined with **trim_2** equals **p2,** and the given **api_line** LINNM2 is redefined with **trim_1** equals **p3.**

The values of LEN1 and LEN2, that shall be in range [EPS,MAX], are measured in *OVC_length_units* ; if an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) The given value of LEN1 shall be less the distance between the intersection point **p1** and the point **trim_1** of the **api_line** LINNM1, and the given value of LEN2 shall be less than the distance between the intersection point **p1** and the point **trim_2** of the **api_line** LINNM2. Otherwise an error occurs.

2) An **api_line** is created, only if the segment length of this **api_line** is in range [EPS,MAX] and the segment length of both given entities, LINNM1 and LINNM2, after shortening is greater than EPS. Otherwise an error occurs.

3) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is greater equal 2), both given **api_line**s shall be in the same plane.

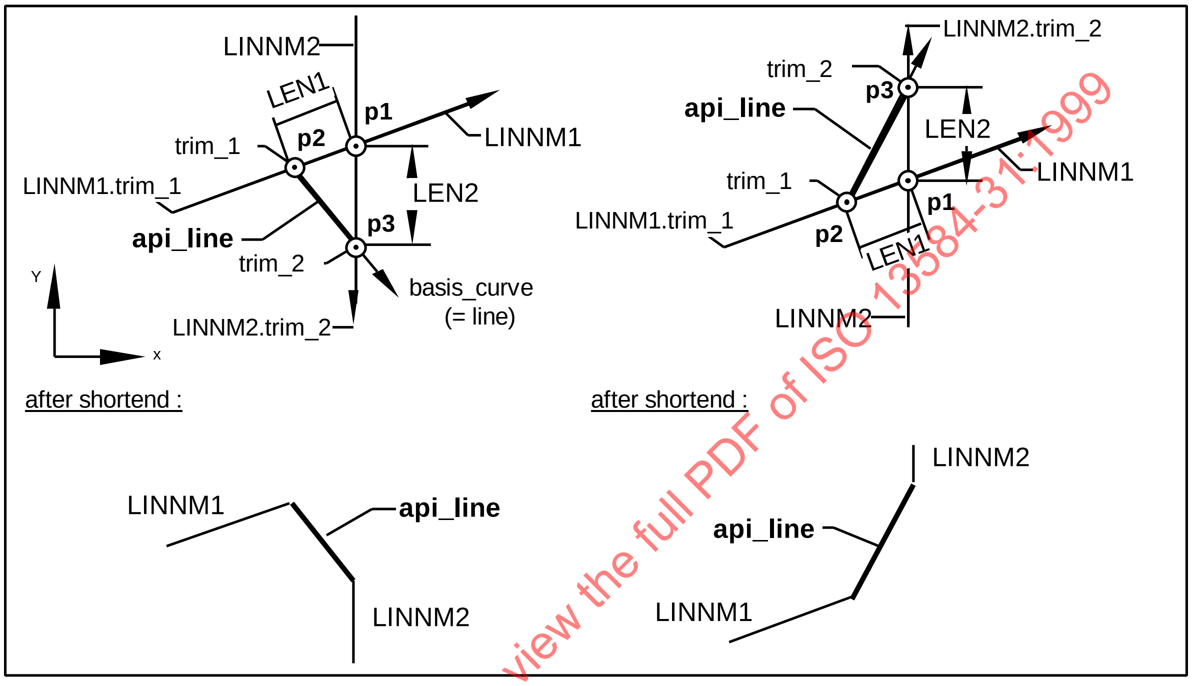


Figure A. 20 — Function: Lin_Chamfer_2_Lin

Internal Reference:
6.1.9, 6.1.12, **6.1.12.1**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	101	attempt to create a degenerated entity
105	attempt to create a degenerated direction during entity creation	108	attempt to create a degenerated basic curve during entity creation
111	attempt to create a line whose segment length is out of range [EPS,MAX]	115	given entities are identical
119	given entities are not in the same plane	120	given cut length too long
122	no intersection of given curves entities	127	geometrical design is not feasible
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.1.2 Circle and circular arcs (api_circular_arc)

Circle by radius and axis2_placement	Circle_Rad_A2p
Circular arc by three points	Arc_3_Pnt

Circular arc by radius and two angles and axis2_placement	Arc_Rad_2_Angle_A2p
Circular arc by radius and three points	Arc_Rad_3_Pnt
Circular arc by radius, two points and axis2_placement	Arc_Rad_2_Pnt_A2p
Circular arc as fillet between two entities	Arc_Fillet_2_Ent
Circular arc tangential to two entities	Arc_Tangential_2_Ent
Circular arc define by its radius and two entities	Arc_Rad_2_Ent
Circular arc defined by three entities	Arc_3_Ent

A.5.3.1.2.1 Circle by radius and axis2_placement

Function name:

Circle_Rad_A2p

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	RAD	D	Radius of **api_circular_arc**	(EPS ≤ RAD ≤ MAX)
in	A2PNAM	N	Name of **axis2_placement**	a2p
in	SENSE	E	Sense agreement flag	[TRUE,FALSE]
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **api_circular_arc**	arc

FORTRAN binding:

```
NAME = CIRCLE_RAD_A2P ( RAD,A2PNAM,SENSE,KFIX )
```

Effects:

Creates a whole circle as an **api_circular_arc** entity given by a radius (RAD), an **axis2_placement** (A2PNAM), and a sense agreement flag (SENSE) that defines the direction of the newly created **api_circular_arc** in conjunction with a **circle** entity as basis_curve.

The **axis2_placement** (A2PNAM) is duplicated as **a2p1** and it has a **null_style** assigned to it. Then:

— a **circle c** is instantiated with **position a2p1** and **radius** equals RAD. This **circle** has a **null_style** assigned to it.

— an **api_circular_arc** is instantiated with **c** as **basis_curve**, a parameter value equals 0 degree as **trim_1** and a parameter value equals 360 degrees as **trim_2**, **sense_agreement** equals SENSE and **master_representation** shall be implementation dependent. This **api_circular_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style**with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_circular_arc** is returned.

The value of RAD shall be in range [EPS,MAX] and is measured in *OVC_length_units* . If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) The interface shall ensure the closure of the created **api_circular_arc** entity (cartesian coordinates of **trim_1** equal to cartesian coordinates of **trim_2**).

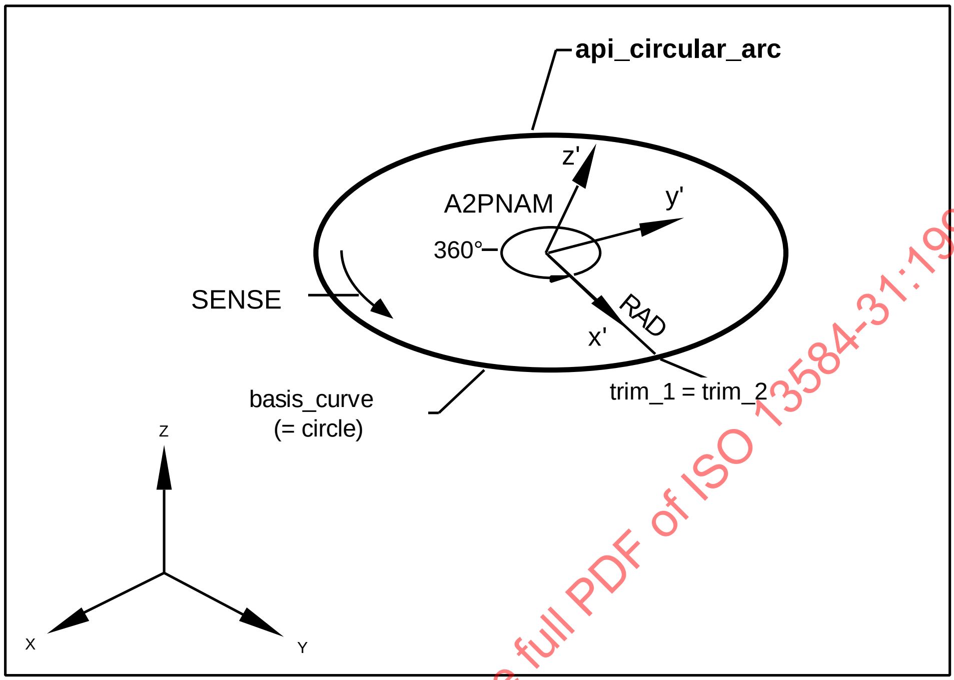


Figure A. 21 — Function: Circle_Rad_A2p

Internal Reference:

6.1.9, 6.1.12, **6.1.12.2**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	101	attempt to create a degenerated entity
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.1.2.2 Circular arc by three points

Function name:

Arc_3_Pnt

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	STAPNT	N	name of start **cartesian_point**	pnt
in	INTPNT	N	name of intermediate **cartesian_point**	pnt
in	ENDPNT	N	name of end **cartesian_point**	pnt
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **api_circular_arc**	arc

FORTRAN binding:

```
NAME = ARC_3_PNT ( STAPNT,INTPNT,ENDPNT,KFIX )
```

Effects:

Creates an **api_circular_arc** entity given by the three **cartesian_point**s (STAPNT, INTPNT and ENDPNT). Where the start **cartesian_point** (STAPNT) defines the start of the circular arc, the end **cartesian_point** (ENDPNT) defines the end of the circular arc, and the intermediate **cartesian_point** (INTPNT) is used to define both the plane and sector of the **api_circular_arc**.

The **cartesian_point**s STAPNT, ENDPNT are duplicated as **p1** and **p3** respectively and they have a **null_style** assigned to them. Let **P2** be a synonym for INTPNT. Then:

— a **cartesian_point p4** is instantiated as the centre of the circular arc, with **coordinates** derived from the cartesian coordinates calculated from the three points **p1**, **P2** and **p3**. This **cartesian_point** has a **null_style** assigned to it.

— a **direction d1** is instantiated with **direction_ratio**s defined by **p1-p4**. This **direction** has a **null_style** assigned to it.

In the case of a 3D view:

— a **direction d0** is instantiated with **direction_ratio**s calculated through the result of a vectorial cross product of a vector defined by **p1-p3** and a vector defined by **P2-p3**. This **direction** has a **null_style** assigned to it.

— an axis2_placement_3d a2p1 is instantiated with location p4 and axis d0 and ref_direction d1. This axis2_placement_3d has a null_style assigned to it.

In the case of a 2D view:

— an axis2_placement_2d a2p1 is instantiated with location derived from the cartesian coordinates of p4 and ref_direction d1. This axis2_placement_2d has a null_style assigned to it.

Then:

— a **circle c** is instantiated with **position** derived from the **axis2_placement a2p1** and **radius** derived from ||**p1-p4**||. This **circle** has a **null_style** assigned to it.

— an **api_circular_arc** is instantiated with **c** as **basis_curve**, **p1** and **p3** as **trim_1** and **trim_2** respectively, **sense_agreement** derived from the position of the **cartesian_point**s **p1**, **P2** and **p3**, and **master_representation** shall be implementation dependent. This **api_circular_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_circular_arc** is returned.

The distance between two of the three points shall not be in the range[ZERO_VALUE,EPS]. Additionally the intermediate point INTPNT shall not be collinear with the start and end points within a range of EPS. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) The **api_circular_arc** is created, only if its calculated radius is in range [EPS,MAX], and its segment length from **trim_1** to **trim_2**, consistent with its **sense_agreement**, is not less than EPS. Otherwise an error occurs.

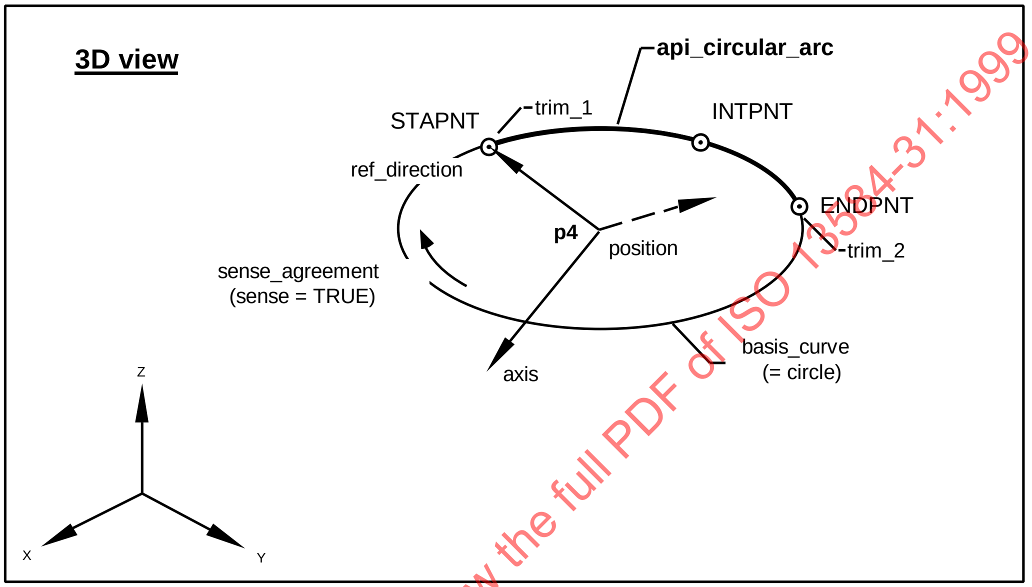


Figure A. 22 — Function: Arc_3_Pnt (in a 3D view)

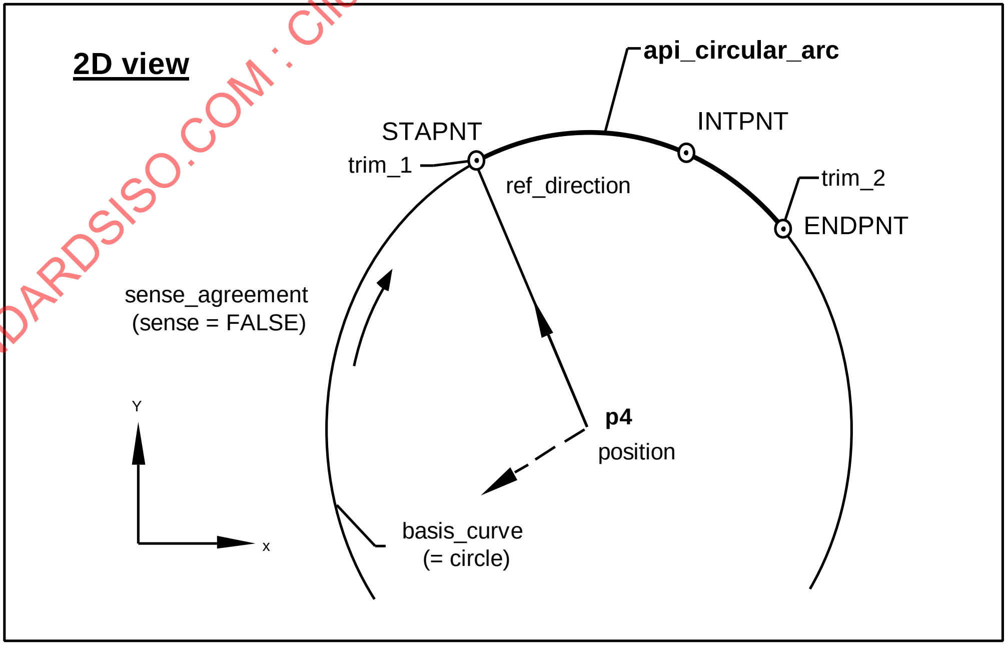


Figure A. 23 — Function: Arc_3_Pnt (in a 2D view)

Internal Reference:
6.1.9, 6.1.12, **6.1.12.2**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
101	attempt to create a degenerated entity	105	attempt to create a degenerated direction during entity creation
106	attempt to create a degenerated axis2_placement during entity creation	112	attempt to create an arc whose segment length is less than EPS
115	given entities are identical	116	given points are linear dependent
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.1.2.3 Circular arc by radius, two angles and axis2_placement

Function name:
Arc_Rad_2_Angle_A2p

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	RAD	D	radius of **api_circular_arc**	(EPS ≤ RAD ≤ MAX)
in	STAPNT	D	start angle in (Oxy) plane relative to (Ox) axis of given a2p	(0° ≤ STAANG ≤ 360°)
in	ENDANG	D	end angle in (Oxy) plane relative to (Ox) axis of given a2p	(0° ≤ ENDANG ≤ 360°)
in	A2PNAM	N	name of **axis2_placement**	a2p
in	SENSE	E	sense agreement flag	[TRUE,FALSE]
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **api_circular_arc**	arc

FORTRAN binding:

```
NAME = ARC_RAD_2_ANGLE_A2P (RAD,STAANG,ENDANG,A2PNAM,SENSE,KFIX )
```

Effects:
Creates an **api_circular_arc** entity given by a radius (RAD), two angles, an **axis2_placement** (A2PNAM), and a sense agreement flag (SENSE) that defines the direction of the newly created **api_circular_arc** in conjunction with a **circle** entity as **basis_curve**, together with a start and an end point that are implicitly defined by the two angles STAANG and ENDANG respectively.

The **axis2_placement** (A2PNAM) is duplicated as **a2p1** and it has a **null_style** assigned to it. Then:

— a **circle c** is instantiated with **position a2p1** and **radius** equals RAD. This **circle** has a **null_style** assigned to it.

— an **api_circular_arc** is instantiated with **c** as **basis_curve**, the parameter value STAANG as **trim_1** and the parameter value ENDANG as **trim_2**, **sense_agreement** equals SENSE and **master_representation** shall be implementation dependent. This **api_circular_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style**with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_circular_arc** is returned.

The value of RAD shall be in range [EPS,MAX] and is measured in *OVC_length_units* ; the given angles are measured in *OVC_angle_units*, counted in (Oxy) plane of given A2PNAM. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) The **api_circular_arc** is created, only if its calculated radius is in range [EPS,MAX], and its segment length from **trim_1** to **trim_2**, consistent with its **sense_agreement**, is not less than EPS. Otherwise an error occurs.

2) If the two parameter values, STAANG and ENDANG, define the same point in a range of [ZERO_VALUE,EPS], then both trimming points **trim_1** and **trim_2** are identical and the created circular arc is a complete circle. Then, the interface ensures the closure of the created **api_circular_arc** entity (cartesian co-ordinates of **trim_1** equal to cartesian co-ordinates of **trim_2**).

3) The **api_circular_arc** is created, only if its segment length from **trim_1** to **trim_2**, constant with **sense_agreement**, is not less than EPS. Otherwise an error occurs.

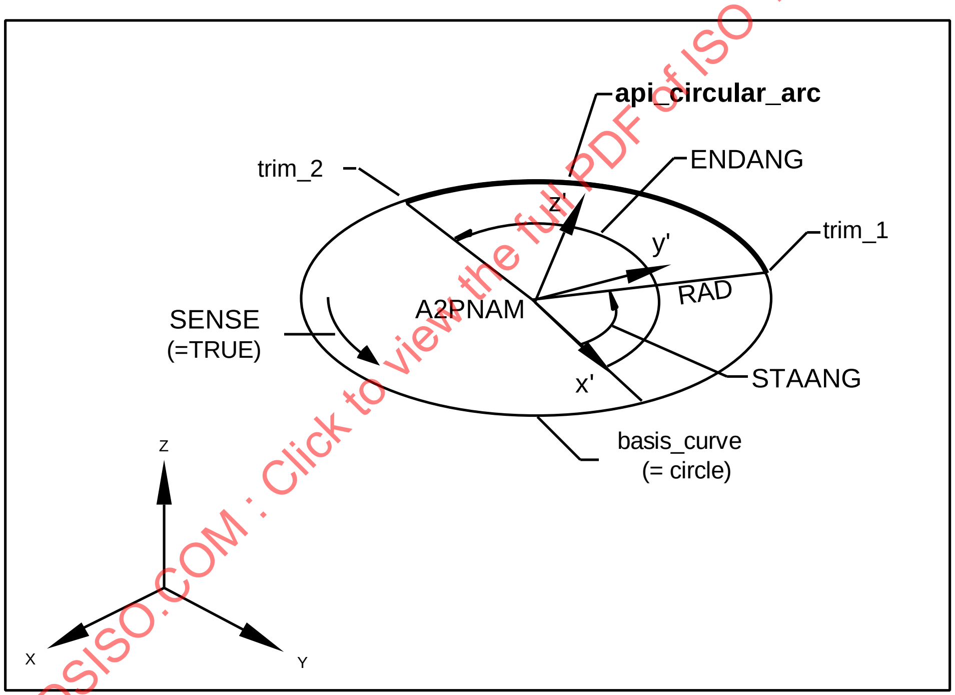


Figure A. 24 — Function: Arc_Rad_2_Angle_A2p

Internal Reference:
6.1.9, 6.1.12, **6.1.12.2**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	4	value for plane angle measure out of permitted range
101	attempt to create a degenerated entity	112	attempt to create an arc whose segment length is less than EPS
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.1.2.4 Circular arc by radius and three points

Function name:

Arc_Rad_3_Pnt

Interface level:	**1**
Geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	RAD	D	radius of **api_circular_arc**	(EPS ≤ RAD ≤ MAX)
in	STAPNT	N	name of start **cartesian_point**	pnt
in	ENDPNT	N	name of end **cartesian_point**	pnt
in	HLPPNT	N	name of help **cartesian_point**	pnt
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **api_circular_arc**	arc

FORTRAN binding:

```
NAME = ARC_RAD_3_PNT (RAD,STAPNT,ENDPNT,HLPPNT,KFIX)
```

Effects:

Creates an **api_circular_arc** entity given by a radius (RAD) and three **cartesian_point**s (STAPNT, ENDPNT and HLPPNT). Where the start **cartesian_point** (STAPNT) defines the start of the circular arc, the end point (ENDPNT) defines the end of the circular arc, and the help point (HLPPNT) is used to define both the plane and sector of the **api_circular_arc**.

The points STAPNT, ENDPNT, HLPPNT are duplicated as **p1**, **p2, p3** respectively and they have a **null_style** assigned to them. Then:

In the case of a 3D view:

— a **direction d0** is instantiated with **direction_ratio**s calculated through the result of a vectorial cross product of the vector defined by **p1-p3** and the vector defined by **p2-p3**. This **direction** has a **null_style** assigned to it.

— a virtual plane is defined by **cartesian_point p3** and the **direction d0**, that is a direction normal to the virtual plane.

— two **cartesian_point**s **p4** and **p5** are instantiated with **coordinates** derived from the cartesian coordinates calculated by the intersections of two virtual circles in the previously defined virtual plane, with radius RAD, and centre points **p1** and **p2** respectively. The **cartesian_point**s **p4** and **p5** have a **null_style** assigned to them.

— from the two **cartesian_point**s **p4** and **p5**, the point that is located nearer to point **p3** is chosen as the centre point for the new **api_circular_arc**, and renamed as **p6**.

— a **direction d1** is instantiated with **direction_ratio**s defined by **p1-p6**. This **direction** has a **null_style** assigned to.

— an axis2_placement_3d a2p1 is instantiated with location p6 and axis d0 and ref_direction d1. This axis2_placement_3d has a null_style assigned to it.

In the case of a 2D view:

— two **cartesian_point**s **p4** and **p5** are instantiated with **coordinates** derived from the cartesian coordinates calculated by the intersections of two virtual circles in the (Oxy) plane of the current OVC reference system, with radius RAD, and centre **cartesian_point**s **p1** and **p2** respectively. These **cartesian_point**s have a **null_style** assigned to it.

— from the two points **p4** and **p5**, the point that is located nearer to point **p3** is chosen as the centre point for the new **api_circular_arc**, and renamed as **p6**.

— a **direction d1** is instantiated with **direction_ratio**s defined by **p1-p6**. This **direction** has a **null_style** assigned to it.

— an **axis2_placement_2d a2p1** is instantiated with **location p6** and **ref_direction d1**. This **axis2_placement_2d** has a **null_style** assigned to it.

Then:

— a **circle c** is instantiated with **position a2p1** and **radius** equals RAD. This **circle** has a **null_style** assigned to it.

— an **api_circular_arc** is instantiated with **c** as **basis_curve**, **p1** and **p2** as **trim_1** and **trim_2** respectively, **sense_agreement** derived from point **p3** defining the valid sector of the circle, and **master_representation** shall be implementation dependent. This **api_circular_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table, too. The name of this **api_circular_arc** is returned.

The distance between two of the three points shall not be in the range [ZERO_VALUE,EPS], additionally the help point **p3** shall not be placed within a distance of EPS to the centre of the circular arc. The value of RAD shall be in range [EPS,MAX] and is measured in *OVC_length_units* ; if an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) The **api_circular_arc** is created, only if its segment length from **trim_1** to **trim_2**, consistent with its **sense_agreement**, is not less than EPS. Otherwise an error occurs.

2) The **api_circular_arc** is created, only if the help point HLPPNT defines unique one sector (the calculated distance between computed centre point and this HLPPNT is not equal to RAD. Otherwise an error occurs.

