Data Exchange

STEP Import/Export User's Guide

Version 6.2 / March 2007

NOTICE FOR USERS:

This User Guide is a general instruction for Open CASCADE study. It may be incomplete and even contain occasional mistakes, particularly in examples, samples, etc. Open CASCADE S.A.S. bears no responsibility for such mistakes. If you find any mistakes or imperfections in this document, or if you have suggestions for improving this document, please, contact us and contribute your share to the development of Open CASCADE Technology: bugmaster@opencascade.com

15 bis rue Ernest RENAN

92136 ISSY LES MOULINEAUX

FRANCE

Contents

1. Overview

This manual is intended to provide technical documentation on the Open CASCADE STEP processor and to help Open CASCADE users with the use of the STEP processor (to read and write STEP files). STEP files conforming to AP 214, AP 203 and partially AP 209 can be read. STEP files that are produced by this interface conform to STEP AP 214 or AP 203, according to the user option.

Only geometrical, topological STEP entities (shapes) and assembly structures are translated by the basic translator described in sections 2 to 6. Data that cannot be translated on this level are also loaded from a STEP file and can be translated later. XDE STEP translator (see section 7 “Reading from and writing to XDE”) translates names, colors, layers, validation properties and other data associated with shapes and assemblies into XDE document.

File translation is performed in the programming mode, via C++ calls.

For testing the STEP component in DRAW Test Harness, a set of commands for reading and writing STEP files and analysis of relevant data are provided by the TKXSDRAW plugin.

2. Reading STEP

2.1. Procedure

You can translate a STEP file into an Open CASCADE shape in the following steps:

1. load the file,
2. check file consistency,
3. set the translation parameters,
4. perform the translation,
5. fetch the results.

2.2. Domain covered

2.2.1. Assemblies

The "ProSTEP Round Table Agreement Log" (version July 1998), item 21, defines two alternatives for the implementation of assembly structure representations: using mapped_item entities and using representation_relationship_with_transformation entities. Both these alternative representations are recognized and processed at reading. On writing, the second alternative is always employed.

Handling of assemblies is implemented in two separate levels: firstly STEP assembly structures are translated into Open CASCADE shapes, and secondly the Open CASCADE shape representing the assembly is converted into any data structure intended for representing assemblies (for example, OCAF).

The first part of this document describes the basic STEP translator implementing translation of the first level, i.e. translation to Open CASCADE Shapes. On this level, the acyclic graph representing the assembly structure in a STEP file is mapped into the structure of nested TopoDS_Compounds in Open CASCADE. The (sub)assemblies become (sub)compounds containing shapes which are the results of translating components of that (sub)assembly. The sharing of components of assemblies is preserved as Open CASCADE sharing of subshapes in compounds.

The attributive information attached to assembly components in a STEP file (such as names and descriptions of products, colors, layers etc.) can be translatd after the translation of the shape itself by parsing the STEP model (loaded in memory). Several tools from the package STEPConstruct provide functionalities to read styles (colors), validation properties, product information etc.

Implementation of the second level of translation (conversion to XDE data structure) is provided by XDE STEP translator and is described in section 7.

2.2.2. Shape representations

Length units, plane angle units and the uncertainty value are taken from shape_representation entities. This data is used in the translation process.

The types of STEP representation entities that are recognized are:

- advanced_brep_shape_representation
- faceted_brep_shape_representation
- manifold_surface_shape_representation

- geometrically_bounded_wireframe_shape_representation
- geometrically_bounded_surface_shape_representation
- hybrid representations (shape_representation containing models of different type)

2.2.3. Topological entities

The types of STEP topological entities that can be translated are:

- vertices
- edges
- loops
- faces
- shells
- solids

For further information see 2.4 Mapping STEP entities to Open CASCADE shapes.

2.2.4. Geometrical entities

The types of STEP geometrical entities that can be translated are:

- points
- vectors
- directions
- curves
- surfaces

For further information see 2.4 Mapping STEP entities to Open CASCADE shapes.

2.3. Description of the process

2.3.1. Loading the STEP file

Before performing any other operation you have to load the file with:

```
STEPControl_Reader reader;
IFSelect_ReturnStatus stat = reader.ReadFile("filename.stp");
```

Loading the file only memorizes the data, it does not translate it.

2.3.2. Checking the STEP file

This step is not obligatory. Check the loaded file with:

```
reader.PrintCheckLoad(failsonly,mode);
```

Error messages are displayed if there are invalid or incomplete STEP entities, giving you the information on the cause of error.

Only fail messages are displayed if failsonly is true. All messages are displayed if failsonly is false. Your analysis of the file can be either message-oriented or entity-oriented. Choose your preference with:

```
IFSelect_PrintCount mode = IFSelect_xxx
```

Where xxx can be one of the following:

`ItemsByEntity` - gives a sequential list of all messages per STEP entity,

`CountByItem` - gives the number of STEP entities with their types per message

`ListByItem` - gives the number of STEP entities with their types and rank numbers per message

2.3.3. Setting the translation parameters

The following parameters can be used to translate a STEP file into an Open CASCADE shape.

If you give a value that is not within the range of possible values it will simply be ignored.

read.precision.mode

Defines which precision value will be used during translation (see section 2.5 below for details on precision and tolerances).

"File" (0): the precision value is set to length_measure in uncertainty_measure_with_unit from STEP file.

"User" (1): the precision value is that of the read.precision.val parameter.

Read this parameter with:

```
Standard_Integer ic = Interface_Static::IVal("read.precision.mode");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("read.precision.mode",1))
.. error ..
```

Default value is "File" (0).

read.precision.val:

User defined precision value. This parameter gives the precision for shape construction when the read.precision.mode parameter value is 1.

0.0001: default

any real positive (non null) value.

This value is a basic value of tolerance in the processor. The value is in millimeters, independently of the length unit defined in the STEP file.

Read this parameter with:

```
Standard_Real rp = Interface_Static::RVal("read.precision.val");
```

Modify this parameter with:

```
if(!Interface_Static::SetRVal("read.precision.val",0.01))
.. error ..
```

Default value is 0.0001.

NOTE

The value given to this parameter is a basic value for ShapeHealing algorithms and the processor. It does its best to reach it. Under certain circumstances, the value you give may not be attached to all of the entities concerned at the end of processing. STEP-to-OpenCASCADE translation does not improve the quality of the geometry in the original STEP file. This means that the value you enter may be impossible to attach to all shapes with the given quality of the geometry in the STEP file.

read.maxprecision.val

Defines the maximum allowed tolerance (in mm) of the shape. It should be not less than the basic value of tolerance set in the processor (either the uncertainty from the file or read.precision.val). Actually, the maximum between read.maxprecision.val and the basis tolerance is used to define the maximum allowed tolerance.

Read this parameter with:

```
Standard_Real rp = Interface_Static::RVal("read.maxprecision.val");
```

Modify this parameter with:

```
if(!Interface_Static::SetRVal("read.maxprecision.val",0.1))
.. error ..
```

Default value is 1.

Note that maximum tolerance even explicitly defined by the user may be insufficient to ensure the validity of the shape (if real geometry is of bad quality). Therefore the user is provided with an additional parameter, which allows him to choose: either he prefers to ensure the shape validity or he rigidly sets the value of maximum tolerance. In the first case there is a possibility that the tolerance will not have any upper limit, in the second case the shape may be invalid.

read.maxprecision.mode:

Defines the mode of applying the maximum allowed tolerance. Its possible values are:

0 ("Preferred") - maximum tolerance is used as a limit but sometimes it can be exceeded (currently, only for deviation of a 3D curve and pcurves of an edge, and vertices of such edge) to ensure the shape validity,

1 ("Forced") - maximum tolerance is used as a rigid limit, i.e. no tolerance can exceed it and if it is the case, the tolerance is trimmed by the maximum tolerance.

Read this parameter with:

```
Standard_Integer ic = Interface_Static::IVal("read.maxprecision.mode");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("read.maxprecision.mode",1))
.. error ..
```

Default value is 0 ("Preferred").

read.stdsameparameter.mode

defines the use of BRepLib::SameParameter. Its possible values are:

0 ("Off") - BRepLib::SameParameter is not called,

1 ("On") - BRepLib::SameParameter is called.

The functionality of BRepLib::SameParameter is used through ShapeFix_Edge::SameParameter. It ensures that the resulting edge will have the lowest tolerance taking pcurves either unmodified from the STEP file or modified by BRepLib::SameParameter.

Read this parameter with:

```
Standard_Integer mv =
Interface_Static::IVal("read.stdsameparameter.mode");
```

Modify this parameter with:

```
if (!Interface_Static::SetIVal ("read.stdsameparameter.mode",1))
.. error ..;
```

Deafault value is 0 ("Off").

read.surfacecurve.mode:

a preference for the computation of curves in an entity which has both 2D and 3D representation.

Each TopoDS_Edge in TopoDS_Face must have a 3D and 2D curve that references the surface.