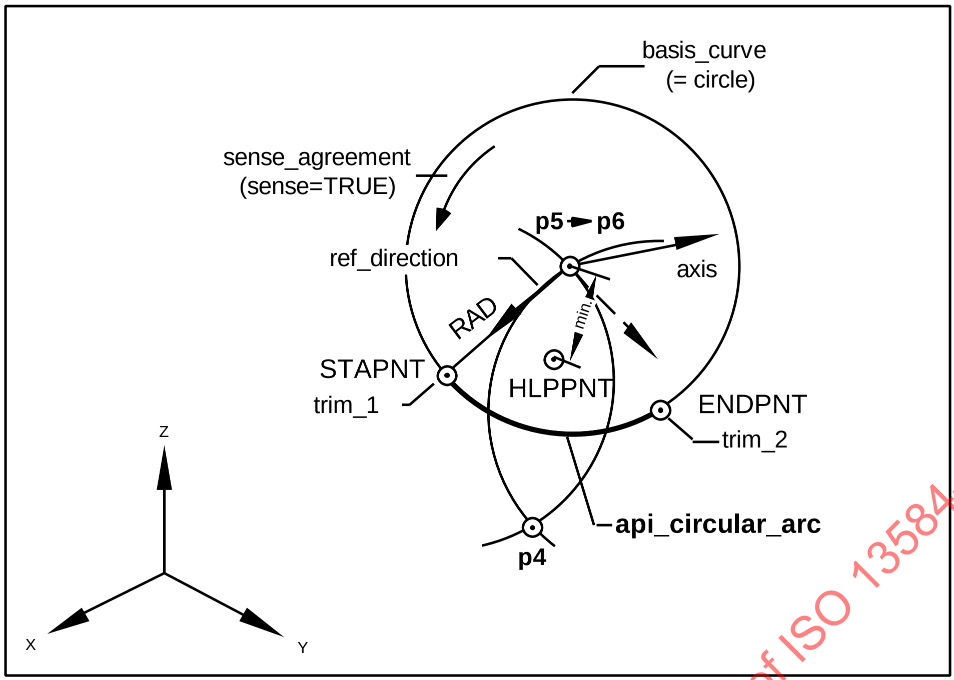


Figure A. 25 — Function: Arc_Rad_3_Pnt

Internal Reference:
6.1.9, 6.1.12, **6.1.12.2**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	101	attempt to create a degenerated entity
105	attempt to create a degenerated direction during entity creation	106	attempt to create a degenerated axis2_placement during entity creation
112	attempt to create an arc whose segment length is less than EPS	115	given entities are identical
116	given points are linear dependent	127	geometrical design is not feasible
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.1.2.5 Circular arc by radius, two points and axis2_placement

Function name:
Arc_Rad_2_Pnt_A2p

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	RAD	D	radius of **api_circular_arc**	(EPS ≤ RAD ≤ MAX)
in	PNTNM1	N	name of first **cartesian_point**	pnt
in	PNTNM2	N	name of second **cartesian_point**	pnt
in	A2PNAM	N	name of **axis2_placement**	a2p
in	SENSE	E	sense agreement flag	[TRUE,FALSE]
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **api_circular_arc**	arc

FORTRAN binding:

```
NAME = ARC_RAD_2_PNT_A2P (RAD,PNTNM1,PNTNM2,A2PNAM,SENSE,KFIX )
```

Effects:

Creates an **api_circular_arc** entity given by a radius (RAD), two **cartesian_point**s, an **axis2_placement** (A2PNAM), and a sense agreement flag (SENSE) that defines the direction of the newly created **api_circular_arc** in conjunction with a **circle** entity as **basis_curve**, together with a start and an end point that are implicitly defined by the two points PNTNM1 and PNTNM2 respectively.

The A2PNAM is duplicated as **a2p1** and has a **null_style** assigned to it, and let **P1** and **P2** be a synonym for the **cartesian_point**s PNTNM1 and PNTNM2 respectively. Then:

— a **circle c** is instantiated with **position a2p1** and **radius** equals RAD. This **circle** has a **null_style** assigned to it.

— a **cartesian_point p3** is instantiated that **coordinates** are derived from the **cartesian_coordinate**s of **location** of **a2p1**. This **cartesian_point** is the centre point of the circle and it has a **null_style** assigned to it.

— a **direction d1** is instantiated with **direction_ratio**s defined by **P1-p3**. This **direction** has a **null_style** assigned to it.

— a **vector v1** is instantiated with **orientation d1** and the **magnitude** equals **||P1-p3||**. This **vector** has a **null_style** assigned to it.

— a **line l1** is instantiated with **pnt p3** and **dir v1**. This **line** has a **null_style** assigned to.

— a **direction d2** is instantiated with **direction_ratio**s defined by **P2-p3**. This **direction** has a **null_style** assigned to it.

— a **vector v2** is instantiated with **orientation d2** and the **magnitude** equals **||P2-p3||**. This **vector** has a **null_style** assigned to it.

– a **line l2** is instantiated with **pnt p3** and **dir v2**. This **line** has a **null_style** assigned to it.

– a **cartesian_point p4** is instantiated that **coordinates** are calculated by an intersection of line **l1** with **circle c** in the positive direction of **line l1**. This **cartesian_point** has a **null_style** assigned to it.

– a **cartesian_point p5** is instantiated that **coordinates** are calculated by an intersection of line **l2** with **circle c** in the positive direction of **line l2**. This **cartesian_point** has a **null_style** assigned to it.

– an **api_circular_arc** is instantiated with **c** as **basis_curve**, **p4** and **p5** as **trim_1** and **trim_2** respectively, **sense_agreement** equals SENSE and **master_representation** shall be implementation dependent. This **api_circular_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_circular_arc** is returned.

The distance between the centre of the **api_circular_arc** and the two given **cartesian_point**s shall not be less than EPS. Additionally the distance between the two given **cartesian_point**s shall not be less than EPS. The value of RAD shall be in the range [EPS,MAX] and is measured in *OVC_length_units*. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) If the two implicit defined **directions d1** and **d2** are the identical in a range [ZERO_VALUE,EPS], then both trimming points **trim_1** and **trim_2** are identical and the created circular arc is a complete circle. Then, the interface ensures the closure of the created **api_circular_arc** entity (cartesian coordinates of **trim_1** equal to cartesian coordinates of **trim_2**).

2) The **api_circular_arc** is created, only if its segment length from **trim_1** to **trim_2**, consistent with its **sense_agreement**, is not less than EPS. Otherwise an error occurs.

3) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is greater equal 2), the points **P1** and **P2** shall be in the (Oxy) plane of the local coordinate system (A2PNAM).

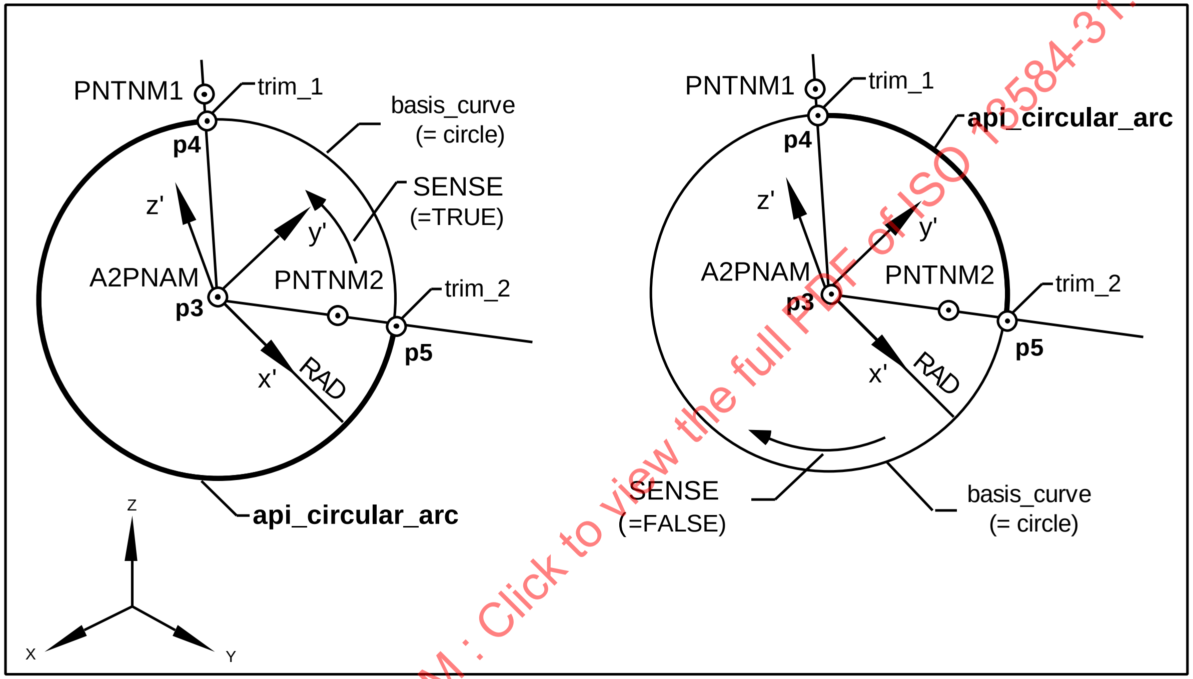


Figure A. 26 — Function: Arc_Rad_2_Pnt_A2p

Internal Reference:

6.1.9, 6.1.12, **6.1.12.2**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	101	attempt to create a degenerated entity
105	attempt to create a degenerated direction during entity creation	112	attempt to create an arc whose segment length is less than EPS
119	given entities are not in the same plane	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.3.1.2.6 Circular arc as fillet between two entities

Function name:

Arc_Fillet_2_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	ENTNM1	N	name of first entity	basic
in	ENTNM1	N	name of second entity	basic
in	RAD	D	radius of fillet	(EPS ≤ RAD ≤ MAX)
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	name of created **api_circular_arc**	arc

FORTRAN binding:

```
NAME = ARC_FILLET_2_ENT ( ENTNM1,ENTNM2,RAD,KFIX )
```

Effects:

Creates an **api_circular_arc** entity as fillet between two basic entities with a radius RAD. They may or may not intersect. Both existing entities, ENTNM1 and ENTNM2, will be shortened and the names of these entities are not changed. They remain in the temporary database until they are explicitly fixed. The newly created **api_circular_arc** starts at the new end of the first entity ENTNM1, that is the new trimming point **trim_2** of ENTNM1 and ends at the new beginning of the second entity ENTNM2, that is the new trimming point **trim_1** of ENTNM2. The orientation of this **api_circular_arc** will be consistent with one another.

The following calculation process will be performed by the interface:

— All geometrical possible tangential **circle**s **cir$_i$** with radius RAD, between both given entities ENTNM1 and ENTNM2 are virtual calculated, with centre points as instances of **cartesian_point pc1$_i$** , tangential points on ENTNM1 as instances of **cartesian_point pt1$_i$** and tangential points on ENTNM2 as instances of **cartesian_point pt2$_i$**. They are numbered by the index **i (i=1,...n),** with **n** equals the max. number of possible solutions. All these created instances have a **null_style** assigned to them.

If these calculations fail, it means the geometrical design is not feasible (**n**=0), or if the given radius RAD for the tangential arc is too big or too small, then:

— an error occurs

— no entity is created and no modification of the existing entities will be done

– the element name returned as zero

If the first given basic entity ENTNM1 is an instance of type **api_line**:

— **n axis2_placement**s **a2p$_i$** are instantiated with **location** equals **pc1$_i$** and the directions of placement axes derived from the common plane of both given entities. All these instances have a **null_style** assigned to them.

— **n circle**s **c$_i$** are instantiated with **position** equals **a2p$_i$** and **radius** RAD. All these instances have a **null_style** assigned to them.

— **n api_circular_arc**s **a$_i$** are instantiated with **c$_i$** as **basis_curve**, **pt1$_i$** as **trim_1** and **pt2$_i$** as **trim_2**, **sense_agreement** equals true, if a direction derived from **trim_2 - trim_1** of the **api_line** (ENTNM1) is equal to the tangential direction at the point **pt1$_i$** on the **basis_curve c$_i$**, otherwise false. The **master_representation** shall be implementation dependent. All these instances have a **null_style** assigned to them.

— a selection process is started and ensures that there is only one solution for the newly created **api_circular_arc**. One of **a$_i$** instances are chosen, where:

- — the tangential direction at the point pt2i on the api_circular_arc ai is equal to the orientation derived from the second entity ENTNM2 (with respect to the sense_agreement_flags).
- — the **api_circular_arc a$_i$** has the smallest radial angle
- — the calculated distance between trimming point **trim_1** of the first entity ENTNM1 and trimming point **trim_1** of the new created **api_circular_arc a$_i$** has the shortest length.

— the chosen **api_circular_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_circular_arc** is returned.

If the first given basic entity ENTNM1 is an instance of type **api_circular_arc**:

— **n axis2_placement**s **a2p$_i$** are instantiated with **location** equals **pc1$_i$** and axes directions derived from **position** of the **basis_curve** from ENTNM1. All these instances have a **null_style** assigned to them.

— **n circle**s **c$_i$** are instantiated with **position** equals **a2p$_i$** and **radius** RAD. All these instances have a **null_style** assigned to them.

— **n api_circular_arc**s **a$_i$** are instantiated with **c$_i$** as **basis_curve**, **pt1$_i$** as **trim_1** and **pt2$_i$** as **trim_2** and if a direction calculated through **pt1$_i$ - pc1$_i$** is equal to a direction calculated through **pt1$_i$ - ENTNM1.basis_curve.position.location**, then **sense_agreement** equals **sense_agreement** from the **api_circular_arc** (ENTNM1), otherwise the **sense_agreement** is the opposite of **sense_agreement** from the **api_circular_arc** (ENTNM1). The **master_representation** shall be implementation dependent. All these instances have a **null_style** assigned to them.

— a selection process is started and ensures only one solution for the newly created **api_circular_arc**. Then one of **a$_i$** instances are chosen, where:

- — the tangential direction at the point **pt2$_i$** on the **api_circular_arc a$_i$** is equal to the orientation derived from the second entity ENTNM2 (with respect to the **sense_agreement_flag**s**)**.
- — the **api_circular_arc a$_i$** has the smallest radial angle
- — the calculated arc length between trimming point **trim_1** of the first entity ENTNM1 and trimming point **trim_1** of the **api_circular_arc a$_i$** , measured from **trim_1** in direction to **trim_2** of ENTNM1, has the shortest length.

— the chosen **api_circular_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_circular_arc** is returned.

– the given instances of ENTNM1 is redefined with **trim_2** equals **trim_1** form the chosen **api_circular_arc** and ENTNM2 is redefined with **trim_1** equals **trim_2** form the chosen **api_circular_arc**.

The value of RAD shall be in range [EPS,MAX] and is measured in *OVC_length_unit*s; if an error occurs, no entity is created and no modification of the existing entities will be done. The element name is returned as zero.

Notes:

1) Parallel **api_line**s or concentric **api_circular_arc**s are not permitted as given entities ENTNM1 and ENTNM2.

2) The created **api_circular_arc** will be less than or equal to its semicircle.

3) The **api_circular_arc** is created, only if its segment length from **trim_1** to **trim_2**, consistent with its **sense_agreement**, is not less than EPS, and the segment length of both given entities, ENTNM1 and ENTNM2, after shortening is greater than EPS. Otherwise an error occurs.

4) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is greater equal 2), both given entities ENTNM1 and ENTNM2 shall be in the same plane.

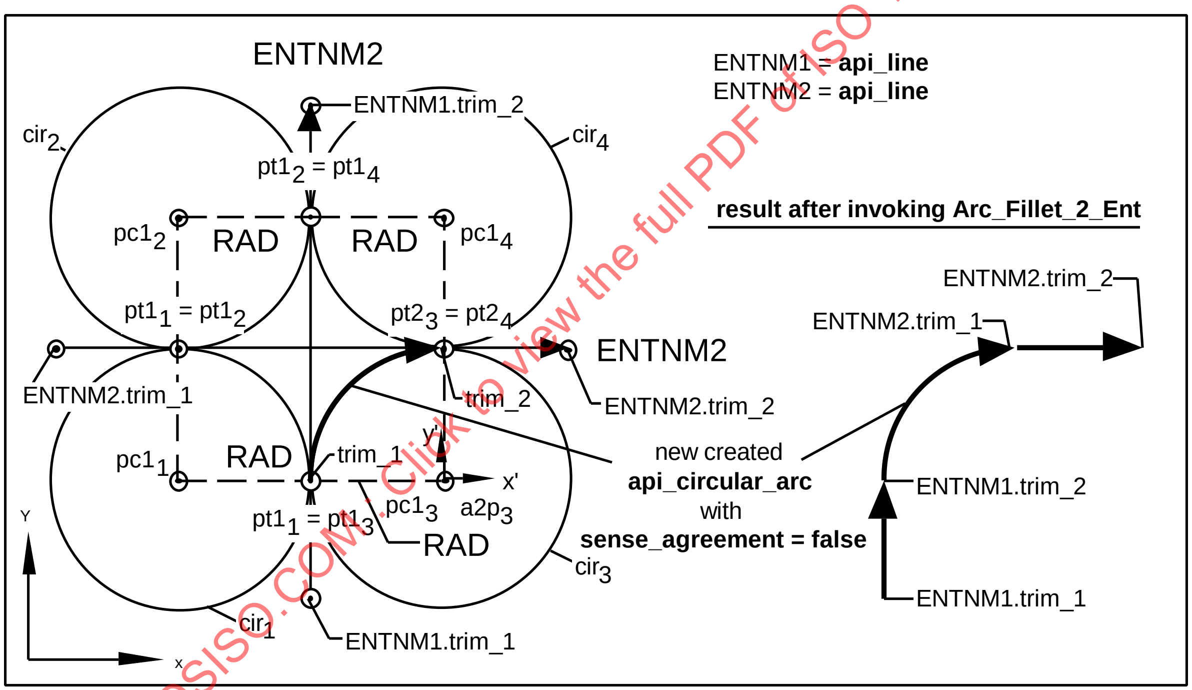


Figure A. 27 — Function: Arc_Fillet_2_Ent (lin/lin)

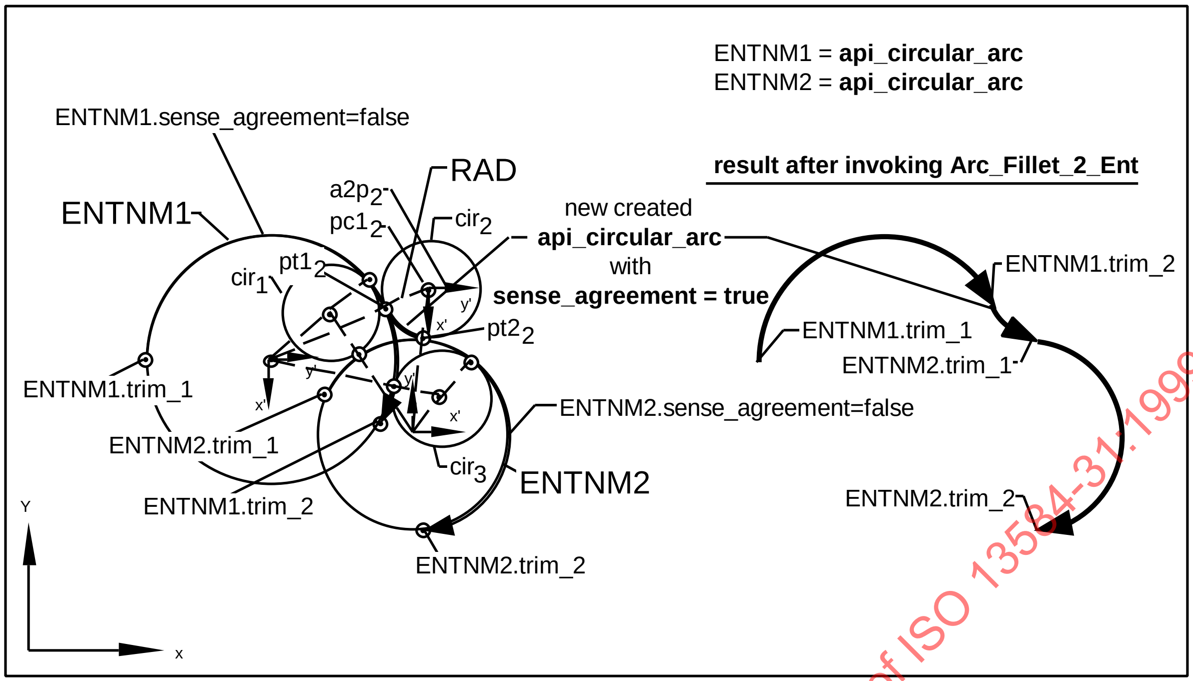


Figure A. 28 — Function: Arc_Fillet_2_Ent (arc/arc)

Internal Reference:
6.1.9, 6.1.12, **6.1.12.2**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	101	attempt to create a degenerated entity
106	attempt to create a degenerated axis2_placement during entity creation	108	attempt to create a degenerated basic curve during entity creation
112	attempt to create an arc whose segment length is less than EPS	115	given entities are identical
118	given curves entities are parallel/concentric	119	given entities are not in the same plane
121	radius too big/small	127	geometrical design is not feasible
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.1.2.7 Circular arc tangential to two entities

Function name:
Arc_Tangential_2_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	meaning	permitted types/values
in	ENTNM1	N	name of first entity	Basic
in	ENTNM1	N	name of second entity	Basic
in	RAD	D	radius of tangential **api_circular_arc**	(EPS ≤ RAD ≤ MAX)
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **api_circular_arc**	Arc

FORTRAN binding:

```
NAME = ARC_TANGENTIAL_2_ENT ( ENTNM1,ENTNM2,RAD,KFIX )
```

Effects:

Creates an **api_circular_arc** entity as a tangential arc between two basic entities with a radius RAD. Both existing entities, ENTNM1 and ENTNM2 may or may not intersect. The orientation of this **api_circular_arc** will be consistent with one another.

The following calculation process will be performed by the interface:

— In the following, we will consider **E1**, **E2** as synonym for the given entities ENTNM1 and ENTNM2 respectively.

— All geometrical possible tangential **circle**s **cir**$_i$ with radius RAD, between both entities **E1** and **E2** are virtual calculated, with centre points as instances of **cartesian_point pc1**$_i$, tangential points on the **basis_curve** of **E1** as instances of **cartesian_point pt1**$_i$ and tangential points on the **basis_curve** of **E2** as instances of **cartesian_point pt2**$_i$. They are numbered by the index **i** (**i=1,...n),** with **n** equal to the max. number of possible solutions. All these created instances have a **null_style** assigned to them.

If these calculations fail, it means the geometrical design is not feasible (n=0), or if the given radius RAD for the tangential arc is too big or too small, then:

- an error occurs
- the element name returned as zero

If the first basic entity **E1** is an instance of type **api_line**:

- **axis2_placement**s **a2p**$_i$ are instantiated with **location** equals **pc1**$_i$ and the directions of placement axes derived from the common plane of both given entities. All these instances have a **null_style** assigned to them.
- **n circle**s **c**$_i$ are instantiated with **position** equals **a2p**$_i$ and **radius** RAD. All these instances have a **null_style** assigned to them.
- **n api_circular_arc**s **a**$_i$ are instantiated with **c**$_i$ as **basis_curve**, **pt1**$_i$ as **trim_1** and **pt2**$_i$ as **trim_2**, **sense_agreement** equals true, if a direction derived from **trim_2 - trim_1** of the **api_line** (**e1**) is equal to the tangential direction at the point **pt1**$_i$ on the **basis_curve c**$_i$, otherwise false. The **master_representation** shall be implementation dependent. All these instances have a **null_style** assigned to them.

— a selection process is started and ensures only one solution for the newly created **api_circular_arc**. Then one of $\mathbf{a_i}$ instances are chosen, where:

  — the tangential direction at the point pt2i on the api_circular_arc ai is equal to the orientation derived from the second entity **E2** (with respect to the **sense_agreement_flag**s)**.**

  — the **api_circular_arc** $\mathbf{a_i}$ has the smallest radial angle

  — **trim_1** and **trim_2** of **api_circular_arc** $\mathbf{a_i}$ are lies within the parametric range [**trim_1**, **trim_2**] of both given entities **E1** and **E2**.

  — the calculated distance between trimming point **trim_1** of the first entity **E1** and trimming point **trim_1** of the new created **api_circular_arc** $\mathbf{a_i}$ has the shortest length.

— this chosen **api_circular_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_circular_arc** is returned.

If the first basic entity **E1** is an instance of type **api_circular_arc**:

— **n axis2_placement**s $\mathbf{a2p_i}$ are instantiated with **location** equals $\mathbf{pc1_i}$ and axes directions derived from **position** of the **basis_curve** from **E1**. All these instances have a **null_style** assigned to them.

— **n circle**s $\mathbf{c_i}$ are instantiated with **position** equals $\mathbf{a2p_i}$ and **radius** RAD. All these instances have a **null_style** assigned to them.

— **n api_circular_arc**s $\mathbf{a_i}$ are instantiated with $\mathbf{c_i}$ as **basis_curve**, $\mathbf{pt1_i}$ as **trim_1** and $\mathbf{pt2_i}$ as **trim_2**. If a direction calculated through $\mathbf{pt1_i}$ - $\mathbf{pc1_i}$ is equal to a direction calculated through $\mathbf{pt1_i}$ - **ENTNM1.basis_curve.position.location**, then **sense_agreement** is set to **sense_agreement** from the **api_circular_arc** (**E1**), otherwise **sense_agreement** is the opposite of **sense_agreement** from the **api_circular_arc** (**E1**). The **master_representation** shall be implementation dependent. All these instances have a **null_style** assigned to them.

— a selection process is started and ensures only one solution for the new created **api_circular_arc**. Then one of $\mathbf{a_i}$ instances are chosen, where:

  — the tangential direction at the point $\mathbf{pt2_i}$ on the **api_circular_arc** $\mathbf{a_i}$ is equal to the orientation derived from the second entity **E2** (with respect to the **sense_agreement_flag**s)**.**

  — the **api_circular_arc** $\mathbf{a_i}$ has the smallest radial angle

  — **trim_1** and **trim_2** of **api_circular_arc** $\mathbf{a_i}$ are lies within the parametric range [**trim_1**, **trim_2**] of both given entities **E1** and **E2**.

  — the calculated arc length between trimming point **trim_1** of the first entity **E1** and trimming point **trim_1** of the **api_circular_arc** $\mathbf{a_i}$, measured from **trim_1** in direction to **trim_2** of **E1**, has the shortest length.

— this chosen **api_circular_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style**with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_circular_arc** is returned.

The value of RAD shall be in range [EPS,MAX] and is measured in *OVC_length_units*; if an error occurs, no entity is created and no modification of the existing entities will be done. The element name is returned as zero.

Notes:

1) Parallel **api_line**s or concentric **api_circular_arc**s are not permitted as given entities **E1** and **E2**.

2) The created **api_circular_arc** will be less than or equal to its semicircle.

3) If the distance between the chosen tangential points, for creating the new **api_circular_arc**, and one of trimming points (**trim_1** or **trim_2**) of the corresponding given **api_circular_arc**s is in range [ZERO_VALUE,EPS], no error occurs and the coordinates derived from this trimming point will be used instead of the calculated one.

4) The **api_circular_arc** is created, only if its segment length from **trim_1** to **trim_2**, consistent with its **sense_agreement**, is not less than EPS.

5) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is greater equal 2), both given entities **E1** and **E2** shall be in the same plane.

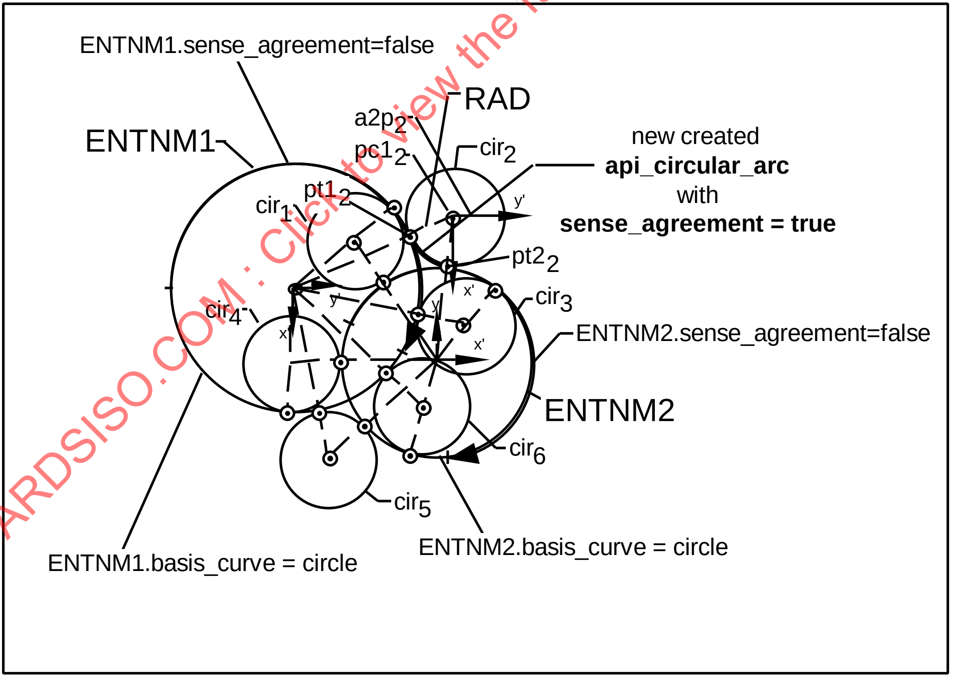


Figure A. 29 — Function: Arc_Tangential_2_Ent

Internal Reference:

6.1.9, 6.1.12, **6.1.12.2**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	101	attempt to create a degenerated entity
106	attempt to create a degenerated axis2_placement during entity creation	110	attempt to create a point outside the parametric range of curves entity
112	attempt to create an arc whose segment length is less than EPS	115	given entities are identical
118	given curves entities are parallel/concentric	119	given entities are not in the same plane
121	radius too big/small	127	geometrical design is not feasible
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.1.2.8 Circular arc defined by its radius and two entities

Function name:

Arc_Rad_2_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	RAD	D	radius of tangential **api_circular_arc**	(EPS ≤ RAD ≤ MAX)
in	ENTNM1	N	name of first entity	basic,pnt
in	ENTNM1	N	name of second entity	basic,pnt
in	IN1	E	relative position of NAME to ENTNM1	[TRUE,FALSE]
in	IN2	E	relative position of NAME to ENTNM2	[TRUE,FALSE]
in	MINLEN	E	choice of arc length	[TRUE,FALSE]
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **api_circular_arc**	arc

FORTRAN binding:

```
NAME = ARC_RAD_2_ENT( RAD,ENTNM1,ENTNM2,IN1,IN2,MINLEN,KFIX)
```

Effects:

In the following, we will consider **E1** and **E2** as synonym for ENTNM1 and ENTNM2.

This function creates an **api_circular_arc** defined by two constraints with two entities **E1** and **E2**. These entities are either a **cartesian_point**, an **api_line**, or an **api_circular_arc**, but cannot both be **cartesian_points**. When one of these entities is a **cartesian_point**, the related constraint is the association of this point to the NAME **api_circular_arc**; else the entity is an **api_line** or an **api_circular_arc**, and the related constraint is the tangency of NAME with the **basis_curve** (respectively **line** and **circle**) of this entity, with the so-called "pulleys dependency" (same orientation of the two **trimmed_curve**s at tangential points). The three Boolean parameters MINLEN, IN1 and IN2 are meant to solve cases of ambiguity, by specifying the arc length and the relative position of the required solution, relatively (inside or outside) to each possibly **api_circular_arc** parameter.

First, the construction plane is associated an **R ref_direction** and, in 3D case, an orthogonal orientation **Z axis_direction** (in 2D case, there is no Z direction). These directions are used to define the parametrisation of the wanted **api_circular_arc**, and are computed as follow:

— if the current open view is defined as a 2D view, then **R** is the **ref_direction** of the OVC Reference System.

— else (3D case), if **E1** or **E2** is an **api_circular_arc**, then let **E** be the first **api_circular_arc** of the list (**E1**, **E2**) in this order.

Then **R** = **E.basis_curve.position.ref_direction**, and **Z** = **E.basis_curve.position.axis .**

— else (3D case, and no **api_circular_arc**s), if **E1**, **E2** are **cartesian_point**s, then an error is raised (a plane cannot be defined).

— else (3D case, no **api_circular_arc**s and at least one **api_line**), let **L** be the first **api_line** of (**E1**,**E2**) in this order, **F** the remaining entities. Let **O** be the origin point of **L**. If F is a **cartesian_point**, then let **G = F**, else **F** is an **api_line** and, if its origin does not belong to the **basis_curve** of **L**, then let **P** be the origin of **F**, else let **P** be its extremity.

Then **R = L.dir.orientation** and **Z = L.dir.orientation ´ OP** (where ´ is the cross product).

Let **S** be a set of **api_circular_arc A**, that are such that:

— if **E1** is a **cartesian_point**, then let **p1** be the duplication of that point. Else, let **p1** be the tangential point between **A** and the **basis_curve** of **E1**.

— if **E2** is a **cartesian_point**, then let **p2** be the duplication of that point. Else, let **p2** be the tangential point between **A** and the **basis_curve** of **E2**.

— p1, p2 are computed by the interface and are assigned a **null_style**.

— for each i in [1..2], if **E_i** is an **api_circular_arc**, then if IN_i =TRUE, then A is inside **E_i**; else (IN_i=FALSE) A is outside **E_i**. If **E_i** is not an **api_circular_arc**, then IN_i is not used for the determination of **A**.

— p1 is the starting trimming point of **A** (i.e A.**trim_1**[1]) and **p2** is its end trimming point (i.e. **A.trim_2**[1]).

— The following is meant to formally define an intuitive notion of pulleys dependencies between an oriented arc tangent to **E1** and **E2** (the tangent vectors shall be coherent):

A.**sense_agreement** is such that:

- — let **BA** be the **basis_curve** of **A**.
- — let **BE_i** be the **basis_curve** of **E_i**, if **E_i** is not a **cartesian_point**.
- — let **P_i** be the tangential point between **BA** and **BE_i**.
- — let **Va_i** be the vector tangent to **BA** at **P_i**.
- — let **VA_i** be **Va_i** if A.**sense_agreement** is true, **-Va_i** otherwise.
- — let **Ve_i** be the vector tangent to **BE_i** at **P_i**.
- — let **VE_i** be **Ve_i** if E_i.**sense_agreement** is true, **-Ve_i** otherwise.

Then for each i such that **E_i** is not a **cartesian_point**, **VA_i** and **VE_i** have the same direction.

There are at most 2 elements in **S**. If there is one element in **S**, let NAME be that element. Else, if there is two elements, then, if MINLEN is TRUE, then NAME is the one which has got the smaller arc length. Else (MINLEN is FALSE), NAME is the other one.

NAME is then instantiated with **trim_1**, **trim_2**, **sense_agreement**, **basis_curve.position.location** defined above, with **basis_curve.position.ref_direction**= **R**, and **with basis_curve.position.axis** = **Z** if needed (3D case).

— If the geometric design is not feasible, an error occurs.

— NAME.**master_representation** is implementation dependent.

This created **api_circular_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style**with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_circular_arc** is returned.

If any error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) When there is no solution (taking into account the IN$_i$ parameters), an error occurs.

2) The IN$_i$ parameters are ignored when the corresponding ENTNM$_i$ parameters are not **api_circular_arc**s.

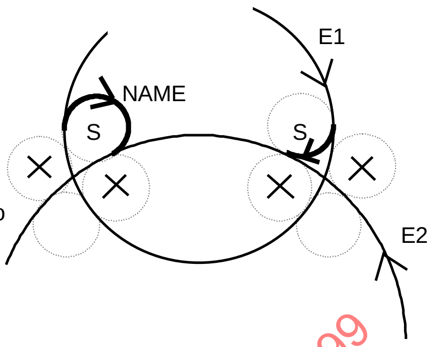
3) The MINLEN parameter is ignored when there is only one **api_circular_arc** computed by the algorithm.

4) The **api_circular_arc** is created only if its curve length from **trim_1** to **trim_2**, consistent with its **sense_agreement**, is not less than EPS. Otherwise, an error occurs.

5) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is greater equal 2), **E1** and **E2** shall be in the same plane.

<u>**EX1**</u> (IN1=TRUE, IN2=FALSE, MIN=TRUE)

There is (at most) 8 possible mathematical circles of a given radius tangent to 2 other ones. Four of them don't fullfill tangency conditions (marked with an "X"), and two of the remaining (here) fullfill the "INi" boolean conditions (marked with an "S"), and define the S set. The MINLEN boolean set to FALSE permits to retain the left **api_circular_arc.** note: in this example, setting both IN1 and IN2 to FALSE would have led to obtain no solution.



<u>**EX2**</u> (IN1 is unused, IN2=FALSE, MIN=TRUE)

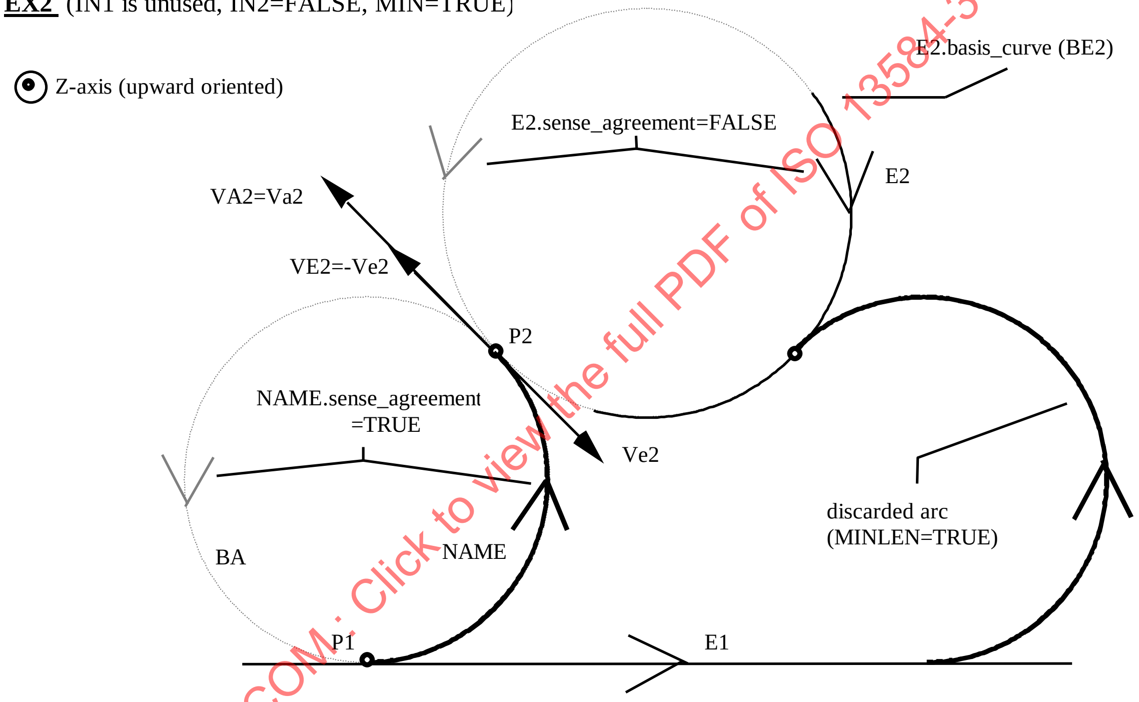


Figure A. 30 — Function: Arc_Rad_2_Ent

<u>Internal Reference:</u>
6.1.9.2, 6.1.12.1, **6.1.12.2**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	101	attempt to create a degenerated entity
106	attempt to create a degenerated axis2_placement during entity creation	110	attempt to create a point outside the parametric range of curves entity
112	attempt to create an arc whose segment length is less than EPS	115	given entities are identical
118	given curves entities are parallel/concentric	119	given entities are not in the same plane
121	radius too big/small	127	geometrical design is not feasible
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.1.2.9 Circular arc defined by three entities

Function name:

Arc_3_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	ENTNM1	N	name of first entity	basic,pnt
in	ENTNM1	N	name of second entity	basic,pnt
in	ENTNM3	N	name of third entity	basic,pnt
in	IN1	E	relative position of NAME to ENTNM1	[TRUE,FALSE]
in	IN2	E	relative position of NAME to ENTNM2	[TRUE,FALSE]
in	IN3	E	relative position of NAME to ENTNM3	[TRUE,FALSE]
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **api_circular_arc**	arc

FORTRAN binding:

```
NAME = ARC_3_ENT ( ENTNM1,ENTNM2,ENTNM3,IN1,IN2,IN3,KFIX )
```

Effects:

In the following, we will consider **E1**, **E2** and **E3** as synonym for ENTNM1, ENTNM2 and ENTNM3.

This function creates an **api_circular_arc** defined by three constraints with three entities **E1**, **E2**, and **E3**. These entities are either a **cartesian_point**, an **api_line**, or an **api_circular_arc**. When one of these entities is a **cartesian_point**, the related constraint is the association of this point to the NAME **api_circular_arc**; else the entity is an **api_line** or an **api_circular_arc**, and the related constraint is the tangency of NAME with the **basis_curve** (respectively **line** and **circle**) of this entity, with the so-called "pulleys dependency" (same orientation of the two **trimmed_curve**s at tangential points). The three Boolean parameters IN$_i$ are meant to solve cases of ambiguity, by specifying the relative position of the required solution, relatively (inside or outside) to each possibly **api_circular_arc** parameter.

If **E1** is a **cartesian_point**, then let **p1** be the duplication of that point. Else, let **p1** be the tangential point between NAME and the **basis_curve** of **E1**.

If **E2** is a **cartesian_point**, then let **p2** be the duplication of that point. Else, let **p2** be the tangential point between NAME and the **basis_curve** of **E2**.

If **E3** is a **cartesian_point**, then let **p3** be the duplication of that point. Else, let **p3** be the tangential point between NAME and the **basis_curve** of **E3**.

p1, p2, p3 are computed by the interface and are assigned a **null_style**.

The NAME returned **api_circular_arc** shall be such that:

— for each i in [1..3], if E_i is an **api_circular_arc**, then if IN_i =TRUE, then NAME is inside E_i; else (IN_i=FALSE) NAME is outside $\mathbf{E_i}$. If $\mathbf{E_i}$ is not an **api_circular_arc**, then IN_i is not used for the determination of NAME.

— **p1** is the starting trimming point of NAME (i.e NAME.**trim_1**[1]), **p3** is its end trimming point (i.e. NAME.**trim_2**[1]), and **p2** belongs to the parameter range defined by these two trimming points **p1**, **p3**. The distance from **p2** to **p1** or to **p3** can be less than ZERO_VALUE, that means that the two points are identical.

— NAME's **position** (**axis_2_placement**) is instantiated with **location** its computed centre, and :

- — if the current open view is defined as a 2D view, then NAME.**basis_curve.position.ref_direction** is the x direction of the OVC Reference System.

- — else (3D case), if one of the three parameter entities is an **api_circular_arc**, then the directions of NAME.**basis_curve.position** are copied from the **axis_2_placement** of the first **api_circular_arc** of the list (**E1**, **E2**, **E3**) in this order.

- — else (3D case, and no **api_circular_arc**s), if **E1**, **E2** and **E3** are all **cartesian_point**s, then

  NAME**.basis_curve.position.ref_direction**= **E1E2** (i.e the **direction** with origin **E1** and extremity **E2**) and NAME**.basis_curve.position.axis**= **E1E2** ´ **E1E3** (where ´ is the cross product). The 3 points shall not be the same, nor be on the same line.

- — else (3D case, no **api_circular_arc**s and at least one **api_line**), let **L** be the first **api_line** of (**E1**,**E2**,**E3**) in this order, **F1** and **F2** the two remaining entities (in the same order). If **F1** and **F2** are both **cartesian_point**s, then let **P** be the first of them that does not belong to the **basis_curve** of **L**. Else (**F1** or **F2** is an **api_line**), let **G** be the first **api_line** of (**F1**,**F2**) in this order, and let **P** be its origin if it does not belong to the **basis_curve** of **L**, its extremity otherwise. Let **O** be the origin point of **L**.

  Then NAME**.basis_curve.position.ref_direction**= **L.dir.orientation** and NAME**.basis_curve.position.axis** = **L.dir.orientation** ´ **OP** .

— Its **sense_agreement** is such that:

- — let BN be the basis_curve of NAME.
- — let $\mathbf{BE_i}$ be the **basis_curve** of $\mathbf{E_i}$, if $\mathbf{E_i}$ is not a **cartesian_point**.
- — let $\mathbf{P_i}$ be the tangential point between **BN** and $\mathbf{BE_i}$.
- — let $\mathbf{Vn_i}$ be the vector tangent to **BN** at $\mathbf{P_i}$.
- — let $\mathbf{VN_i}$ be $\mathbf{Vn_i}$ if NAME.**sense_agreement** is true, $\mathbf{-Vn_i}$ otherwise.
- — let $\mathbf{VE_i}$ be the vector tangent to BE_i at P_i.
- — let Ve_i be VE_i if E_i.**sense_agreement** is true, $\mathbf{-VE_i}$ otherwise.

Then for each i such that $\mathbf{E_i}$ is not a **cartesian_point**, $\mathbf{Vn_i}$ and $\mathbf{Ve_i}$ have the same direction.

If the three entities are all **cartesian_point**, NAME.**sense_agreement** equals TRUE.

— If the geometric design is not feasible, an error occurs.

— the value of NAME's radius shall be in range [EPS,MAX].

— NAME.**master_representation** is implementation dependent.

This created **api_circular_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style**with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_circular_arc** is returned.

If any error occurs, no entity is created and the entity name is returned as zero.

Notes:

1) When there is no solution (taking into account the IN_i parameters), an error occurs.

2) The IN_i parameters are ignored when the corresponding $ENTNM_i$ parameters are not **api_circular_arc**s.

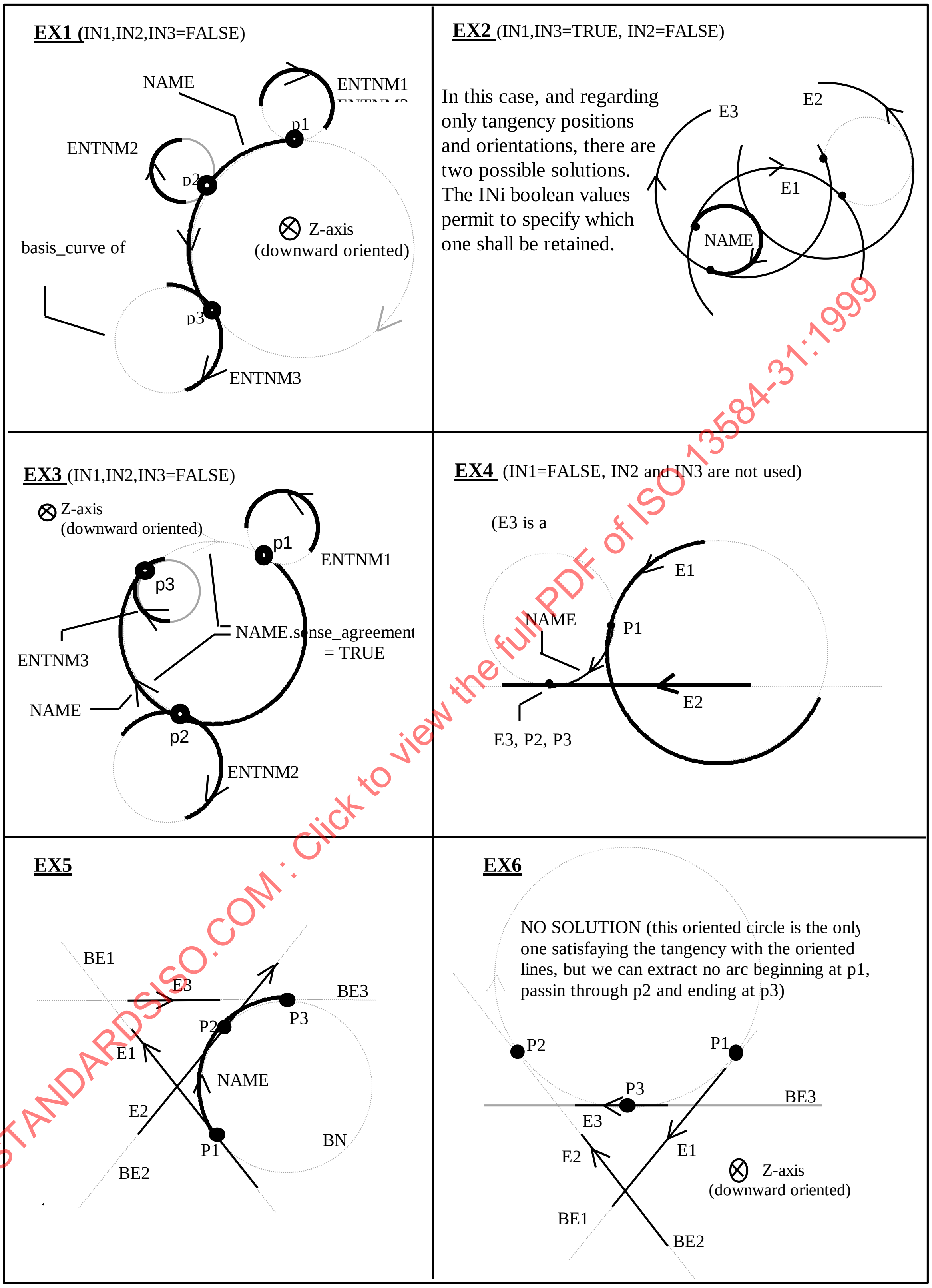
3) The **api_circular_arc** is created only if its curve length from **trim_1** to **trim_2**, consistent with its **sense_agreement**, is not less than EPS. Otherwise, an error occurs.

4) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is greater equal 2), **E1**, **E2** and **E3** shall be in the same plane.

Internal Reference:
6.1.9.2, 6.1.12.1, **6.1.12.2**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
101	attempt to create a degenerated entity	106	attempt to create a degenerated axis2_placement during entity creation
110	attempt to create a point outside the parametric range of curves entity	112	attempt to create an arc whose segment length is less than EPS
115	given entities are identical	118	given curves entities are parallel/concentric
119	given entities are not in the same plane	127	geometrical design is not feasible
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range



In this case, and regarding only tangency positions and orientations, there are two possible solutions. The INi boolean values permit to specify which one shall be retained.

NO SOLUTION (this oriented circle is the only one satisfaying the tangency with the oriented lines, but we can extract no arc beginning at p1, passin through p2 and ending at p3)

Figure A. 31 — Function: Arc_3_Ent

A.5.3.2 Conic arc curve entities

A.5.3.2.1 Ellipse and elliptical arc (api_elliptical_arc)

Ellipse by two diameters and placement	Ellipse_Diameter_A2p
Elliptical arc generation	Elc_Gen

A.5.3.2.1.1 Ellipse by two diameters and placement

Function name:

Ellipse_2_Diameter_A2p

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	SEMI1	D	Semi_axis_1 length of **api_elliptical_arc**	(EPS ≤ SEMI1 ≤ MAX)
in	SEMI2	D	Semi_axis_2 length of **api_elliptical_arc**	(EPS ≤ SEMI2 ≤ MAX)
in	A2PNAM	N	Name of **axis2_placement**	a2p
in	SENSE	E	Sense agreement flag	[TRUE,FALSE]
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **api_elliptical_arc**	elc

FORTRAN binding:

```
NAME = ELLIPSE_2_DIAMETER_A2P ( SEMI1,SEMI2,A2PNAM,SENSE,KFIX )
```

Effects:

Creates a whole ellipse as an **api_elliptical_arc** entity given by the lengths of the semi-major diameter (SEMI1) and the semi-minor diameter (SEMI2), an **axis2_placement** (A2PNAM), and a sense agreement flag (SENSE) that defines the direction of the newly created **api_circular_arc** in conjunction with an **ellipse** entity as **basis_curve**. The direction of the (Ox) axis of the given position A2PNAM defines the direction of the semi-major axis.

The **axis2_placement** (A2PNAM) is duplicated as **a2p1** and it has a **null_style** assigned to it. Then:

— an **ellipse e** is instantiated with **position a2p1**, **semi_axis_1** equals SEMI1 and **semi_axis_2** equals SEMI2. This **ellipse** has a **null_style** assigned to it.

— an **api_elliptical_arc** is instantiated with **e** as **basis_curve**, a parameter value equals 0 degree as **trim_1** and a parameter value equals 360 degrees as **trim_2**, **sense_agreement** equals SENSE and **master_representation** shall be implementation dependent. This **api_elliptical_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_elliptical_arc** is returned.

The values of MAJA and MINA shall not be in range [EPS,MAX] and are measured in *OVC_length_units* . If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) The interface ensure the closure of the created **api_elliptical_arc** entity (cartesian coordinates of **trim_1** equal to cartesian coordinates of **trim_2**).

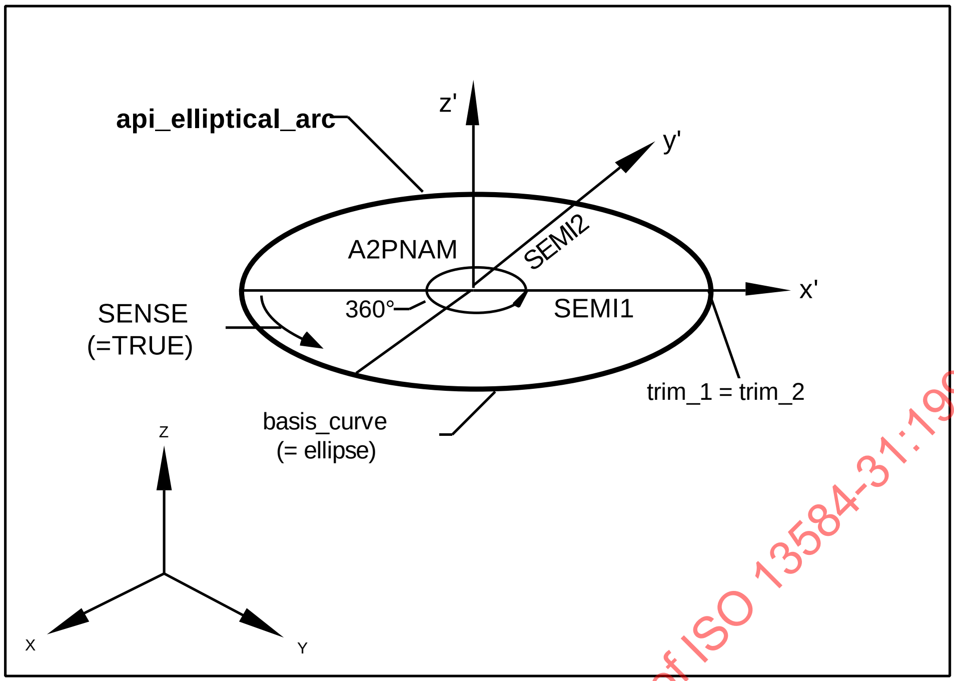


Figure A. 32 — Function: Ellipse_2_Diameter_A2p

Internal Reference:

6.1.9, **6.1.13.1**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	101	attempt to create a degenerated entity
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.2.1.2 Elliptical arc generation

Function name:

Elc_Gen

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	SEMI1	D	semi_axis_1 length of **api_elliptical_arc**	(EPS ≤ SEMI1 ≤ MAX)
in	SEMI2	D	semi_axis_2 length of **api_elliptical_arc**	(EPS ≤ SEMI2 ≤ MAX)
in	STAANG	D	start angle in (Oxy) plane relative to (Ox) axis of the given a2p	(0° ≤ STAANG ≤ 360°)
in	ENDANG	D	end angle in (Oxy) plane relative to (Ox) axis of the given a2p	(0° ≤ STAANG ≤ 360°)
in	A2PNAM	N	name of **axis2_placement**	a2p
in	SENSE	E	sense agreement flag	[TRUE,FALSE]
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **api_elliptical_arc**	elc

FORTRAN binding:

```
NAME = ELC_GEN ( SEMI1,SEMI2,STAANG,ENDANG,A2PNAM,SENSE,KFIX )
```

Effects:

Creates an **api_elliptical_arc** entity given by the lengths of the semi-major diameter (SEMI1) and the semi-minor diameter (SEMI2), an a**xis2_placement** (A2PNAM), and a sense agreement flag (SENSE) that defines the direction of the newly created **api_circular_arc** in conjunction with an **ellipse** entity as **basis_curve**, together with a start and an end point that are implicitly defined by the two angles STAANG and ENDANG, respectively. The direction of the (Ox) axis of the given position A2PNAM defines the direction of the semi-major axis.

The **axis2_placement** (A2PNAM) is duplicated as **a2p1** and it has a **null_style** assigned to it. Then:

— an **ellipse e** is instantiated with **position a2p1**, **semi_axis_1** equals SEMI1 and **semi_axis_2** equals SEMI2. This **ellipse** has a **null_style** assigned to it.

— an **api_elliptical_arc** is instantiated with **e** as **basis_curve**, the parameter value STAANG as **trim_1** and the parameter value ENDANG as **trim_2**, **sense_agreement** equals SENSE and **master_representation** shall be implementation dependent. This **api_elliptical_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style**with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_elliptical_arc** is returned.

The values of SEMI1 and SEMI2 shall be in range [EPS,MAX] and are measured in *OVC_length_units* ; the given angles are measured in *OVC_angle_unit*s, counted in (Oxy) plane of given A2PNAM. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) If the two parameter values, STAANG and ENDANG, defines the same point in a range of [ZERO_VALUE,EPS], that means that both trimming points **trim_1** and **trim_2** are identical, the created circular arc is exact a whole ellipse. Then, the interface ensure the closure of the created **api_elliptical_arc** entity.

2) The **api_elliptical_arc** is created, only if its segment length from **trim_1** to **trim_2**, consistent with its **sense_agreement**, is grater then EPS. Otherwise an error occurs.

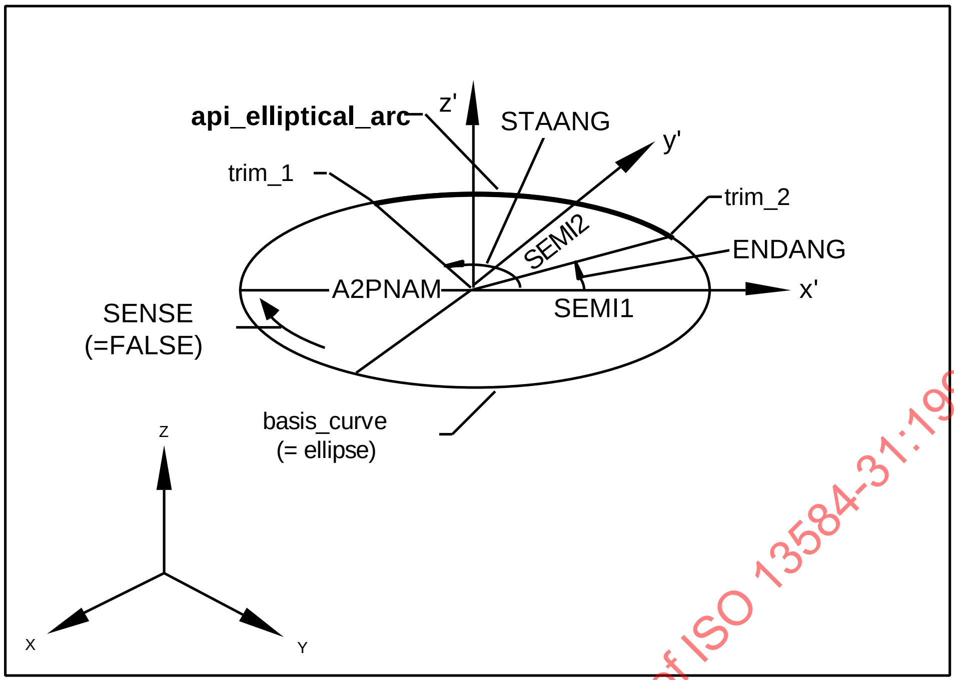


Figure A. 33 — Function: Elc_Gen

Internal Reference:
6.1.9, **6.1.13.1**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	4	value for plane angle measure out of permitted range
101	attempt to create a degenerated entity	112	attempt to create an arc whose segment length is less than EPS
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	1001	enumerated value out of range

A.5.3.2.2 Hyperbolical arc (api_hyperbolic_arc)

Hyperbolical arc generation Hyp_Gen

A.5.3.2.2.1 Hyperbolical arc generation

Function name:
Hyp_Gen

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	SEMAXI	D	Semi axis length of **hyperbola**	(EPS ≤ SEMAXI ≤ MAX)
in	SEMIMG	D	Semi imag axis length of **hyperbola**	(EPS ≤ SEMIMG ≤ MAX)
in	STAANG	D	end angle in (Oxy) plane of the given a2p, around the focal point, starting at (Ox) axis	(0° ≤ STAANG ≤ 360°)
in	ENDANG	D	end angle in (Oxy) plane of the given a2p, around the focal point, starting at (Ox) axis	(0° ≤ STAANG ≤ 360°)
in	A2PNAM	N	Name of **axis2_placement**	a2p
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of created **api_hyperbolic_arc**	hyp

FORTRAN binding:

```
NAME = HYP_GEN ( SEMAXI,SEMIMG,STAANG,ENDANG,A2PNAM,KFIX )
```

Effects:

Creates an **api_hyperbolic_arc** entity that is a segment of one **hyperbola** curve, defined by the lengths of the semi axis (SEMAXI) and the semi-imag axis (SEMIMG), and an **axis2_placement** (A2PNAM) that defines the location and orientation in the centre of the **hyperbola**. The branch of the **hyperbola** represented is that pointed to by the x-direction of the given A2PNAM. The direction of the newly created **api_hyperbolic_arc** is implicitly defined by a start point and an end point given through the two angles STAANG and ENDANG, respectively. The direction of the (Ox) axis of the given position A2PNAM defines the direction of the semi axis of the **hyperbola**.

The **axis2_placement** (A2PNAM) is duplicated as **a2p1** and it has a **null_style** assigned to it. Then:

— a **hyperbola h** is instantiated with **position a2p1**, **semi_axis_1** equals SEMAXI and **semi_imag_axis_2** equals SEMIMG. This **hyperbola** has a **null_style** assigned to it.

— a **cartesian_point p1** is instantiated, with **coordinates** derived from an intersection of a line, starting at the focal point of the **hyperbola h**, in direction STAANG, and the **hyperbola h**. This **cartesian_point** has a **null_style** assigned to it.

— a **cartesian_point p2** is instantiated, with **coordinates** derived from an intersection of a line, starting at the focal point of the **hyperbola h**, in direction ENDANG, and the **hyperbola h**. This **cartesian_point** has a **null_style** assigned to it.

— an **api_hyperbolic_arc** is instantiated with **h** as **basis_curve**, **p1** as **trim_1** and **p2** as **trim_2**, **sense_agreement** equals true, if STAANG < ENDANG otherwise **sense_agreement** equals false, and **master_representation** shall be implementation dependent. This **api_hyperbolic_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style**with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_hyperbolic_arc** is returned.

The values of SEMAXI and SEMIMG shall be in range [EPS,MAX] and are measured in *OVC_length_units*. The given angles are measured in *OVC_angle_unit*s, counted in (Oxy) plane of given A2PNAM around the focal point, starting at the (Ox) axis. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) If the two given values, STAANG and ENDANG, define the same point in the range of [ZERO_VALUE,EPS], then the segment length of the created **api_hyperbolic_arc** is less than

EPS, or if the calculation process for the two **cartesian_points p1** and **p2** delivers no real solution, no **api_hyperbolic_arc** is created and an error occurs.

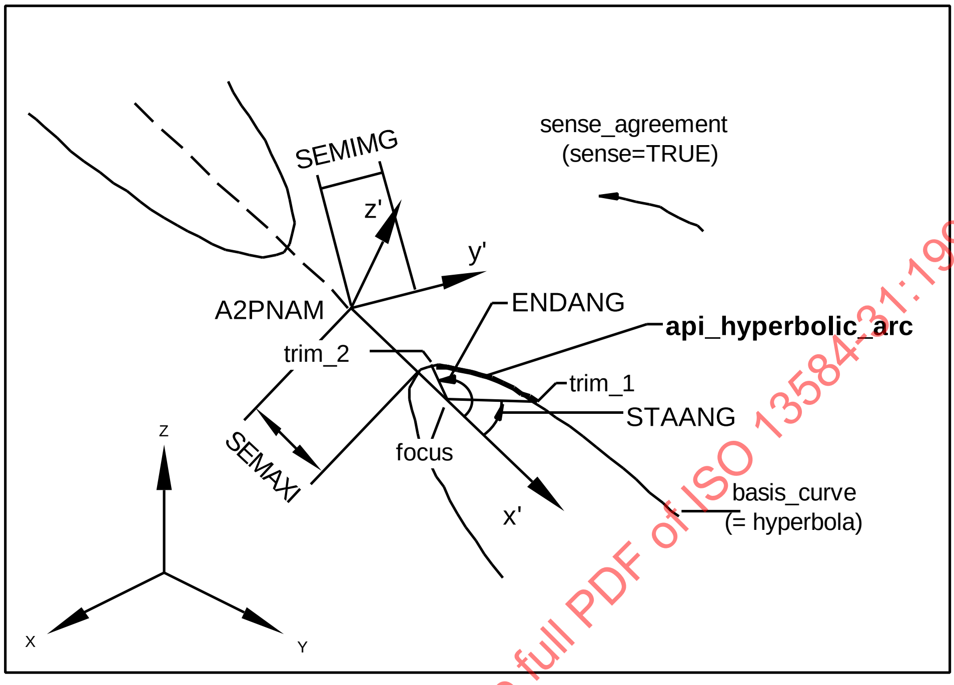


Figure A. 34 — Function: Hyp_Gen

Internal Reference:
6.1.9, **6.1.13.2**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	4	value for plane angle measure out of permitted range
101	attempt to create a degenerated entity	112	attempt to create an arc whose segment length is less than EPS
128	calculation process for creating a conical arc numerical not stable	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.3.2.3 Parabolical arc (api_parabolic_arc)

Parabolical arc generation Par_Gen

A.5.3.2.3.1 Parabolical arc generation

Function name:

Par_Gen

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	FOCLEN	D	distance of the focal point of **parabola**	(EPS ≤ FOCLEN ≤ MAX)
in	STAANG	D	end angle in (Oxy) plane of the given a2p, around the focal point, starting at (Ox) axis	(0° ≤ STAANG ≤ 360°)
in	ENDANG	D	end angle in (Oxy) plane of the given a2p, around the focal point, starting at (Ox) axis	(0° ≤ STAANG ≤ 360°)
in	A2PNAM	N	name of **axis2_placement**	a2p
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **api_hyperbolic_arc**	par

FORTRAN binding:

```
NAME = PAR_GEN ( FOCLEN,STAANG,ENDANG,A2PNAM,KFIX )
```

Effects:

Creates an **api_parabolic_arc** entity that is a segment of a **parabola** curve, defined by its focal length (FOCLEN) and an **axis2_placement** (A2PNAM) that defines the location and orientation in the apex of the **parabola**. The value of FOCLEN defines the distance of the focal point from the apex point in direction of the (Ox) axis of A2PNAM. The direction of the newly created **api_parabolic_arc** is implicitly defined by a start point and an end point given through the two angles STAANG and ENDANG, respectively. The direction of the (Ox) axis of the given position A2PNAM defines the axis of symmetry of the **parabola**.

The **axis2_placement** (A2PNAM) is duplicated as **a2p1** and it has a **null_style** assigned to it. Then:

— a **parabola p** is instantiated with **position a2p1**, **focal_dist** equals FOCLEN. This **parabola** has a **null_style** assigned to it.

— a **cartesian_point p1** is instantiated, with **coordinates** derived from an intersection of a line, starting at the focal point of the **parabola p**, in direction STAANG, and the **parabola p**. This **cartesian_point** has a **null_style** assigned to it.

— a **cartesian_point p2** is instantiated, with **coordinates** derived from an intersection of a line, starting at the focal point of the **parabola p**, in direction ENDANG, and the **parabola p**. This **cartesian_point** has a **null_style** assigned to it.

— an **api_parabolic_arc** is instantiated with **p** as **basis_curve**, **p1** as **trim_1** and **p2** as **trim_2**, **sense_agreement** equals true, if STAANG < ENDANG otherwise **sense_agreement** equals false, and **master_representation** shall be implementation dependent. This **api_parabolic_arc** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_parabolic_arc** is returned.

The value of FOCLEN shall be in the range [EPS,MAX] and is measured in *OVC_length_units* . The given angles are measured in *OVC_angle_units*, counted in (Oxy) plane of given A2PNAM around the focal point, starting at the (Ox) axis. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) If the two given values, STAANG and ENDANG, define the same point in a range of [ZERO_VALUE,EPS], then the segment length of the created **api_parabolic_arc** is less than

EPS, or if the calculation process for the two **cartesian_points p1** and **p2** delivers no real solution, no **api_parabolic_arc** is created and an error occurs.

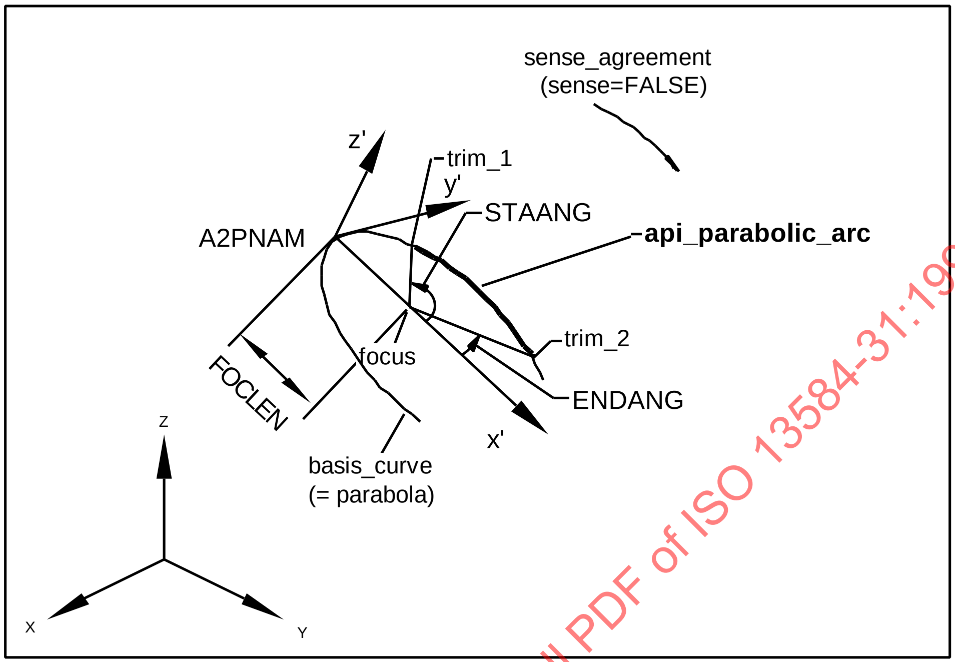


Figure A. 35 — Function: Par_Gen

Internal Reference:
6.1.9, **6.1.13.3**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	4	value for plane angle measure out of permitted range
101	attempt to create a degenerated entity	112	attempt to create an arc whose segment length is less than EPS
128	calculation process for creating a conical arc numerical not stable	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.3.3 General curve entities

A.5.3.3.1 Polyline

Polyline by cartesian coordinates	Pln_Cartesian_Coordinate
Polyline by list of points	Pln_Pnt_List

A.5.3.3.1.1Polyline by cartesian coordinates

Function name:

Pln_Cartesian_Coordinate

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	N	I	length of each coordinate lists XLST,YLST and ZLST, that defines the number (**n**) of **cartesian_point** in the LIST OF point for **polyline**	≥ 2
in	XLST	nxD	list of x- coordinates of **cartesian_point** $\mathbf{p_i}$ (see Note (1))	(0.0 OR (EPS≤|XLST(i)|≤MAX))
in	YLST	nxD	list of y- coordinates of **cartesian_point** $\mathbf{p_i}$ (see Note (1))	(0.0 OR (EPS≤|YLST(i)|≤MAX))
in	ZLST	nxD	list of z- coordinates of **cartesian_point** $\mathbf{p_i}$ (see Note (1) and (2))	(0.0 OR (EPS≤|ZLST(i)|≤MAX))
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	name of created **polyline**	pln

FORTRAN binding:

```
NAME = PLN_CARTESIAN_COORDINATE ( N,XLST,YLST,ZLST,KFIX )
```

Effects:

Creates a **polyline** entity defined by a three lists of cartesian coordinates (XLST,YLST,ZLST). Each given coordinate triple x,y and z (XLST(i),YLST(i),ZLST(i), for 1 ≤ i ≤ **n**) of these lists, defines a **cartesian_point**, that represented a point in the LIST OF **cartesian_point** of the points attribute of **polyline**.

— **n** numbers of **cartesian_point**s $\mathbf{p_i}$ are instantiated, with **coordinates x**,**y** and **z** derived from $XLST_i$, $YLST_i$ and $ZLST_i$ with i=1,...,n. All these **cartesian_point**s have a **null_style** assigned to them.

— a **polyline** is instantiated with **points** equal to a list of **cartesian_point**s, derived from **n** **cartesian_point**s $\mathbf{p_i}$, i=1,...,n. This **polyline** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **polyline** is returned.

The length of each linear segment, built up by two **cartesian_point**s $\mathbf{p_i}$ and $\mathbf{p_{i+1}}$, i=1,...,n-1 shall be in range [EPS,MAX]. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) The length (**n**) of each given list XLST,YLST and ZLST, that defines the number of points $\mathbf{p_i}$ building the linear segments of the polyline, shall be the same for each list.

2) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the list of z-coordinates ZLST will subsequently be ignored by the interface.