If both 2D and 3D representation of the entity are present, the computation of these curves depends on the following values of parameter:

"Default" (0): no preference, both curves are taken (default value),

"3DUse_Preferred" (3): 3D curves are used to rebuild 2D ones.

Read this parameter with:

```
Standard_Integer rp = Interface_Static::IVal("read.surfacecurve.mode");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("read.surfacecurve.mode",3))
.. error ..
```

Default value is (0).

read.encoderegularity.angle

This parameter is used for call to BRepLib::EncodeRegularity() function which is called for the shape read from an IGES or a STEP file at the end of translation process. This function sets the regularity flag of the edge in the shell when this edge is shared by two faces. This flag shows the continuity these two faces are connected with at that edge.

Read this parameter with:

```
Standard_Real era =
Interface_Static::RVal("read.encoderegularity.angle");
```

Modify this parameter with:

```
if (!Interface_Static::SetRVal ("read.encoderegularity.angle",0.1))
.. error ..;
```

Default value is 0.01.

step.angleunit.mode

This parameter is obsolete (it was required in the past for STEP files with a badly encoded angle unit). It indicates what angle units should be used when a STEP file is read: the units from file (default), or forced RADIANS or DEGREES.

Default value is File

read.step.resource.name

read.step.sequence

These two parameters define the name of the resource file and the name of the sequence of operators (defined in that file) for Shape Processing, which is automatically performed by the STEP translator. Shape Processing is a user-configurable step, which is performed after translation and consists in

applying a set of operators to a resulting shape. This is a very powerful tool allowing customizing the shape and adapting it to the needs of a receiving application. By default the sequence consists of a single operator ShapeFix - that is how Shape Healing is called from the STEP translator.

Please find an example of the resource file for STEP (which defines parameters corresponding to the sequence applied by default, i.e. if the resource file is not found) in the Open CASCADE installation, by the path %CASROOT%/src/XSTEPResource/STEP ($CASROOT/src/XSTEPResource/STEP).

In order for the STEP translator to use that file, you have to define the CSF_STEPDefaults environment variable, which should point to the directory where the resource file resides. Note that if you change parameter read.step.resource.name, you will change the name of the resource file and the environment variable correspondingly.

Default values: read.step.resource.name - STEP, read.step.sequence - FromSTEP.

read.scale.unit

This parameter is obsolete (the parameter xstep.cascade.unit should be used instead when necessary). If it is set to 'M', the shape is scaled 0.001 times (as if it were in meters) after translation from IGES or STEP.

Default value is MM.

xstep.cascade.unit

This parameter defines units to which a shape should be converted when translated from IGES or STEP to CASCADE. Normally it is MM; only those applications that work internally in units other than MM should use this parameter.

Default value is MM.

read.step.product.mode:

Defines the approach used for selection of top-level STEP entities for translation, and for recognition of assembly structures

1 ("ON") - PRODUCT_DEFINITION entities are taken as top-level ones; assembly structure is recognized by NEXT_ASSEMBLY_USAGE_OCCURRENCE entities. This is regular mode for reading valid STEP files conforming to AP 214, AP203 or AP 209.

0 ("OFF") - SHAPE_DEFINITION_REPRESENTATION entities are taken as top-level ones; assembly is recognized by CONTEXT_DEPENDENT_SHAPE_REPRESENTATION entities. This is compatibility mode, which can be used for reading legacy STEP files produced by older versions of STEP translators and having incorrect or incomplete product information.

Read this parameter with:

```
Standard_Integer ic = Interface_Static::IVal("read.step.product.mode");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("read.step.product.mode",1))
.. error ..
```

Default value is 1 ("ON").

Note that the following parameters have effect only if read.step.product.mode is ON.

read.step.product.context:

When reading AP 209 STEP files, allows selecting either only ‘design’ or ‘analysis’, or both types of products for translation

1 ("all") - translate all products

2 ("design") - translate only products that have PRODUCT_DEFINITION_CONTEXT with field life_cycle_stage set to ‘design’

3 (“analysis”) - translate only products associated with PRODUCT_DEFINITION_CONTEXT entity whose field life_cycle_stage set to ‘analysis’

Note that in AP 203 and AP214 files all products should be marked as ‘design’, so if this mode is set to ‘analysis’, nothing will be read.