Internal Reference:

6.1.9, **6.1.14.1**, 6.2.4, 8.2

Errors:

3	value for length measure out of permitted range	5	integer value out of permitted range
101	attempt to create a degenerated entity	103	distance between two points out of range [EPS,MAX]
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	205	maximal number of points per polyline exceeded
1001	enumerated value out of range		

A.5.3.3.1.2 Polyline by list of points

Function name:

Pln_Pnt_List

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	N	I	length of PNTLST, that defines the number of points	≥ 2
in	PNTLST	nxN	list of **cartesian_point** names (see below)	pnt
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	name of created **polyline**	pln

FORTRAN binding:

```
NAME = PLN_PNT_LIST ( N,PNTLST,KFIX )
```

Effects:

Creates a **polyline** entity defined by a list of existing **cartesian_point**s (PNTLST). The given length of PNTLST, N (**n**), defines the number of points building the linear segments of the **polyline**.

— a list **list1** of **cartesian_points** $\mathbf{p_i}$, i=1,...,n are created by duplicating the **n cartesian_point**s that contains the PNTLST. All these **cartesian_point**s have a **null_style** assigned to them.

— a **polyline** is instantiated with **points** equals **list1**. This **polyline** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **polyline** is returned.

The length of each linear segment, build up by two **cartesian_point**s $\mathbf{p_i}$ and $\mathbf{p_{i+1}}$, i=1,...,n-1 shall be in range [EPS,MAX]. If an error occurs, no entity is created and the element name is returned as zero.

Internal Reference:

6.1.9, **6.1.14.1**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
5	integer value out of permitted range	101	attempt to create a degenerated entity
103	distance between two points out of range [EPS,MAX]	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
205	maximal number of points per polyline exceeded	1001	enumerated value out of range

A.5.3.3.2 Planar contour (api_contour)

Generation of a contour Ctr_Gen

A.5.3.3.2.1 Generation of a contour

Function name:

Ctr_Gen

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	N	I	length of ENTLST, that defines the number of elements	≥ 1
in	ENTLST	nxN	list of entity names, that defines the **api_contour**	basic,conic_arc,pln,grp
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	name of created **api_contour**	ctr

FORTRAN binding:

```
NAME = CTR_GEN ( N,ENTLST,KFIX )
```

Effects:

Creates an **api_contour** entity that is a non self-intersecting oriented planar closed **composite_curve** built up by a list of existing entities (ENTLST). This given list of curve entities may be ordered or not. All instances of given entities, defined in ENTLST, are duplicated and they have a **null_style** assigned to them. Then:

— a list **list1** of **composite_curve_segments** are computed by the interface using the replicas of the given entities. The process explained in **6.1.14.2** ensures the closeness of the contour and delivers also a number **n** of **composite_curve_segments** using for **n_segment** attribute.

— an **api_contour** is instantiated with **segments** equals **list1**, **self_intersect** equals false, **n_segments** equals **n** and **closed_curve** equals true. This **api_contour** has the current *curve_style* entry from the interface status table assigned to it, and in case of an open 2D view with both entries for *hidden_line* equals ON and for *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **api_contour** is returned.

If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) The interior of the **api_contour** shall be capable of containing a circle of diameter EPS.

2) If the instance of an entity, given by ENTLST, is of type **api_group.** Then all instances of entities that are contained within this group, and that are valid for the parametric range of this function (basic, conic_arc and pln) are duplicated for the creation of the list of **composite_curve_segments list1**.

3) The entities that are permitted for contour representation are defined in the *contour_entities* entry of the interface description table. If some other curve entities used in this function that are also permitted, but not in the range of *contour_entities*, this entities shall be simulated by the simulation process defined for this entities through the interface.

4) When the current open view is defined as a 3D view (it means that the *geometrical_power_level* entry of the interface status table is greater equal 2), all given elements defined in ENTLST shall be in the same plane.

Internal Reference:

6.1.12, 6.1.13, 6.1.14, **6.1.14.2**, 6.1.19, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
5	integer value out of permitted range	101	attempt to create a degenerated entity
113	attempt to create a self-intersected contour entity	115	given entities are identical
119	given entities are not in the same plane	129	approximation process to ensure contour closure failed
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	206	maximal number of entities per contour exceeded
1001	enumerated value out of range		

A.5.4 Fill area entities

Generation of an annotation_fill_area	Afa_Gen
Generation of a fill_area_style_hatching	Fsh_Gen
Hatching of an annotation_fill_area	Hatch_Afa

A.5.4.1 Generation of an annotation_fill_area

Function name:

Afa_Gen

interface level:	**1**
geometrical power level:	**1**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	CTRNAM	N	name of the **api_contour** of the outer boundary	ctr
in	N	I	length of CTRLST, that defines the number of **api_contour**s for the inner boundaries	≥ 0
in	CTRLST	nxN	list of **api_contour** names, defining the inner contours	ctr
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	name of created **annotation_fill_area**	afa

FORTRAN binding:

```
NAME = AFA_GEN ( CTRNAM,N,CTRLST,KFIX )
```

Effects:

Creates an **annotation_fill_area** entity that is a set of **curve**s that may be filled with hatching, shading, colour or tiling. The **annotation_fill_area** is described by boundaries, that consist of non-intersecting and non-self-intersecting closed **curve**s. These **curve**s form the boundary of the planar areas to be filled according to the style for the **annotation_fill_area**. An **annotation_fill_area** is planar connective 2-manifold whose external boundary is an **api_contour,** given by CTRNAM and that may have internal boundaries defined by a list of **api_contour**s CTRLST.

All the contours are in the same plane and shall not intersect each other. All the (possible) contours that define the internal boundaries of the fill area shall belong to the interior of the contour that defines its external boundary and have non intersecting interiors.

The given **api_contours** are duplicated as $\mathbf{a_1,...,a_n}$ and they have a **null_style** assigned to them. Then:

— an **annotation_fill_area** is instantiated with **boundaries** as a set of the **api_contours** $\mathbf{a_1,...,a_n}$ and has the current *fill_area_style* entry from the interface status table assigned to it, and if both entries for *hidden_line* equals ON and *hidden_line_involved* equals TRUE, it is attached to an **api_pre_defined_occlusion_style** with the current values of *view_level* and *hidden_line_aspect* entries from the interface status table. The name of this **annotation_fill_area** is returned. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) In the context of the **api_abstract_schema**, an **annotation_fill_area** is allowed for 2D views only, i.e., when the interface is initialised with *geometrical_power_level* equals 1.

2) In case of no inner boundaries (length of list CTRLST equal to zero), the parameter CTRLST shall be ignored.

3) If there is two or more **api_contours** in the specification of the **annotation_fill_area**, the distance between two **api_contours** shall not be in range [ZERO_VALUE,EPS].

Internal Reference:

6.1.14, **6.1.15.1**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
5	integer value out of permitted range	101	attempt to create a degenerated entity
104	distance between two contours less than EPS	119	given entities are not in the same plane
123	an intersection of given contours detected	125	an overlapping of given contours detected
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	207	maximal number of inner boundaries exceeded
1001	enumerated value out of range		

A.5.4.2 Generation of a fill_area_style_hatching

Function name:

Fsh_Gen

interface level:	**1**
geometrical power level:	**1**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	REFPNT	N	name of **cartesian_point** of the origin for mapping	pnt
in	DIST1	D	distance of parallel hatching lines	(EPS ≤ DIST1 ≤ MAX)
in	DIST2	D	distance from REFPNT, defining the pattern start point	(0.0 OR (EPS ≤ DIST2 ≤ MAX))
in	ANGLE	D	angle of the parallel hatching lines relative to (Ox) axis of the current OVC	(0° ≤ ANGLE ≤ 180°)
out	NAME	N	name of created **fill_area_style_hatching**	fsh

FORTRAN binding:

```
NAME FSH_GEN ( REFPNT,DIST1,DIST2,ANGLE )
```

Effects:

Creates a **fill_area_style_hatching** entity that defines a styled pattern of curves for hatching visible annotation fill areas

The given **cartesian_point** REFPNT, that defines the origin point for further mapping of this **fill_area_style_hatching** entity NAME is duplicated as **p1** and it has a **null_style** assigned to it. Let **l** be a virtual reference hatch line, going through point **p1** with direction derived from given angle ANGLE. Then:

— a **vector v** is instantiated with **orientation** perpendicular to the reference hatching line **l**, and **magnitude** DIST1. This **vector** has a **null_style** assigned to it.

— a **one_direction_repeat_factor o** is instantiated with **repeat_factor v** .

— a **cartesian_point p2** is instantiated, either by a duplication of REFPNT if the given value of DIST2 is equal to zero, or by a calculation of point on reference hatching line **l** with distance DIST2 from the given reference point REFPNT. This **cartesian_point** has a **null_style** assigned to it.

— a ***curve_style*** **c** is defined by the interface status table entries: *hatch_width*, *hatch_curve_font* and *hatch_colour*.

— a **fill_area_style_hatching** is instantiated with **hatch_line_appearance** equals **c**, **start_of_next_hatch_line** equals **o**, **point_of_reference_hatch_line** equals **p1**, **pattern_start p2**, **hatch_line_angle** equals ANGLE.

Both given distances DIST1 and DIST2 are measured in *OVC_length_units* and the value for DIST1 shall be in range [EPS,MAX]. The value for DIST2 shall be either equal to zero or also in range [EPS,MAX]. The value for ANGLE is counted in *OVC_angle_units*. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) In the context of the **api_abstract_schema fill_area_style_hatching** is only used for styling an **annotation_fill_area**. Therefore a **fill_area_style_hatching** is stored in the TDB only.

2) In the context of the **api_abstract_schema hatch_line_appearance** shall be defined through **pre_defined_items.**

3) The creation of a **fill_area_style_hatching** entity is allowed for 2D views only, i.e., when the interface is initialised with *geometrical_power_level* equals 1.

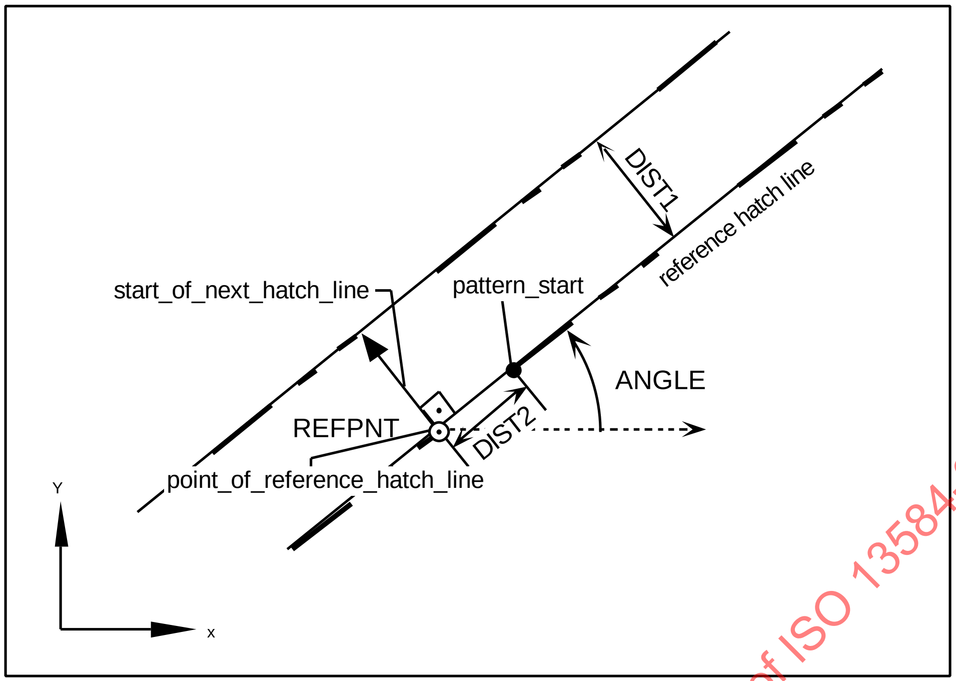


Figure A. 36 — Function: Fsh_Gen

Internal Reference:
6.1.9, 6.1.15, **6.2.3.6**, 6.2.5, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	4	value for plane angle measure out of permitted range
101	attempt to create a degenerated entity	201	temporary database overflow
204	function not compatible with current power level		

A.5.4.3 Hatching of an annotation_fill_area

Function name:

Hatch_Afa

interface level:	**1**
geometrical power level:	**1**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	HATCH	N	name of **fill_area_style_hatching**, to be mapped	fsh
in	AFA	N	name of **annotation_fill_area**	afa
in	TARPNT	N	name of **cartesian_point**, used as starting location for hatching	pnt

FORTRAN binding:

```
CALL HATCH_AFA ( HATCH,AFA,REFPNT )
```

Effects:

Invokes the assignment of a **fill_area_style_hatching** (HATCH) to an **annotation_fill_area** (AFA) with the target point given through the **cartesian_point**, TARPNT. The target point specifies the starting point for the fill style. This given **cartesian_point** TARPNT is duplicated as **p1** and has a **null_style** assigned to it. Then:

— an **annotation_fill_area_occurrence** entity is instantiated, with **fill_style_target** equals **p1**, fill style **SELF.styles** equals HATCH and representation item **SELF.item** equals AFA.

The assignment shall be performed by transformation created from the start point of the reference hatch line of HATCH as origin and the point **fill_style_target** (**p1**) as target, using an implicit mapping of the x axis of the OVC reference system to that the HATCH entity belongs on the x axis of the OVC reference system to that the target point REFPNT belongs. If an error occurs, no assignment will be done.

Notes:

1) In the context of the **api_abstract_schema**, this style assignment is allowed for 2D views only , i.e., when the interface is initialised with *geometrical_power_level* equals 1.

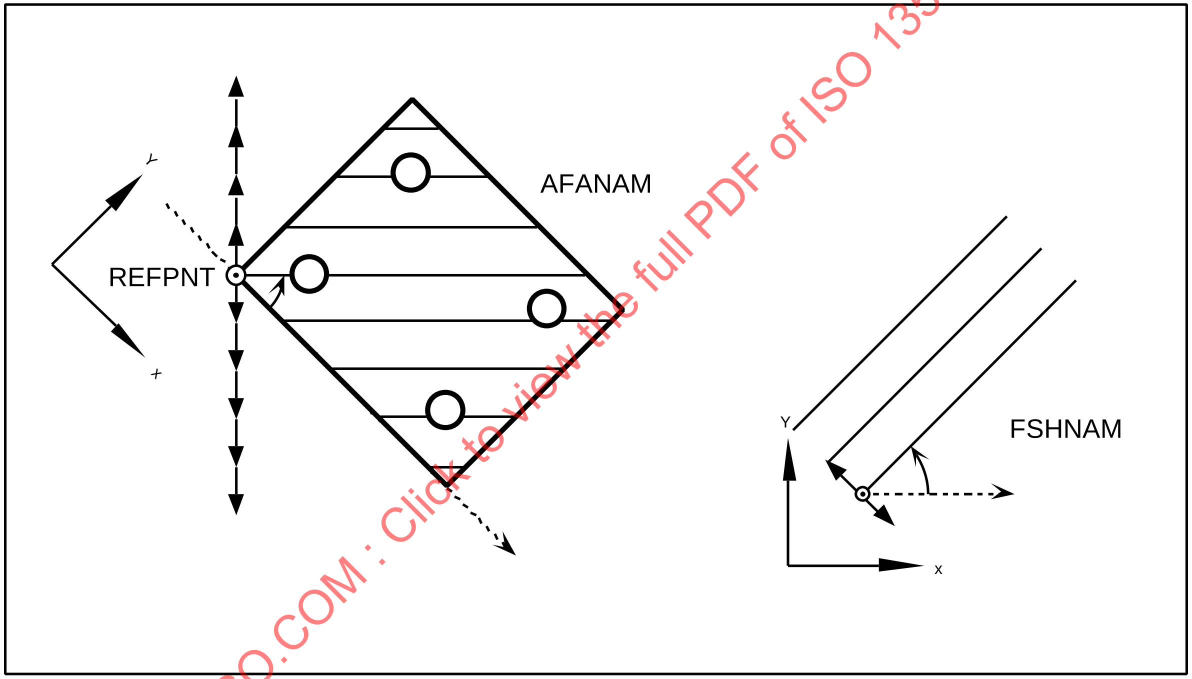


Figure A. 37 — Function: Hatch_Afa

Internal Reference:

6.1.9, 6.1.15, **6.2.3.13**, 6.2.3.6, 6.2.5, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
201	temporary database overflow	204	function not compatible with current power level
403	assignment of hatch style failed		

A.5.5 Surface entities

Generation of an api_planar_surface Aps_Gen

A.5.5.1 Generation of an api_planar_surface

Function name:

Aps_Gen

interface level:	**2**
geometrical power level:	**2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	CTRNAM	N	name of the **api_contour** of the outer boundary	ctr
in	N	I	length of CTRLST, that defines the number of **api_contour**s for the inner boundaries	≥ 0
in	CTRLST	nxN	list of **api_contour** names, defining the inner contours	ctr
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **api_planar_surface**	aps

FORTRAN binding:

```
NAME = APS_GEN ( CTRNAM,N,CTRLST,KFIX )
```

Effects:

An **api_planar_surface** is specified through an **api_contour** CTRNAM that corresponds to the external boundary of the surface, and a list CTRLST of **api_contour**s that correspond to the (possible) inner boundaries of the surface. If there are no inner boundaries, the parameter N (length of CTRLST) shall be set to zero. All the contours shall be in the same plane and shall not intersect each other. All the contours that corresponds to internal boundaries shall belong to the bounded surface defined by the **api_contour** CTRNAM that corresponds to the external boundary and none of them shall belong to the bounded surface defined by another **api_contour**. That is, the **api_planar_surface** shall be arcwise connected. If these conditions hold, the **api_planar_surface** is computed by the interface. All **api_contour** entities are duplicated and they have a **null_style** assigned to them. Then:

1) The **plane** of the surface is computed by its **position**. The **position.location** shall be the first point of the first **composite_curve_segment** of the **api_contour** CTRNAM. The x axis of the **position,** that is the **position.p[1]**, is the tangent to this **composite_curve_segment** in the sense defined by its **same_sense** attribute. The z axis of the **position**, that is the **position.p[3]** is computed as orthogonal to the plane that contains the **api_contour** CTRNAM. Its sense is defined in such a way that this **api_contour** is oriented counter clockwise with respect to this oriented axis.

2) For each **api_contour** that defines the **api_planar_surface**, a **surface_curve** is instantiated, each one referring to this **api_contour** as its **curve_3d**. The **associated_geometry** attribute of this **surface_curve** contains only one element that is the **plane** of the **api_planar_surface** computed in step 1. The **master_representation** attribute of this **surface_curve** equals **curve_3d.**

3) For each computed **surface_curve**, a closed **composite_curve_segment** is instantiated. This **composite_curve_segment**

— refers to the **surface_curve** it corresponds to, as its **parent_curve**;

— contains the **transition** value of the last **composite_curve_segment** of the **api_contour** that is the **curve_3d** of the **surface_curve** it corresponds to, as its **transition** attribute;

— contains a **same_sense** attribute whose value equals **true** for the **composite_curve_segment** that correspond to the external boundary, and whose value is computed to ensure that all the other closed **composite_curve_segment**s are oriented clockwise with respect to the z axis of the **plane** of the **api_planar_surface** (i.e., the **position.p[3]**, see step 1).

4) an **outer_boundary_curve** is then instantiated whose **segments** contains only one element : the **composite_curve_segment** whose **parent_curve** refers, as its **curve_3d**, to the **api_contour** that corresponds to the external boundary of the **api_planar_surface.**

5) For each other **composite_curve_segment**, a **boundary_curve** is instantiated whose **segments** contains only this **composite_curve_segment**.

6) Finally, the **api_planar_surface** is instantiated. Its **basis_surface** is the **plane** of the **api_planar_surface**. Its **boundaries** are the (possible) **boundary_curve**s and the **outer_boundary_curve** computed in step 5 and 4. Its **implicit_outer** attribute equals **false**. This **api_planar_surface** has the current *surface_style* entry from the interface status table assigned to, and the name of this **api_planar_surface** is returned. All other entities have a **null_style** assigned to them.

Notes:

1) The creation of an **api_planar_surface** is allowed for 3D views only, i.e., when the interface is initialised with *geometrical_power_level* is greater than 2.

2) In case of no inner boundaries (length of list CTRLST equal to zero), the parameter CTRLST shall be ignored.

3) If there are two or more **api_contour**s in the specification of the **api_planar_surface**, the distance between two **api_contour**s shall not be in the range [ZERO_VALUE,EPS].

Internal Reference:

6.1.15, 6.1.16, **6.1.17.1**, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
5	integer value out of permitted range	101	attempt to create a degenerated entity
104	distance between two contours less than EPS	119	given entities are not in the same plane
123	an intersection of given contours detected	125	an overlapping of given contours detected
201	temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
207	maximal number of inner boundaries exceeded	1001	enumerated value out of range

A.5.6 Geometric solid entities

A.5.6.1 CSG Primitives

Generation of a sphere	Sph_Gen
Generation of a cone	Con_Gen
Generation of a cylinder	Cyl_Gen
Generation of a torus	Tor_Gen
Generation of a block	Blk_Gen
Generation of a wedge	Wdg_Gen

A.5.6.1.1 Generation of a sphere

Function name:

Sph_Gen

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	RAD	D	radius of the **sphere**	(EPS ≤ RAD ≤ MAX)
in	CNTPNT	N	name of **cartesian_point**	pnt
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **sphere**	sph

FORTRAN binding:

```
NAME = SPH_GEN ( RAD,CNTPNT,KFIX )
```

Effects:

Creation of a **sphere** entity that is a CSG primitive with a spherical shape defined by a **cartesian_point** CNTPNT as a centre and a radius RAD. The centre point CNTPNT is duplicated as **p1** and it has a **null_style** assigned to it. Then:

— a **sphere** is instantiated with **radius** equals RAD and **p1** as **centre**. This **sphere** has one **presentation_style_assignment** that contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this **sphere** is returned.

The radius RAD is measured in *OVC_length_units* and shall be in range [EPS ,MAX]. If an error occurs, no entity is created and the element name is returned as zero.

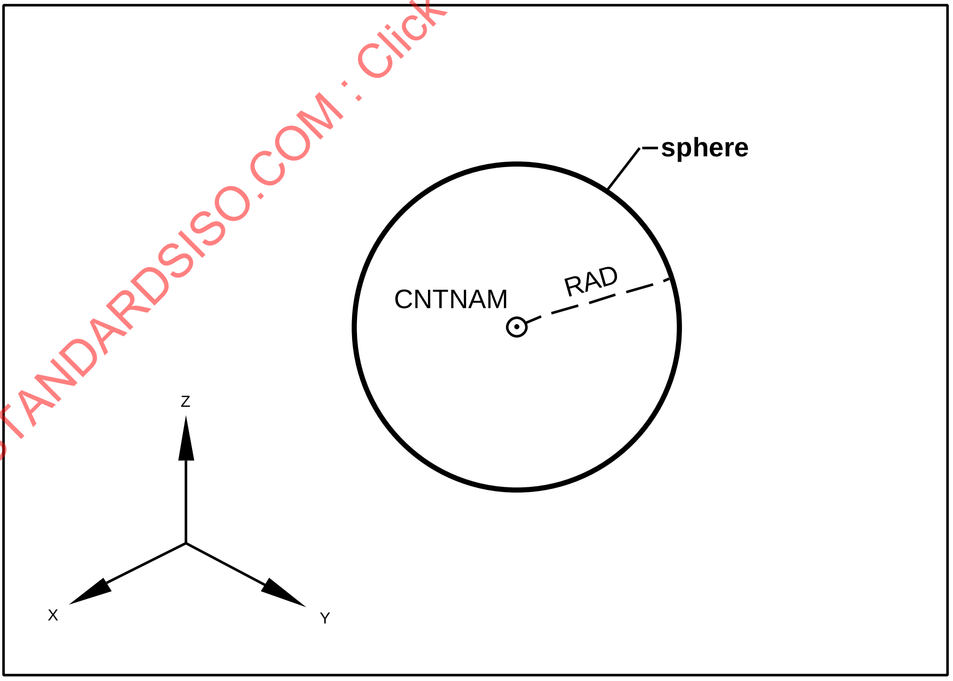


Figure A. 38 — Function: Sph_Gen

Notes:
–

Internal Reference:
6.1.9, **6.1.18.4.1**, 6.2.4, 6.2.3.2, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	201	temporary database overflow
202	error while sending entity to CAD system	203	function not compatible with implemented interface level
204	function not compatible with current power level	1001	enumerated value out of range

A.5.6.1.2 Generation of a cone

Function name:
Con_Gen

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	ANGLE	D	semi angle, that is the angle between central axis and the conical surface	(0° ≤ ANGLE ≤ 90°)
in	HEIGHT	D	height of **cone**	(EPS ≤ HEIGHT ≤ MAX)
in	RAD	D	radius of the cone at the point of the cone axis	(0.0 OR (EPS ≤ RAD ≤ MAX))
in	A1PNAM	N	name of **axis1_placement**	a1p
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **right_circular_cone**	con

FORTRAN binding:

```
NAME = CON_GEN ( ANGLE,HEIGHT,RAD,A1PNAM,KFIX )
```

Effects:
Creation of a **right_circular_cone** entity that is a CSG primitive in the form of a cone that may be truncated. The generation is specified by an **axis1_placement** A1PNAM, the semi-angle ANGLE of the cone and a height HEIGHT. The **axis1_placement** defines the direction of the central axis of symmetry of the cone and a point on this axis, that is on one of the planar circular faces or, if radius is zero, at the apex. In addition a radius is given, that, if non zero, gives the size and location of a truncated face of the cone. If the radius RAD is greater than EPS, the height HEIGHT specifies the distance between the planar circular faces of the cone; if the radius is equal to zero, the length defines the distance from the base to the apex. The given **axis1_placement** is duplicated as **a1p1** and it has a **null_style** assigned to it. Then:

— a **right_circular_cone** is instantiated with **position** as **a1p1**, **height** equals HEIGHT, **radius** equal to RAD and **semi_angle** equals ANGLE. This **right_circular_cone** has one **presentation_style_assignment**, that contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this **right_circular_cone** is returned.

HEIGHT and RAD are measured in *OVC_length_units*. RAD, shall be either zero or in range [EPS,MAX]. The given angle is counted in *OVC_angle_units*. If an error occurs, no entity is created and the element name is returned as zero.

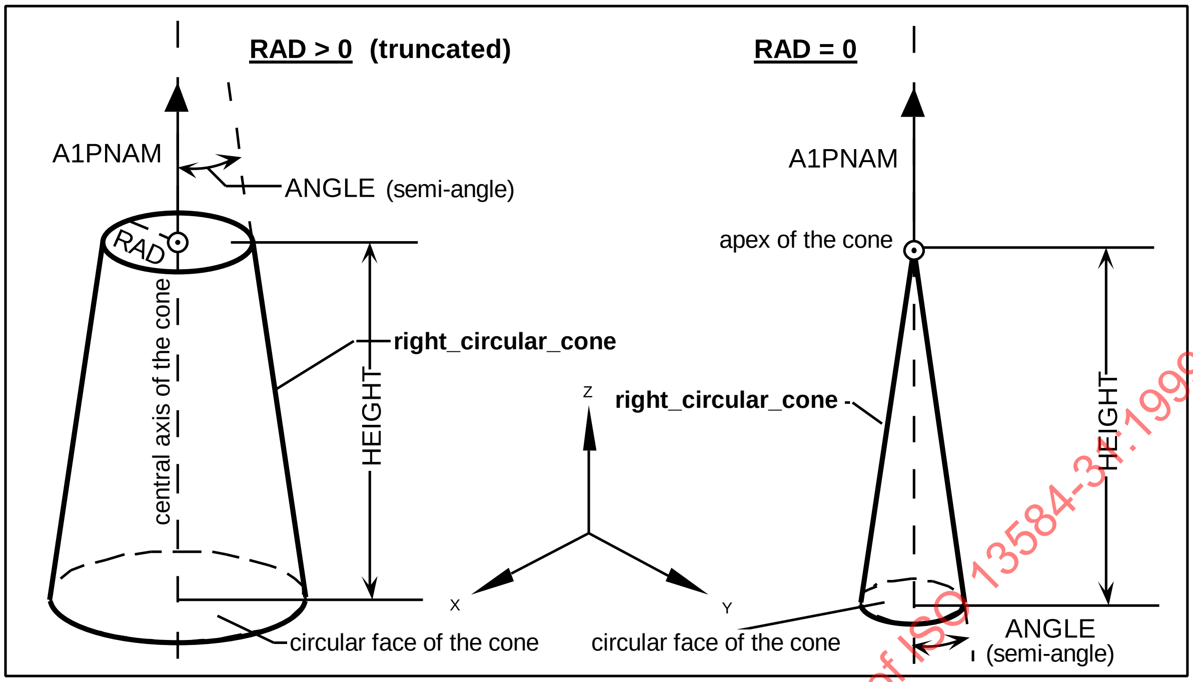


Figure A. 39 — Function: Con_Gen

Notes:
–

Internal Reference:
6.1.9, **6.1.18.4.2**, 6.2.4, 6.2.3.2, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	4	value for plane angle measure out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.6.1.3 Generation of a cylinder

Function name:

Cyl_Gen

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	RAD	D	radius of **cylinder**	(EPS ≤ RAD ≤ MAX)
in	HEIGHT	D	height of **cylinder**	(EPS ≤ HEIGHT ≤ MAX)
in	A1PNAM	N	name of **axis1_placement**	a1p
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **right_circular_cylinder**	cyl

FORTRAN binding:

```
NAME = CYL_GEN ( RAD,HEIGHT,A1PNAM,KFIX )
```

Effects:

Creation of a **right_circular_cylinder** entity, which is a CSG primitive specified by a radius RAD, a height HEIGHT and an **axis1_placement** A1PNAM. The **axis1_placement** defines the axis of the cylinder and the centre point of one planar circular face. The height is the distance from the first circular face centre in the positive direction of the axis to the second circular face centre. The given **axis1_placement** is duplicated as **a1p1** and it has a **null_style** assigned to it. Then:

— a **right_circular_cylinder** is instantiated with **position** as **a1p1**, **height** equals HEIGHT and **radius** equal to RAD. This **right_circular_cylinder** has one **presentation_style_assignment**, which contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this **right_circular_cylinder** is returned.

HEIGHT and RAD are measured in *OVC_length_units*. RAD and HEIGHT shall be in range [EPS ,MAX]. If an error occurs, no entity is created and the element name is returned as zero.

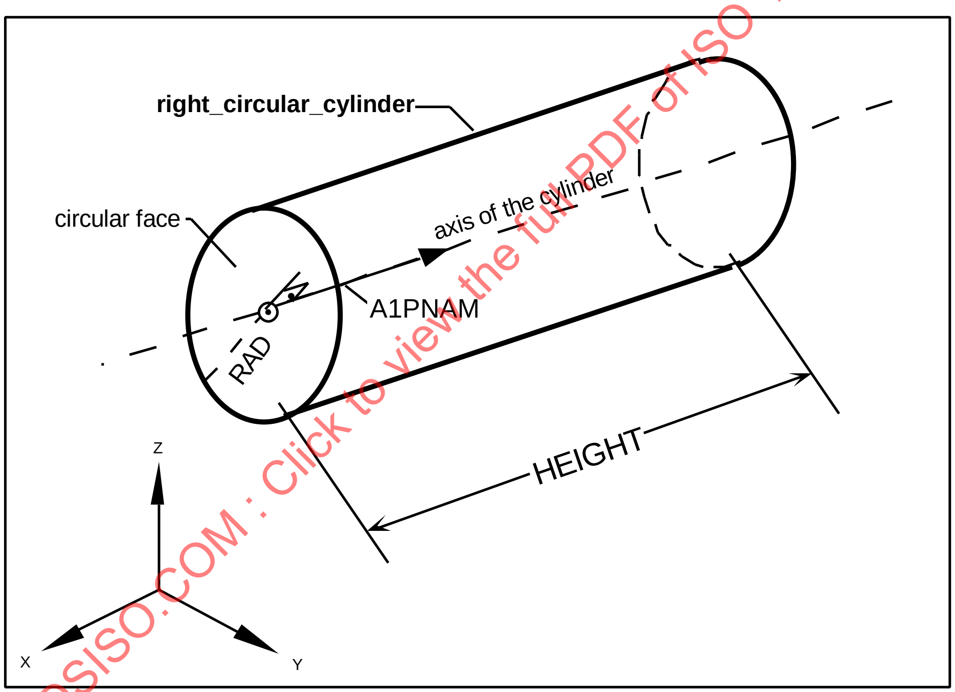


Figure A. 40 — Function: Cyl_Gen

Notes:

–

Internal Reference:

6.1.9, **6.1.18.4.3**, 6.2.4, 6.2.3.2, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	201	temporary database overflow
202	error while sending entity to CAD system	203	function not compatible with implemented interface level
204	function not compatible with current power level	1001	enumerated value out of range

A.5.6.1.4 Generation of a torus

Function name:

Tor_Gen

interface level:	3
geometrical power level:	3

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	MAJOR	D	radius of directrix of the **torus**	(MINOR ≤MAJOR≤ MAX)
in	MINOR	D	radius of generatrix of the **torus**	(EPS ≤ MINOR ≤ MAX)
in	A1PNAM	N	name of **axis1_placement**	a1p
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **torus**	tor

FORTRAN binding:

```
NAME = TOR_GEN ( MAJOR,MINOR,A1PNAM,KFIX )
```

Effects:

Creation of a **torus** entity, which is a CSG primitive specified by sweeping the area of a circle (the generatrix) with the radius MINOR about a larger circle (the directrix) with the radius MAJOR. The centre and plane of the directrix is defined by the **axis1_placement** A1PNAM. The given **axis1_placement** is duplicated as **a1p1** and it has a **null_style** assigned to it. Then:

— a **torus** is instantiated with **position** as **a1p1**, **major_radius** equals MAJOR and **minor_radius** equal to MINOR. This **torus** has one **presentation_style_assignment**, which contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this **torus** is returned.

The radii are measured in *OVC_length_units* and shall be in range [EPS ,MAX]. If an error occurs, no entity is created and the element name is returned as zero.

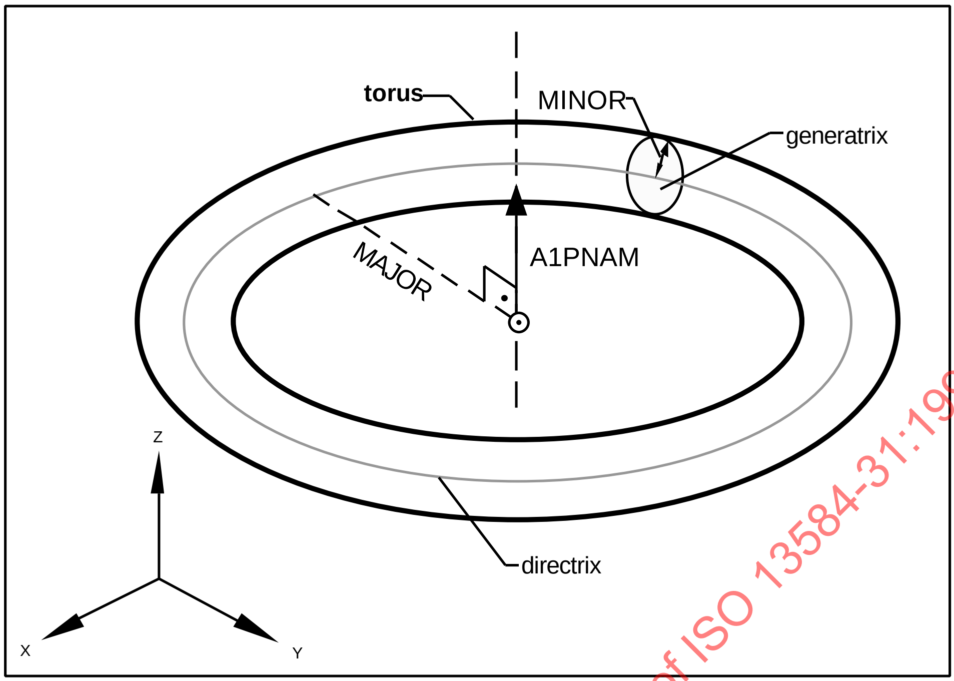


Figure A. 41 — Function: Tor_Gen

Notes:
–

Internal Reference:
6.1.9, **6.1.18.4.4**, 6.2.4, 6.2.3.2, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	201	temporary database overflow
202	error while sending entity to CAD system	203	function not compatible with implemented interface level
204	function not compatible with current power level	1001	enumerated value out of range

A.5.6.1.5 Generation of a block

Function name:
Blk_Gen

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	LENX	D	**block** length along the x-axis	(EPS ≤ LENX ≤ MAX)
in	LENY	D	**block** length along the y-axis	(EPS ≤ LENY ≤ MAX)
in	LENZ	D	**block** length along the z-axis	(EPS ≤ LENZ ≤ MAX)
in	A2PNAM	N	name of **axis2_placement_3d**	a2p
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **block**	blk

FORTRAN binding:

```
NAME = BLK_GEN ( LENX,LENY,LENZ,A2PNAM,KFIX )
```

Effects:

Creation of a **block** entity that is a CSG primitive specified as a solid rectangular parallel piped (block), defined with a location and placement coordinate system. The block is specified by the positive length LENX, LENY and LENZ along the axes of the placement coordinate system defined by the **axis2_placement_3d** A2PNAM. The block has one vertex at the origin of the placement coordinate system. The given **axis2_placement_3d** is duplicated as **a2p1** and it has a **null_style** assigned to it. Then:

— a **block** is instantiated with **position** as **a2p1**, **x, y and z** equals LENX, LENY and LENZ respectively. This **block** has one **presentation_style_assignment** that contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this **block** is returned.

The values of the block lengths are measured in *OVC_length_units* and shall be in range [EPS,MAX]. If an error occurs, no entity is created and the element name is returned as zero.

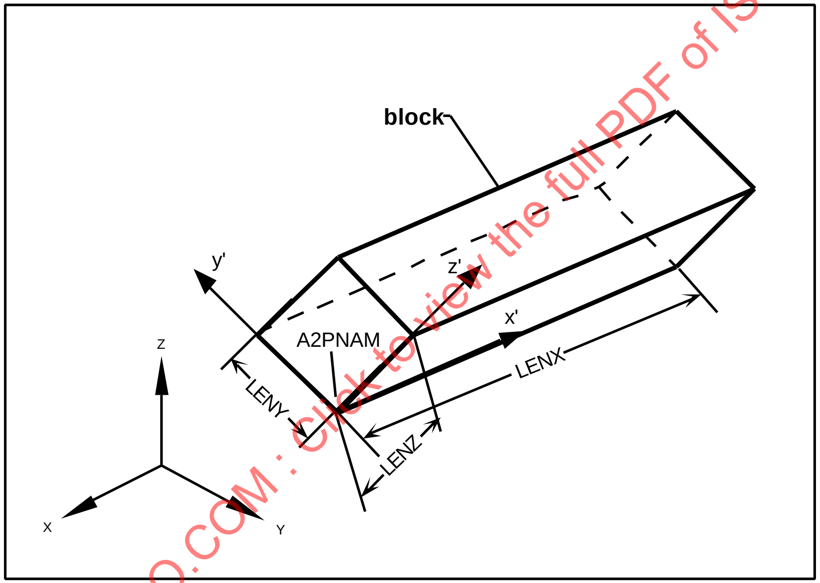


Figure A. 42 — Function: Blk_Gen

Notes:

–

Internal Reference:

6.1.9, **6.1.18.4.5**, 6.2.4, 6.2.3.2, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	201	temporary database overflow
202	error while sending entity to CAD system	203	function not compatible with implemented interface level
204	function not compatible with current power level	1001	enumerated value out of range

A.5.6.1.6 Generation of a wedge

Function name:

Wdg_Gen

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	LENX	D	size of wedge long the x-axis	(EPS ≤ LENX ≤ MAX)
in	LENY	D	size of wedge long the y-axis	(EPS ≤ LENY ≤ MAX)
in	LENZ	D	size of wedge long the z-axis	(EPS ≤ LENZ ≤ MAX)
in	LTX	D	height of smaller surface of the wedge	(0.0 OR (EPS ≤ LTX ≤ MAX))
in	A2PNAM	N	name of **axis2_placement_3d**	a2p
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **right_angular_wedge**	wdg

FORTRAN binding:

```
NAME = WDG_GEN ( LENX,LENY,LENZ,LTX,A2PNAM,KFIX )
```

Effects:

Creation of a **right_angular_wedge** entity that is a CSG primitive specified as a solid that is a result of intersection a **block** with a **plane** perpendicular to one of its faces. The triangular/trapezoidal face is defined by positive length LENX and LENY along the placement X- and Y axis, by the length LTX (if non zero) parallel to the (Ox) axis at a distance LENY from the placement origin and by the line connecting the ends of the LENX and LTX segments. The positive length LENZ defines a distance along the (Oz) axis through that the trapezoid or triangle is extruded.

The **axis2_placement_3d** A2PNAM is duplicated as **a2p1** and it has a **null_style** assigned to it. Then:

— a **right_angular_wedge** is instantiated with **position a2p1** and the given values of LENX, LENY, LENZ and LTX. This **right_angular_wedge** has one **presentation_style_assignment**, that contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this **right_angular_wedge** is returned.

The values of the lengths are measured in *OVC_length_units* and LENX, LENY and LENZ shall be in range [EPS,MAX]. The value of LTX is allowed to be equal to zero. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) If LTX = 0.0, the wedge has five faces and the basic face is a triangle; otherwise, it has six faces and the basic face is a trapezoid.

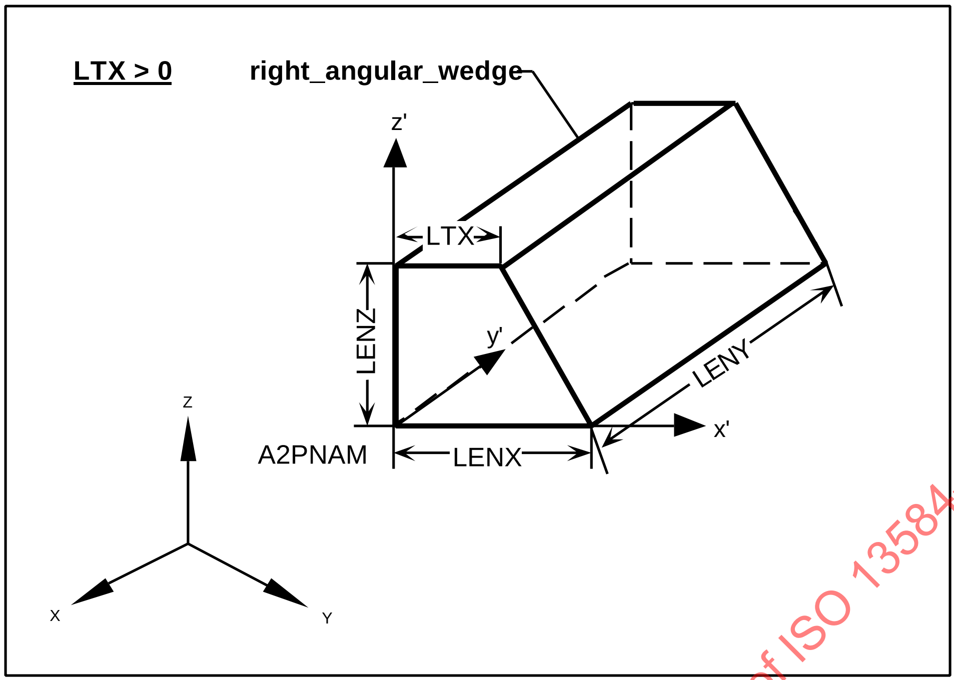


Figure A. 43 — Function: Wdg_Gen

Internal Reference:
6.1.9, **6.1.18.4.6**, 6.2.4, 6.2.3.2, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	201	temporary database overflow
202	error while sending entity to CAD system	203	function not compatible with implemented interface level
204	function not compatible with current power level	1001	enumerated value out of range

A.5.6.2 CSG regularised Boolean operations

Union of solids	Union_Sld
Intersection of solids	Intersection_Sld
Difference of solids	Difference_Sld

A.5.6.2.1 Union of solids

Function name:
Union_Sld

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	BOPNM1	N	name of first **boolean_operand**	solids
in	BOPNM2	N	name of second **boolean_operand**	solids
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **boolean_result**	brs

FORTRAN binding:

```
NAME = UNION_SLD ( BOPNM1,BOPNM2,KFIX )
```

Effects:

Performs the regularised union operation of solids. The **boolean_operand**s BOPNAM1 and BOPNM2 are duplicated as **b1** and **b2** and they have a **null_style** assigned to them. A **boolean_operator o** equal to union is instantiated. Then a **boolean_result** is created with **first_operand b1**, **second_operand b2** and **operator o**. This **boolean_result** has one **presentation_style_assignment**, that contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this **boolean_result** is returned.

The **boolean_operand**s shall not consist of several non-connected parts. The union shall not be empty. If an error occurs, no entity is created and the element name is returned as zero.

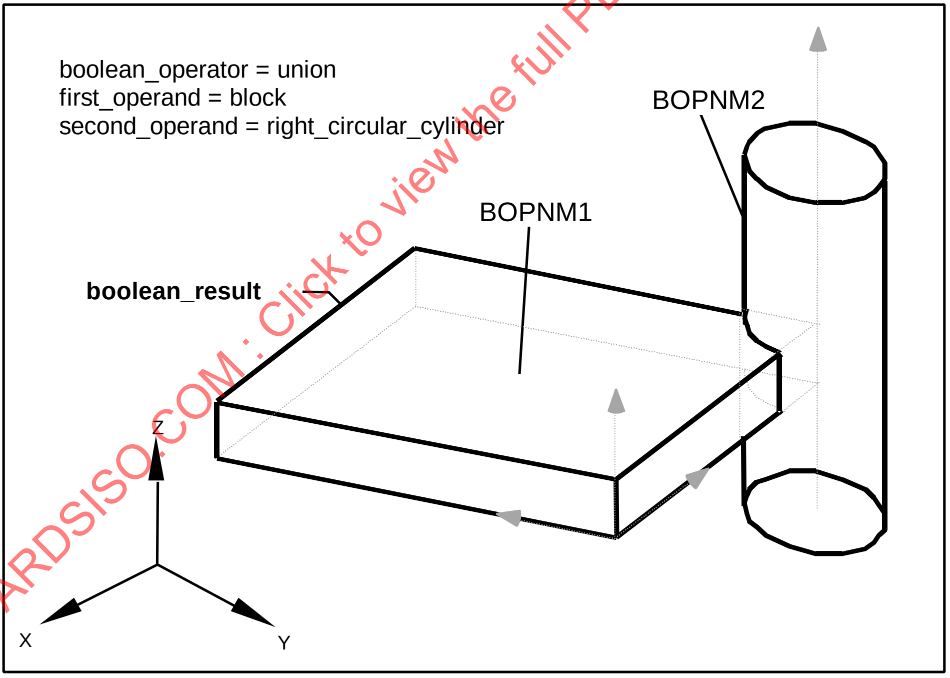


Figure A. 44 — Function: Union_Sld

Notes:

–

Internal Reference:

6.1.18, **6.1.18.3,** 6.2.3.2, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
101	attempt to create a degenerated entity	130	Boolean operation failed
201	temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.6.2.2 Intersection of solids

Function name:

Intersection_Sld

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	BOPNM1	N	name of first **boolean_operand**	solids
in	BOPNM2	N	name of second **boolean_operand**	solids
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **boolean_result**	brs

FORTRAN binding:

```
NAME = INTERSECTION_SLD ( BOPNM1,BOPNM2,KFIX )
```

Effects:

Performs the regularised intersection of two solids. The **boolean_operand**s BOPNM1 and BOPNM2 are duplicated as **b1** and **b2** and they have a **null_style** assigned to them. A **boolean_operator o** equals intersection is instantiated. Then a **boolean_result** is created with **first_operand b1**, **second_operand b2** and **operator o.** This **boolean_result** has one **presentation_style_assignment**, that contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this **boolean_result** is returned.

The result may be composed of several separate parts. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

–

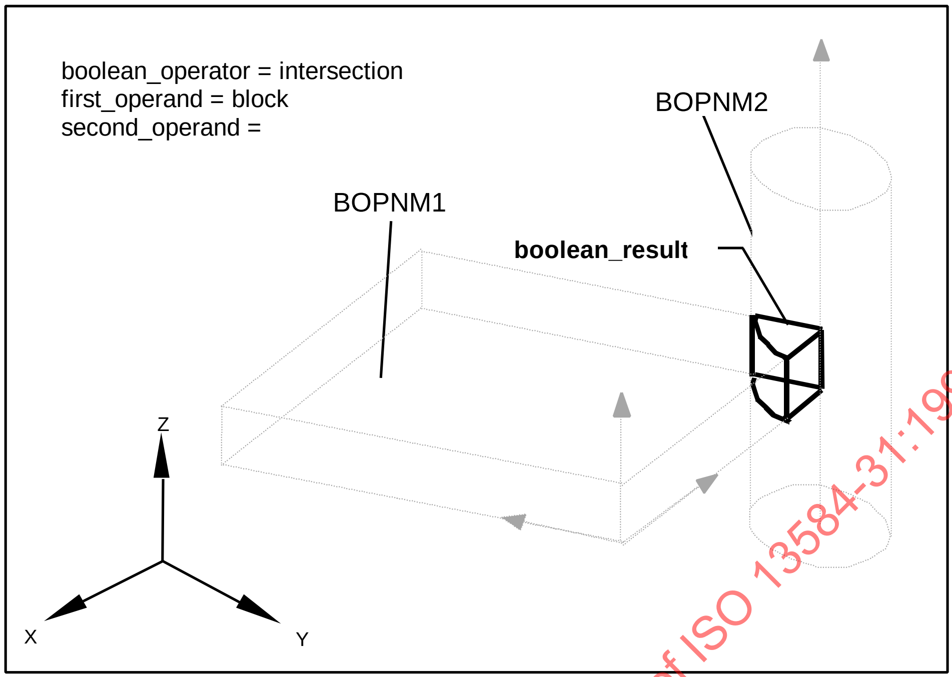


Figure A. 45 — Function: Intersection_Sld

Internal Reference:

6.1.18, **6.1.18.3,** 6.2.3.2, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
101	attempt to create a degenerated entity	130	Boolean operation failed
201	temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.6.2.3 Difference of solids

Function name:

Difference_Sld

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	BOPNM1	N	name of first **boolean_operand**	solids
in	BOPNM2	N	name of second **boolean_operand**	solids
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **boolean_result**	brs

FORTRAN binding:

```
NAME = DIFFERENCE_SLD ( BOPNM1,BOPNM2,KFIX )
```

Effects:

Performs the regularised difference of two **boolean_operand**s. The **boolean_operand**s BOPNAM1 and BOPNM2 are duplicated as **b1** and **b2** and they have a **null_style** assigned to them. A

boolean_operator o equals difference is instantiated. Then a **boolean_result** is created with **first_operand b1**, **second_operand b2** and **operator o.** This **boolean_result** has one **presentation_style_assignment**, that contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this **boolean_result** is returned.

The result may be composed of several separate parts. If an error occurs, no entity is created and the element name is returned as zero.

Notes:
–

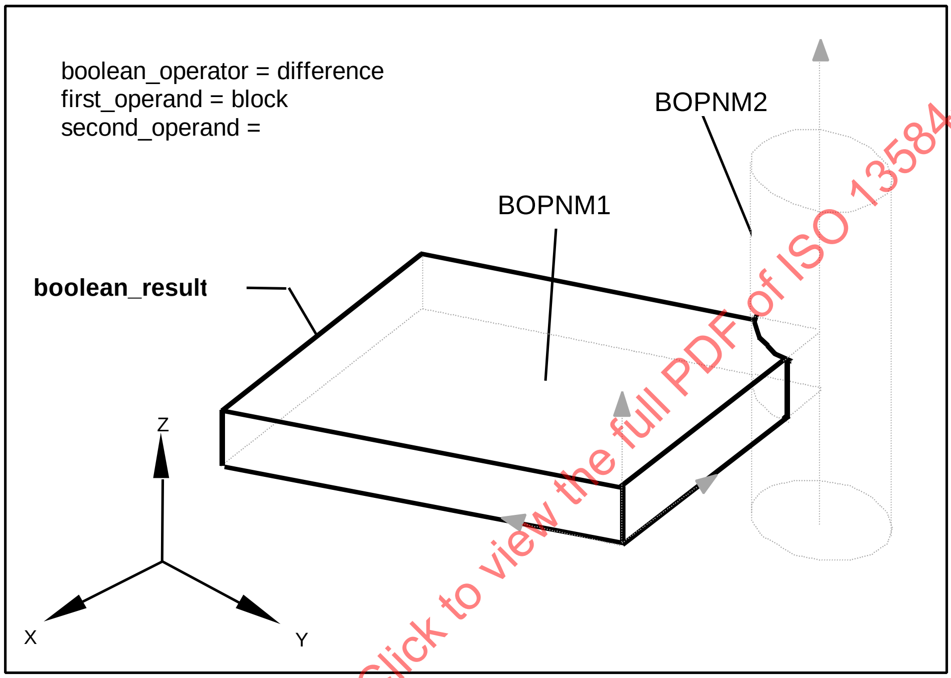


Figure A. 46 — Function: Difference_Sld

Internal Reference:
6.1.18, **6.1.18.3,** 6.2.3.2, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
101	attempt to create a degenerated entity	130	Boolean operation failed
201	temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.6.3 Swept_area solid entities

Extrusion	Sld_Extrusion
Revolution	Sld_Revolution

A.5.6.3.1 Extrusion

Function name:

Sld_Extrusion

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	SRFNAM	N	name of an **api_planar_surface**	aps
in	DIRNAM	N	direction of the sweep	dir
in	DEPTH	D	length of translation	(EPS ≤ DEPTH ≤ MAX)
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **extruded_area_solid**	eas

FORTRAN binding:

```
NAME = SLD_EXTRUSION ( SRFNAM,DIRNAM,DEPTH,KFIX )
```

Effects:

Creation of an **extruded_area_solid** entity by a sweeping of a bounded planar surface, that is the **api_planar_surface** SRFNAM. The direction of the translation is defined by a **direction** vector DIRNAM, and the length of the translation is defined by a distance DEPTH.

The **direction** DIRNAM is duplicated as **d** and it has a **null_style** assigned to it. Then:

— The **api_planar_surface** SRFNAM is duplicated as **s** and it has a **null_style** assigned to it.

— An **extruded_area_solid** is instantiated with **swept_area s**, **extruded_direction d** and **depth** equal to DEPTH. This **extruded_area_solid** has one **presentation_style_assignment**, that contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this **extruded_area_solid** is returned.

The length DEPTH shall be in range [EPS,MAX]. The planar area may have holes that will sweep into holes in the solid. If an error occurs, no entity is created and the element name is returned as zero.

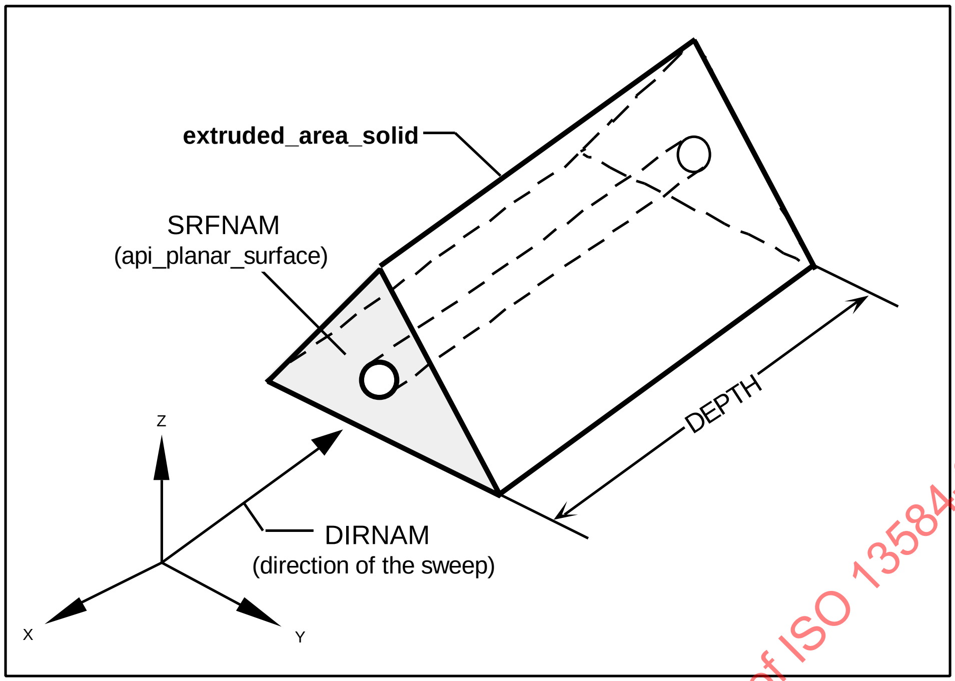


Figure A. 47 — Function: Sld_Extrusion

Notes:
–

Internal reference:
6.1.9, 6.1.17, **6.1.18.6**, 6.2.3.2, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	101	attempt to create a degenerated entity
201	temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.6.3.2 Revolution

Function name:
Sld_Revolution

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	SRFNAM	N	name of an **api_planar_surface**	aps
in	ANG	D	angle of rotation	(0° < ANG ≤ 360°)
in	A1PNAM	N	name of **axis1_placement**	a1p
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **revolved_area_solid**	ras

FORTRAN binding:

```
NAME = SLD_REVOLUTION ( SRFNAM,ANG,A1PNAM,KFIX )
```

Effects:

Creation of a **revolved_area_solid** entity that is formed by revolving a planar bounded surface, that is the **api_planar_surface** SRFNAM, about an axis. The bounded surface is revolved clockwise about the axis A1PNAM with the angle ANG.

The **axis1_placement** is duplicated as **a** and it has a **null_style** assigned to it. Then:

— The **api_planar_surface** SRFNAM is duplicated as **s** and it has a **null_style** assigned to it.

— A **revolved_area_solid** is instantiated with **swept_area s**, **axis a** and **angle** equals ANG. This **revolved_area_solid** has one **presentation_style_assignment**, that contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this **revolved_area_solid** is returned.

The angle is measured in *OVC_angle_unit*s, counted clockwise when viewing along the axis in positive direction, beginning at the given **api_planar_surface**. The planar area may have holes that will sweep into holes in the solid. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

–

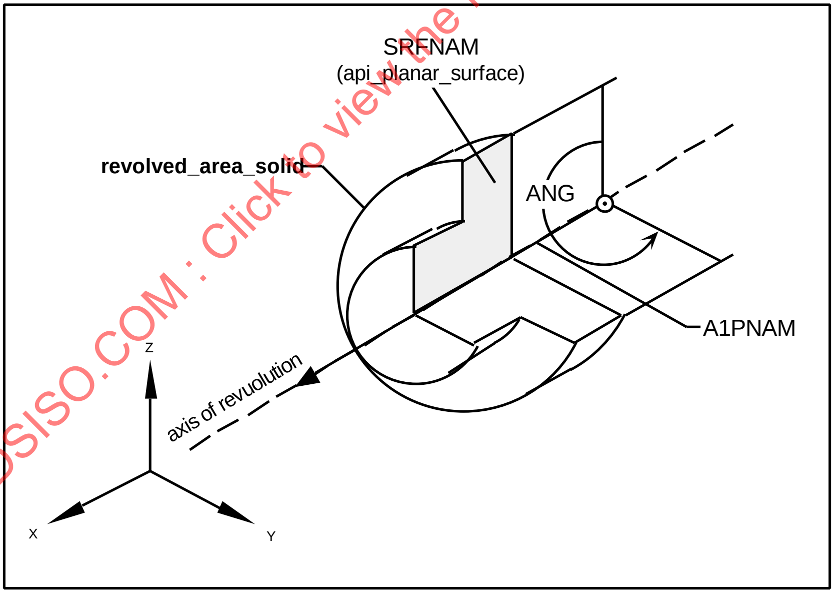


Figure A. 48 — Function: Sld_Revolution

Internal Reference:

6.1.9, 6.1.17, **6.1.18.7**, 6.2.3.2, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
4	value for plane angle measure out of permitted range	101	attempt to create a degenerated entity
124	an intersection between axis an plane of surface detected	126	axis of revolution not in plane of surface
201	temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
1001	enumerated value out of range		

A.5.6.4 CSG solid pipe entity

Generation of a pipe Sld_Pipe

A.5.6.4.1 Generation of a pipe

Function name:

Sld_Pipe

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	PLNNAM	N	name of the **polyline**	pln
in	SRFNAM	N	name of an **api_planar_surface**	aps
in	RAD	D	rounding radius	(EPS ≤ RAD ≤ MAX)
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of created **boolean_result**	brs

FORTRAN binding:

```
NAME = SLD_PIPE ( PLNNAM,SRFNAM,RAD,KFIX )
```

Effects:

Creation of a **boolean_result** formed by sweeping a planar bounded surface, that is the **api_planar_surface** SRFNAM, about the directrix defined by rounding the guide line **polyline** with fillets of radius RAD. The rounding process will be automatically created and self intersections are checked.

The **api_planar_surface** SRFNAM is duplicated as **s** and it has a **null_style** assigned to. The **polyline** with n+1 points is divided in several **api_lines** $\mathbf{l_1}$,...,$\mathbf{l_n}$ that are connected by the **api_circular_arcs** $\mathbf{a_1}$,...,$\mathbf{a_{n-1}}$ with the radius RAD. All these instances have a **null_style** assigned to them. Then:

— An extruded_area_solid e_1 is generated from the planar surface s with extruded_direction from api_line.basis_curve.dir and depth as difference of the parameters of api_line.trim[1] and api_line.trim[2] from api_line l_1.

— An axis1_placement a1p_1 is instantiated from api_circular_arc.basis_curve.position.p[3] a_1 and the rotation angle ang$_1$ is retrieved from the difference of the parameters of api_circular_arc.trim[1] and api_circular_arc.trim[2] from api_circular_arc a_1. This axis1_placement has a null_style assigned to it. A revolved_area_solid r_1 is instantiated from the planar surface s with axis a1p_1 and angle ang$_1$.

— A boolean_result b_1 is instantiated with the first_operand e_1, second_operand r_1 and boolean_operator equals union.

— For the next api_lines l_j, j=2,...,n api_circular_arcs a_k, k = 2,...,n-1 and i=2,...,2n-2 the following process is performed:

- — An extruded_area_solid ej is generated from the planar surface s with extruded_direction from api_line.basis_curve.dir and depth as difference of the parameters of api_line.trim[1] and api_line.trim[2] from api_line lj
- — A **boolean_result** $\mathbf{b_i}$ is instantiated with the **first_operand** $\mathbf{b_{(i-1)}}$, **second_operand** $\mathbf{e_j}$ and **boolean_operator** equal to union.
- — An axis1_placement a1p_k is instantiated from api_circular_arc.basis_curve.position.p[3] a_k and the rotation angle ang$_k$ is retrieved from the difference of the parameters of api_circular_arc.trim[1] and api_circular_arc.trim[2] from api_circular_arc a_k. This axis1_placement has a null_style assigned to. A revolution_area_solid r_k is instantiated from the planar surface s with axis a1p_k and angle ang$_k$.
- — A boolean_result $b_{(i+1)}$ is instantiated with the first_operand b_i, second_operand r_k and boolean_operator equals union. This boolean_result has one presentation_style_assignment, that contains the current entries of the interface status table for a *surface_style* and a *curve_style* assigned to it. The name of this boolean_result is returned.

(The indexes **i** and **j** are increased by one and the index **k** is increased by two).

The rounding radius is measured in *OVC_length_units* and shall be in range [EPS,MAX]. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) If the given **api_planar_surface** SRFNAM that is sweeping about the guide line has the shape of a circle or annulus, than the result of the interface function is a **boolean_result** called a ' Pipe '.

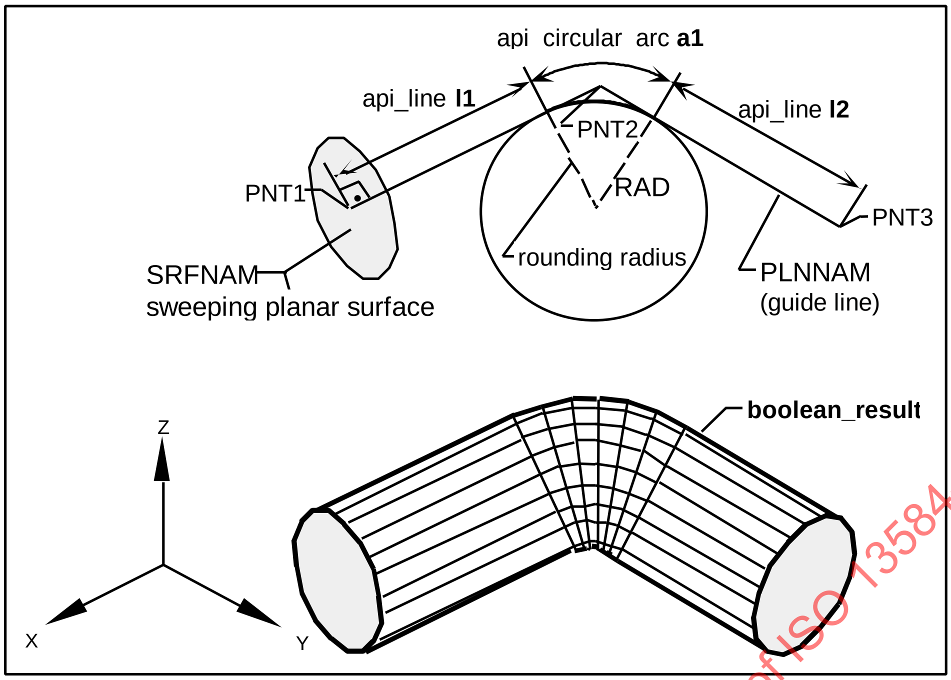


Figure A. 49 — Function: Sld_Pipe

Internal reference:

6.1.14, 6.1.17, 6.1.18, 6.2.3.2, 6.2.4, 8.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	101	attempt to create a degenerated entity
107	attempt to create a degenerated axis1_placement during entity creation	109	attempt to create a degenerated solid during entity creation
114	attempt to create an overlapping solid	130	Boolean operation failed
201	Temporary database overflow	202	error while sending entity to CAD system
203	function not compatible with implemented interface level	204	function not compatible with current power level
1001	Enumerated value out of range		

A.5.6.5 Half space solid entity

Generation of a half_space_solid Hss_Gen

A.5.6.5.1 Generation of a half_space_solid

Function name:

Hss_Gen

interface level:	**3**
geometrical power level:	**3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	A2PNAM	N	name of **axis2_placement_3d**	a2p
in	AGREMF	E	**agreement_flag**	[TRUE,FALSE]
out	NAME	N	name of created **half_space_solid**	hss

FORTRAN binding:

```
NAME = HSS_GEN ( A2PNAM,AGREMF )
```

Effects:

An **half_space_solid** is defined by the half space that is the regular subset of the domain that lies on one side of an unbounded surface. The side of the surface that is in the half space is determined by the surface normal and the agreement flag. If the agreement flag is TRUE, then the subset is the one the normal point away from. If the agreement flag is FALSE, then the subset is the one the normal point into.

The **axis2_placement_3d** is duplicated as **a** and it has a **null_style** assigned to it. Then:

— A **plane p** is instantiated with **position a** and it has a **null_style** assigned to it.

— A **half_space_solid** is instantiated with **base_surface** equals **p** and **agreement_flag** equals AGREMF. This **half_space_solid** has a **null_style** assigned to it.

The name of the entity is returned. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) An **half_space_solid** is not a sub-type of **solid_model**; **half_space_solid**s are only useful as operands in **boolean_expression**s. Therefore an **half_space_solid** is stored in the TDB only.

Internal Reference:

6.1.9, 6.1.16, 6.1.17.1, **6.1.18.8**, 6.2.1.2

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
201	temporary database overflow	203	function not compatible with implemented interface level
204	function not compatible with current power level	1001	enumerated value out of range

A.6 FUNCTIONS FOR STRUCTURE ENTITIES

A.6.1 Structure entities in the TDB

Create group	Create_Grp
Close group	Close_Grp
Reopen group	Reopen_Grp
Remove entity from group	Remove_Ent_Grp
Gathering entity into new group	Gather_Ent_Grp
Adding entity into group	Add_Ent_Grp

A.6.1.1 Create group

Function name:

Create_Grp

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
out	NAME	N	name of a created group	grp

FORTRAN binding:

```
NAME = CREATE_GRP ( )
```

Effects:

Creates a new **api_group** with attribute **name** equal to the empty string. The newly created **api_group** belongs to the current open group, is put on the top of the group stack and becomes the current open group. A name NAME, of the newly created **api_group**, is returned.

If an error occurs, no group is created and the name is returned as zero.

Notes:

–

Internal Reference:

5.4.1, 6.1.19, **6.1.19.1**

Errors:

201	temporary database overflow	204	function not compatible with current power level
208	maximal number of groups exceeded	210	group stack overflow

A.6.1.2 Close group

Function name:

Close_Grp

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
–				

FORTRAN binding:

```
CALL CLOSE_GRP ( )
```

Effects:

The **api_group** that is at the top of the group stack is closed. This group is removed from the group stack and the new top of the group stack become the current open group. If an error occurs, no group is closed.

Notes:

1) If the current open group is the root group, an error occurs and no group is closed.

Internal Reference:

5.4.1, 6.1.19, **6.1.19.1**

Errors:

204	function not compatible with current power level	301	attempt to close the root group

A.6.1.3 Reopen a group

Function name:

Reopen_Grp

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	GRPNAM	N	name of the **api_group**	grp

FORTRAN binding:

```
CALL REOPEN_GRP ( GRPNAM )
```

Effects:

This function reopens an existing **api_group**. This group shall not be the current open group and shall not be one of the groups in the stack of open groups. Afterwards the **api_group** GRPNAM is put on top of the group stack and becomes the current open group. This process does not change the group structure. All entities created in the TDB after the group reopening will belong to this group until its closure. If an error occurs, no group is reopened.

Notes:

–

Internal Reference:

5.4.1, 6.1.19, **6.1.19.1**

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
204	function not compatible with current power level	210	group stack overflow
302	attempt to reopen an already open group		

A.6.1.4 Remove entity from group

Function name:

Remove_Ent_Grp

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	ENTNAM	N	name of an entity to be removed from **api_group**	all graphical entities, grp

FORTRAN binding:

```
CALL REMOVE_ENT_GRP ( ENTNAM )
```

Effects:

This function removes an entity (geometric or structured) from the **api_group** it belongs to. Then it becomes a member of the root group. The entity ENTNAM is removed from the **items** set of the assigned **api_group_assignment** entity of its **api_group**. Then the entity is stored in the **items** set of the **api_group_assignment** that belongs to the root group. If an error occurs, no entity is removed.

Notes:
1) Removal of an entity from the root group is not allowed.

Internal Reference:
5.4.1, 6.1.19, **6.1.19.1**

Errors:

1	entity name not defined (zero, or unknown)	204	function not compatible with current power level
303	entity is member of root group		

A.6.1.5 Gathering entities into new group

Function name:

Gather_Ent_Grp

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	N	I	length of ENTLST	≥ 1
in	ENTLST	nxN	list of entities to be grouped	all graphical entities, grp
out	NAME	N	name of created **api_group**	grp

FORTRAN binding:

```
NAME = GATHER_ENT_GRP ( N,ENTLST )
```

Effects:
The function allows the gathering into a new group of a list of entities (geometric or structured). All these entities are removed from the group they belong to and are put into this new group. This new group shall belong to the current open group and is closed after the creation process is finish. The list of entities that are gathered into the new group shall not contain a group that contains the current open group.

A new **api_group** is instantiated with attribute **name** equal to the empty string and belongs to the current open group. The given entities are removed from the group they belong to and are stored in **items** set of the **api_group_assignment** entity. A name GRPNAM, of the newly created **api_group** is returned. If an error occurs, no group is created and the name is returned as zero.

Notes:
–

Internal Reference:
5.4.1, 6.1.19, **6.1.19.1**

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
5	integer value out of permitted range	201	temporary database overflow
204	function not compatible with current power level	208	maximal number of groups exceeded
210	group stack overflow	304	entity contains the current open group

A.6.1.6 Adding entity into group

Function name:

Add_Ent_Grp

interface level:	**1**

geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	GRPNAM	N	name of the **api_group**	grp
in	ENTNAM	N	name of entity to be added	all graphical entities, grp

FORTRAN binding:

```
CALL ADD_ENT_GRP ( GRPNAM,ENTNAM )
```

Effects:

The function allows the addition of an entity (geometric or structured) to the group GRPNAM. The entity is removed from its **items** set of the **api_group_assignment** that belongs to its **api_group**. Then the entity ENTNAM is stored in the **items** set of the **api_group_assignment** that belongs to **api_group** GRPNAM.

If the entity is a group, this group shall not contain the group whose name is given. If an error occurs, no modifications shall be done.

Notes:

–

Internal Reference:

5.4.1, 6.1.19, **6.1.19.1**

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
204	function not compatible with current power level	304	entity contains the current open group
305	attempt to create cyclical group structure		

A.6.2 Structure entities to sent into the CAD system

Open a set	Open_Set
Close a set	Close_Set

A.6.2.1 Open set

Function name:

Open_Set

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	SETNAM	S	name of set	

FORTRAN binding:

```
CALL OPEN_SET ( SETNAM )
```

Effects:

Creates a new **api_set** with attribute **name** derived from SETNAM as unique identifier that belongs to the current open set. It is put on the top of the set stack and becomes the current open set. If an error occurs, no set is created.

Notes:

1) The interface has to ensure that the name of **api_set** is unique.

Internal Reference:

5.4.2, 6.1.19, **6.1.19.3**

Errors:

204	function not compatible with current power level	209	maximal number of character per string exceeded
211	set stack overflow	306	name of set not unique

A.6.2.2 Close set

Function name:

Close_Set

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
–				

FORTRAN binding:

```
CALL CLOSE_SET ( )
```

Effects:

The **api_set** that is at the top of the set stack is closed. This set is removed from the stack and the new top of the stack become the current open set. If an error occurs, no set is closed.

Notes:

1) If the current open **api_set** is the root set, an error occurs and no set is closed.

2) A closed **api_set** shall never be reopened by the application program.

Internal Reference:

5.4.2, 6.1.19, **6.1.19.3**

Errors:

204	function not compatible with current power level	307	attempt to close the root set

A.7 FUNCTIONS FOR GEOMETRIC MANIPULATION OF ENTITIES

A.7.1 Duplicating entities

Duplicate entity	Dup_Ent

A.7.1.1 Duplicate entity

Function name:

Dup_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	ENTNAM	N	Name of an entity	all graphical entities,grp
in	KFIX	E	Storage location	[TDB,CAD]
out	NAME	N	Name of duplicated entity	same type as ENTNAM

FORTRAN binding:

```
NAME = DUP_ENT ( ENTNAM,KFIX )
```

Effects:

Creates a new entity by duplicating the given entity ENTNAM. The duplicated entity contains copies of all internal attributes of the original entity ENTNAM and is assigned the same presentation style. The name of the duplicated entity is returned. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) Manipulation of entities that are instances of type **fill_area_style_hatching** or **half_space_solid**, are only allowed for the usage within the TDB. For these kind of entities the manipulation will be performed, but the duplicated entity remain within in TDB (KFIX equals CAD, will be ignored).

2) Duplication of a **group** is a new **group**. If the **group** is not an empty **group**, than the contents of the **group** is also duplicated one by one.

Internal reference:

6.1, 6.2.1

Errors:

1	entity name not defined (zero, or unknown)	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
208	maximal number of groups exceeded	212	usage of entity only allowed for usage within the TDB
1001	enumerated value out of range		

A.7.2 Mirroring entities

Mirror entity	Mirror_Ent
Duplicate and mirror entity	Duplicate_Mirror_Ent

A.7.2.1 Mirror an entity

Function name:

Mirror_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	Meaning	permitted types/values
in	ENTNAM	N	name of an entity	all graphical entities,grp
in	LINNAM	N	name of **api_line**	lin

FORTRAN binding:

```
CALL MIRROR_ENT ( ENTNAM,LINNAM )
```

Effects:

Mirroring of an entity on an axis. The given **api_line**, LINNAM, defined the mirror axis by its attributes **pnt** and **dir** of its **basic_curve** entity **line**.

Specially for **conic** entities as given entity ENTNAM, that are in the context of **api_abstract_schema** an **api_circular_arc**, **api_elliptical_arc**, **api_hyperbolic_arc** or an **api_parabolic_arc** the following procedure will be done:

In the case of a 2D view:

— for **conic** entities, the **SELF\basis_curve.position** is first mirrored. Then a **conic** with the same parameters as the **basis_curve** of the conic arc is created using the mirrored **axis2_placement_2d**.

— a **trimmed_curve** is instantiated with a **sense_agreement** equal to the opposite of the **sense_agreement** of the initial conic arc and with trimming parameters computed by the interface to ensure that the mirrored entity is point to point mirrored from the initial conic arc.

see NOTE (1)
In the case of a 3D view:

— for **conic** entities, the right-handed **axis2_placement_3d SELF\basis_curve.position** is mirrored. Then a **conic** with the same parameters as the **basis_curve** of the conic arc is created using the mirrored **axis2_placement_3d**.

— a **trimmed_curve** is instantiated with a **sense_agreement** equal to the **sense_agreement** of the initial conic arc and with trimming parameters computed by the interface to ensure that the mirrored entity is point to point mirrored from the initial conic arc.

see NOTE (2)
The name of entity ENTNAM, all visual presentation (e.g. presentation style, view_level), group and set structure of the entity are not changed. If an error occurs, no modification will be done.

Notes:

1) If **trim_1** and **trim_2** are **cartesian_point**s, then the trimming parameters of the mirrored conic arc are the **cartesian_point**s that result from mirroring **trim_1** and **trim_2** in this order.

2) The sense of the **basis_curve** is changed, but not the **sense_agreement** nor the parametrisation.

3) If the given entity ENTNAM is an instance of type **api_group**, all entities that refer to this group shall be mirrored.

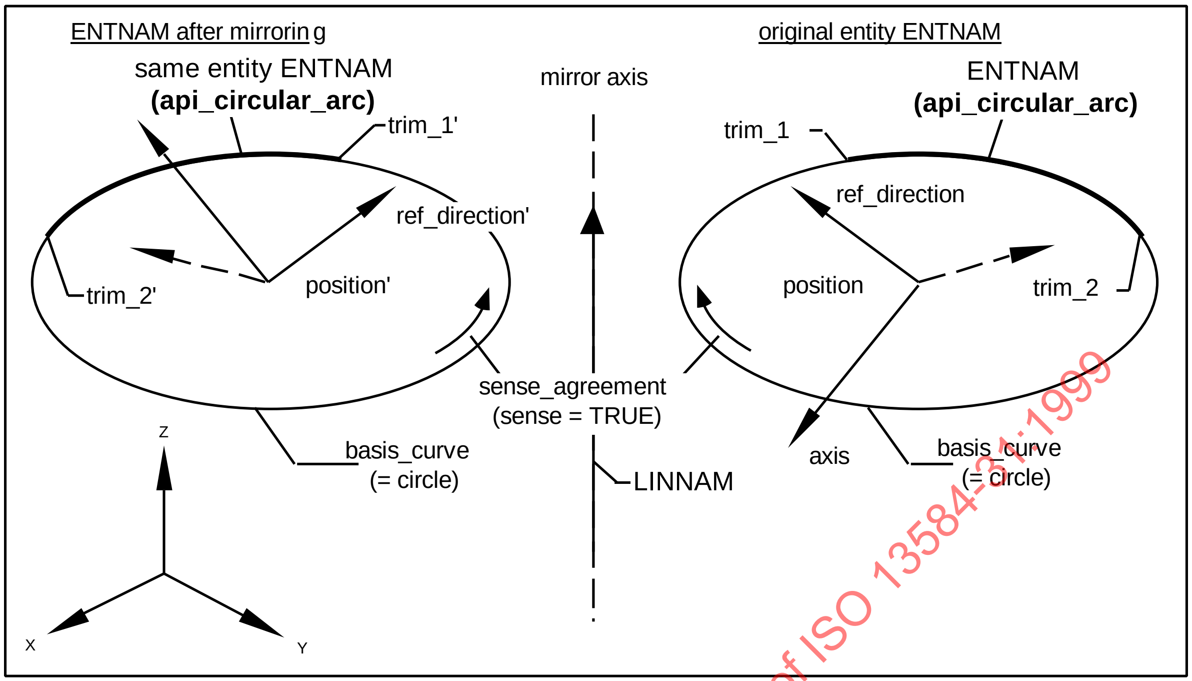


Figure A. 50 — Function: Mirror_Ent (3D view)

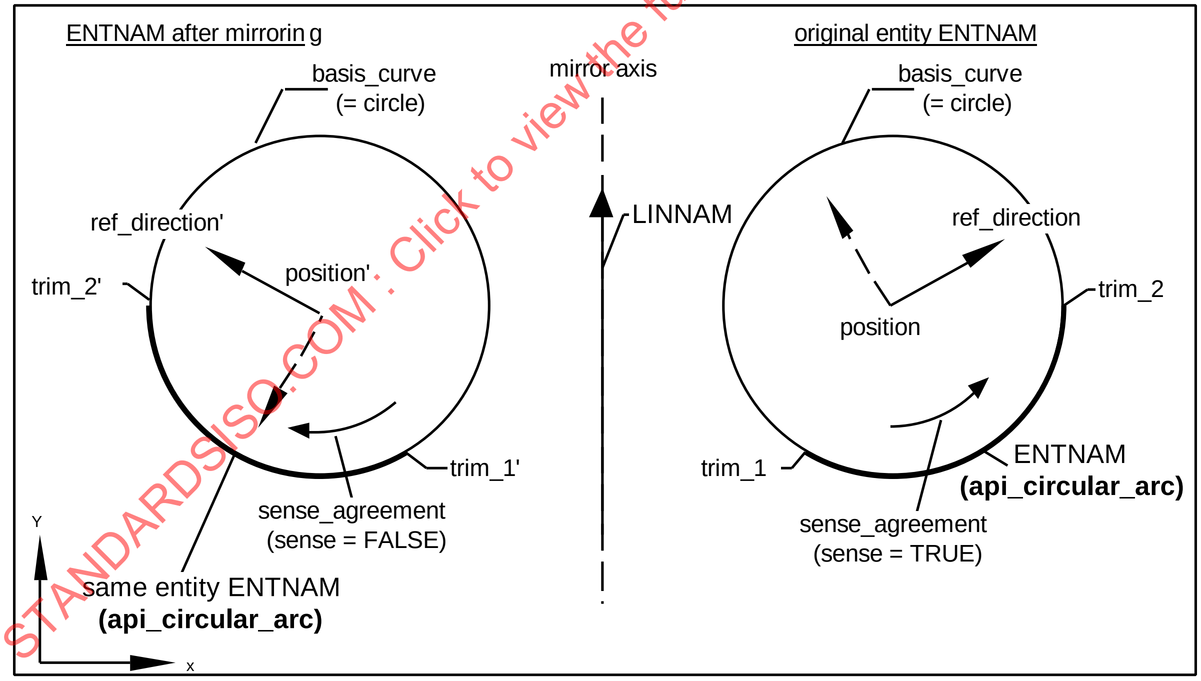


Figure A. 51 — Function: Mirror_Ent (2D view)

Internal Reference:
6.1, 6.2.1

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
204	function not compatible with current power level		

A.7.2.2 Duplicate and mirror an entity

Function name:

Dup_Mirror_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	Data type	meaning	permitted types/values
in	ENTNAM	N	name of an entity	all graphical entities,grp
in	LINNAM	N	name of **api_line**	lin
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of duplicated and mirrored entity	same type as ENTNAM

FORTRAN binding:

```
NAME = DUP_MIRROR_ENT ( ENTNAM,LINNAM,KFIX )
```

Effects:

This function invokes the function **Dup_Ent**. The returned entity is mirrored by the function **Mirror_Ent** and the name of the mirrored entity is returned. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) Manipulation of entities that are instances of type **fill_area_style_hatching** or **half_space_solid**, are only allowed for the usage within the TDB. For these kind of entities the manipulation will be performed, but the duplicated entity remain within in TDB (KFIX equals CAD, will be ignored).

2) If the given entity ENTNAM is an instance of type **api_group**, all entities that refer to this group shall be also mirrored.

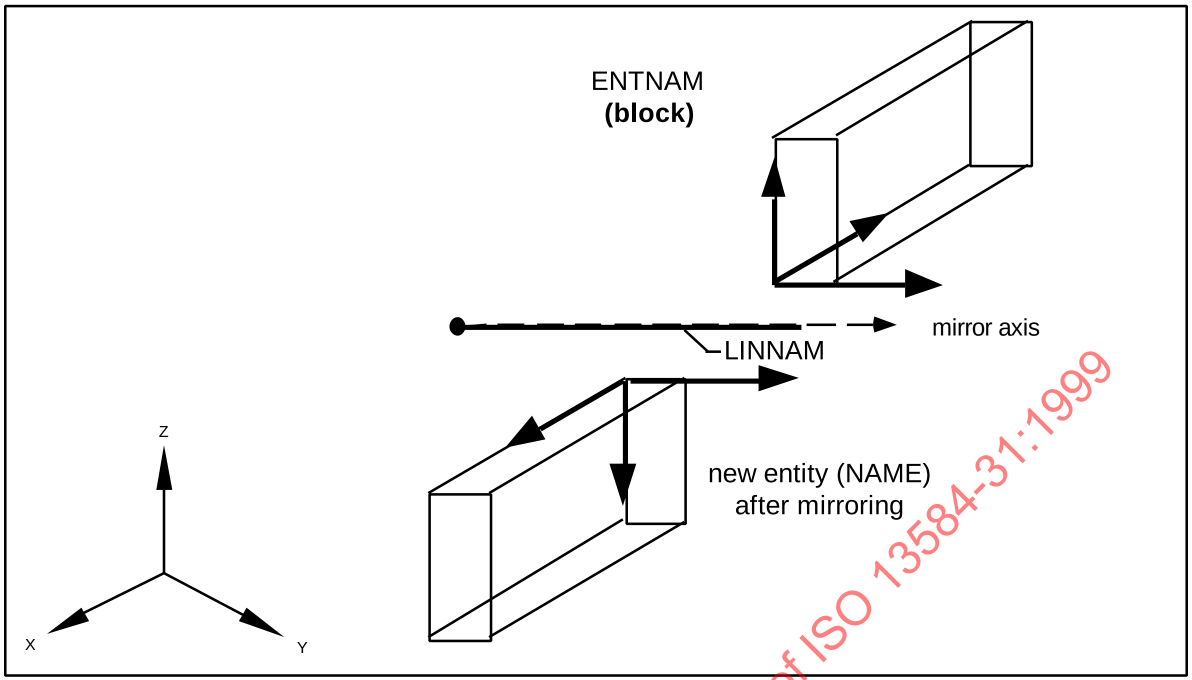


Figure A. 52 — Function: Dup_Mirror_Ent

Internal Reference:
6.1, 6.2.1, A7.1.1, A7.2.1

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	208	maximal number of groups exceeded
212	usage of entity only allowed for usage within the TDB	1001	enumerated value out of range

A.7.3 Shifting entities

Shift an entity defined by a direction	Shift_Dir_Ent
Shift an entity defined by displacements	Shift_Displacement_Ent
Duplicate and shift an entity defined by a direction	Dup_Shift_Dir_Ent
Duplicate and shift an entity defined by displacements	Dup_Shift_Displacement_Ent

A.7.3.1 Shift an entity defined by a direction

Function name:

Shift_Dir_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	ENTNAM	N	name of an entity	all graphical entities,grp
in	DIRNAM	N	name of **direction**	dir
in	SHFLEN	D	shift length	(0.0 OR (EPS ≤ SHFLEN ≤ MAX))

FORTRAN binding:

```
CALL SHIFT_DIR_ENT ( ENTNAM,DIRNAM,SHFLEN )
```

Effects:

Translates the specified entity ENTNAM. The translation is defined by the given **direction** DIRNAM, and length SHFLEN. The name of entity ENTNAM, all visual presentation (e.g. presentation style, view_level), group and set structure of the entity are not changed. The shift length SHFLEN is measured in *OVC_length_units* and shall either be equal to zero or in range [EPS,MAX]. If an error occurs, no modification will be done.

Notes:

1) If the given entity ENTNAM is an instance of type **api_group**, all entities that refer to this group shall be shifted.

Internal reference:

6.1, 6.2.1

Errors:

1	Entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	Value for length measure out of permitted range	204	function not compatible with current power level

A.7.3.2 Shift an entity defined by displacements

Function name:

Shift_Displacement_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	ENTNAM	N	name of an entity	all graphical entities,grp
in	DX	D	displacement in x-direction	(0.0 OR (EPS≤|DX|≤MAX))
in	DY	D	displacement in y-direction	(0.0 OR (EPS≤|DY|≤MAX))
in	DZ	D	displacement in z-direction	(0.0 OR (EPS≤|DZ|≤MAX))

FORTRAN binding:

```
CALL SHIFT_DISPLACEMENT_ENT ( ENTNAM,DX,DY,DZ )
```

Effects:

Moves an entity ENTNAM by a given displacement from its original position. The given values for the displacement are measured in *OVC_length_units* and they shall either be equal to zero or in range [EPS,MAX]. The name of entity ENTNAM, all visual presentation (e.g. presentation style, view_level), group and set structure of the entity are not changed. If an error occurs, no modification will be done.

Notes:

1) If the given entity ENTNAM is an instance of type **api_group**, all entities that refer to this group shall be shifted.

2) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the given parameter DZ will subsequently be ignored by the interface.

Internal reference:

6.1, 6.2.1

Errors:

1	Entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	Value for length measure out of permitted range	204	function not compatible with current power level

A.7.3.3 Duplicate and shift an entity defined by a direction

Function name:

Dup_Shift_Dir_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	Name	data type	meaning	permitted types/values
in	ENTNAM	N	name of an entity	all graphical entities,grp
in	DIRNAM	N	name of **direction**	dir
in	SHFLEN	D	shift length	(0.0 OR (EPS ≤ SHFLEN ≤ MAX))
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of duplicated and shifted entity	same type as ENTNAM

FORTRAN binding:

```
NAME = DUP_SHIFT_DIR_ENT ( ENTNAM,DIRNAM,SHFLEN,KFIX )
```

Effects:

This function invokes the function **Dup_Ent**. The returned entity is translated by the function **Shift_Dir_Ent** and the name of the translated entity is returned. The shift length SHFLEN is measured in *OVC_length_units* and shall either be equal to zero or in range [EPS,MAX]. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) If the given shift length is equal to zero, only a copy of entity ENTNAM shall be created by invoking the function **Dup_Ent**.

2) Manipulation of entities that are instances of type **fill_area_style_hatching** or **half_space_solid**, are only allowed for the usage within the TDB. For these kind of entities the manipulation will be performed, but the duplicated entity remain within in TDB (KFIX equals CAD, will be ignored).

3) If the given entity ENTNAM is an instance of type **api_group**, all entities that refer to this group shall be also shifted.

Internal Reference:

6.1, 6.2.1, A7.1.1, A7.3.1

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
3	value for length measure out of permitted range	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
208	maximal number of groups exceeded	212	usage of entity only allowed for usage within the TDB
1001	Enumerated value out of range		

A.7.3.4 Duplicate and shift an entity defined by displacements

Function name:

Dup_Shift_Displacement_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	meaning	permitted types/values
in	ENTNAM	N	name of an entity	all graphical entities,grp
in	DX	D	displacement in x-direction	(0.0 OR (EPS≤|DX|≤MAX))
in	DY	D	displacement in y-direction	(0.0 OR (EPS≤|DY|≤MAX))
in	DZ	D	displacement in z-direction	(0.0 OR (EPS≤|DZ|≤MAX))
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of duplicated and shifted entity	same type as ENTNAM

FORTRAN binding:

```
NAME = DUP_SHIFT_DISPLACEMENT_ENT ( ENTNAM,DX,DY,DZ,KFIX )
```

Effects:

This function invokes the function **Dup_Ent**. The returned entity is translated by the function **Shift_Displacement_Ent** and the name of the translated entity is returned. The given values for the displacement are measured in *OVC_length_units* and they shall either be equal to zero or in range [EPS,MAX]. If an error occurs, no entity is created and the element name is returned as zero.

Notes:

1) If the calculated Euclidean norm of displacement is equal to zero, only a copy of entity ENTNAM shall be created by invoking the function **Dup_Ent**.

2) Manipulation of entities that are instances of type **fill_area_style_hatching** or **half_space_solid**, are only allowed for the usage within the TDB. For these kind of entities the manipulation will be performed, but the duplicated entity remain within in TDB (KFIX equals CAD, will be ignored).

3) If the given entity ENTNAM is an instance of type **api_group**, all entities that refer to this group shall be also duplicated and shifted.

4) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the given parameter DZ will subsequently be ignored by the interface.

Internal Reference:

6.1, 6.2.1, A7.1.1, A7.3.1

Errors:

1	entity name not defined (zero, or unknown)	3	value for length measure out of permitted range
201	temporary database overflow	202	error while sending entity to CAD system
204	function not compatible with current power level	208	maximal number of groups exceeded
212	usage of entity only allowed for usage within the TDB	1001	enumerated value out of range

A.7.4 Rotating entities

Rotate an entity	Rotate_Ent
Duplicate and rotate an entity	Dup_Rotate_Ent

A.7.4.1 Rotate an entity

Function name:

Rotate_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	meaning	permitted types/values
in	ENTNAM	N	name of an entity	all graphical entities,grp
in	PNTNAM	N	name of reference **cartesian_point** for the rotation	pnt
in	ANG1	D	rotation angle in (Oxy) plane around (Oz) axis	(-360° ≤ ANG1 ≤ 360°)
in	ANG2	D	rotation angle in (Oyz) plane around (Ox) axis	(-360° ≤ ANG2 ≤ 360°)
in	ANG3	D	rotation angle in (Ozx) plane around (Oy) axis	(-360° ≤ ANG3 ≤ 360°)

FORTRAN binding:

```
CALL ROTATE_ENT ( ENTNAM,PNTNAM,ANG1,ANG2,ANG3 )
```

Effects:

Rotates an entity ENTNAM. The reference **cartesian_point** is the origin of the virtual coordinate system, used for the rotation. The virtual coordinate system is defined by translating the current reference coordinate system of the OVC onto the PNTNAM point. The rotation shall be done in the following order:

1st – rotation around (Oz) axis,

2nd – rotation around (Ox) axis,

3rd – rotation around (Oy) axis.

The given angles are measured in *OVC_angle_unit*s. The name of entity ENTNAM, all visual presentation (e.g. presentation style, view_level), group and set structure of the entity are not changed. If an error occurs, no modification will be done.

Notes:

1) If the calculated displacement between origin location of entity and rotated location of this entity is in range [ZERO_VALUE,EPS], no modification shall be performed and no error occurs.

2) If the given entity ENTNAM is an instance of type **api_group**, all entities that refer to this group shall be also rotated.

3) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the given parameters ANG2 and ANG3 for axis rotations will subsequently be ignored by the interface and only the rotation of x,y-plane around PNTNAM will be performed.

Internal Reference:
6.1, 6.2.1

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
4	value for plane angle measure out of permitted range	204	function not compatible with current power level

A.7.4.2 Duplicate and rotate an entity

Function name:
Dup_Rotate_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	meaning	permitted types/values
in	ENTNAM	N	name of an entity	all graphical entities,grp
in	PNTNAM	N	name of reference **cartesian_point** for the rotation	pnt
in	ANG1	D	rotation angle in (Oxy) plane around (Oz) axis	(-360° ≤ ANG1 ≤ 360°)
in	ANG2	D	rotation angle in (Oyz) plane around (Ox) axis	(-360° ≤ ANG2 ≤ 360°)
in	ANG3	D	rotation angle in (Ozx) plane around (Oy) axis	(-360° ≤ ANG3 ≤ 360°)
in	KFIX	E	storage location	[TDB,CAD]
out	NAME	N	name of duplicated and rotated entity	same type as ENTNAM

FORTRAN binding:

```
NAME = DUP_ROTATE_ENT ( ENTNAM,PNTNAM,ANG1,ANG2,ANG3,KFIX )
```

Effects:
This function invokes the function **Dup_Ent**. The returned entity is rotated by the function **Rotate_Ent** and the name of the translated entity is returned. If an error occurs, no entity is created and the element name is returned as zero.

Notes:
1) If the calculated displacement between origin location of entity and rotated location of this entity is in range [ZERO_VALUE,EPS], only a copy of entity ENTNAM shall be created by invoking the function **Dup_Ent**.

2) Manipulation of entities that are instances of type **fill_area_style_hatching** or **half_space_solid**, are only allowed for the usage within the TDB. For these kind of entities the manipulation will be performed, but the duplicated entity remain within in TDB (KFIX equals CAD, will be ignored).

3) If the given entity ENTNAM is an instance of type **api_group**, all entities that refer to this group shall be also duplicated and rotated.

4) When the current open view is defined as a 2D view (it means that the *geometrical_power_level* entry of the interface status table equals 1), the given parameters ANG2 and ANG3 for axis rotations will subsequently be ignored by the interface and only the duplication and a rotation of x,y-plane around PNTNAM will be performed.

Internal Reference:
6.1, 6.2.1, A7.1.1, A7.4.1

Errors:

1	entity name not defined (zero, or unknown)	2	entity type out of permitted range
4	value for plane angle measure out of permitted range	201	temporary database overflow
202	error while sending entity to CAD system	204	function not compatible with current power level
208	maximal number of groups exceeded	212	usage of entity only allowed for usage within the TDB
1001	Enumerated value out of range		

A.7.5 Changing entities

Change the orientation of a curves entity	Chg_Orientation_Ent
Change the sense of a circular or elliptical entity	Chg_Sense_Ent
Homotetia of an entity	Homotetia_Ent

A.7.5.1 Change the orientation of a curves entity

Function name:
Chg_Orientation_Ent

interface level:	**1**
geometrical power level:	**1,2,3**

Parameters:

in/ out	name	data type	meaning	permitted types/values
in	ENTNAM	N	name of an entity	curves

FORTRAN binding:

```
CALL CHG_ORIENTATION_ENT ( ENTNAM )
```

Effects:
Changes the orientation of an curves entity, ENTNAM. If the given entity is in range of basic or conic arc , the following procedure is executed:

— The two trimming points **trim_1** and **trim_2** interchanged.

— The **sense_agreement** flag is changed.

If the given curves entity ENTNAM is an instance of type **polyline,** the following procedure is executed:

— The order of the list of the **cartesian_point**s is reversed.

If the given curves entity ENTNAM is an instance of type **api_contour,** the following procedure is executed:

— Every item of the list of **composite_curve_segment** is changed by the procedure, that is specified above in this function.

— The order of the list of the **segments** is reversed.