Read this parameter with:

```
Standard_Integer ic =
                 Interface_Static::IVal("read.step.product.context");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("read.step.product.context",1))
.. error ..
```

Default value is 1 ("all").

read.step.shape.repr:

Specifies preferred type of representation of the shape of the product, in case if a STEP file contains more than one representation (i.e. multiple PRODUCT_DEFINITION_SHAPE entities) for a single product

1 ("All") - Translate all representations (if more than one, put in compound).

2 ("ABSR") - Prefer ADVANCED_BREP_SHAPE_REPRESENTATION

3 ("MSSR") - Prefer MANIFOLD_SURFACE_SHAPE_REPRESENTATION

4 ("GBSSR") - Prefer GEOMETRICALLY_BOUNDED_SURFACE_SHAPE_REPRESENTATION

5 ("FBSR") - Prefer FACETTED_BREP_SHAPE_REPRESENTATION

6 ("EBWSR") - Prefer EDGE_BASED_WIREFRAME_SHAPE_REPRESENTATION

7 ("GBWSR") - Prefer GEOMETRICALLY_BOUNDED_WIREFRAME _SHAPE_REPRESENTATION

When this option is not equal to 1, for products with multiple representations the representation having a type closest to the selected one in this list will be translated.

Read this parameter with:

```
Standard_Integer ic = Interface_Static::IVal("read.step.shape.repr");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("read.step.shape.repr",1))
.. error ..
```

Default value is 1 ("All").

read.step.assembly.level:

Specifies which data should be read for the products found in the STEP file:

1 ("All") - Translate both the assembly structure and all associated shapes. If both shape and sub-assemblies are associated with the same product, all of them are read and put in a single compound.

Note that this situation is confusing, as semantics of such configuration is not defined clearly by the STEP standard (whether this shape is an alternative representation of the assembly or is an addition to it), therefore warning will be issued in such case.

2 ("assembly") - Translate the assembly structure and shapes associated with parts only (not with sub-assemblies).

3 ("structure") - Translate only the assembly structure without shapes (a structure of empty compounds). This mode can be useful as an intermediate step in applications requiring specialized processing of assembly parts.

4 ("shape") - Translate only shapes associated with the product, ignoring the assembly structure (if any). This can be useful to translate only a shape associated with specific product, as a complement to “assembly” mode.

Read this parameter with:

```
Standard_Integer ic =
                 Interface_Static::IVal("read.step.assembly.level");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("read.step.assembly.level ",1))
.. error ..
```

Default value is 1 ("All").

read.step.shape.relationship:

Defines whether shapes associated with the main SHAPE_DEFINITION_REPRESENTATION entity of the product via SHAPE_REPRESENTATIONSHIP_RELATION should be translated. This kind of association is used for the representation of hybrid models (i.e. models whose shape is composed of different types of representations) in AP 203 files since 1998, but it can be also used to associate auxiliary data with the product. This parameter allows to avoid translation of such auxiliary data.

1 ("ON") - translate

0 ("OFF") - do not translate

Read this parameter with:

```
Standard_Integer ic =
          Interface_Static::IVal("read.step.shape.relationship");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("read.step.shape.relationship",1))
.. error ..
```

Default value is 1 ("ON").

read.step.shape.aspect:

Defines whether shapes associated with the PRODUCT_DEFINITION_SHAPE entity of the product via SHAPE_ASPECT should be translated. This kind of association was used for the representation of hybrid models (i.e. models whose shape is composed of different types of representations) in AP 203 files before 1998, but it is also used to associate auxiliary information with the sub-shapes of the part. Though STEP translator tries to recognize such cases correctly, this parameter may be useful to avoid unconditionally translation of shapes associated via SHAPE_ASPECT entities.

1 ("ON") - translate

0 ("OFF") - do not translate

Read this parameter with:

```
Standard_Integer ic =
                 Interface_Static::IVal("read.step.shape.aspect");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("read.step.shape.aspect",1))
.. error ..
```

Default value is 1 ("ON").

2.3.4. Performing the STEP file translation

Perform the translation according to what you want to translate. You can choose either root entities (all or selected by the number of root), or select any entity by its number in the STEP file. There is a limited set of types of entities that can be used as starting entities for translation. Only the following entities are recognized as transferable:

- product_definition
- next_assembly_usage_occurrence
- shape_definition_representation
- subtypes of shape_representation (only if referred representation is transferable)
- manifold_solid_brep
- brep_with_voids
- faceted_brep
- faceted_brep_and_brep_with_voids
- shell_based_surface_model
- geometric_set and geometric_curve_set
- mapped_item
- subtypes of face_surface (including advanced_face)
- subtypes of shape_representation_relationship
- context_dependent_shape_representation

The following methods are used for translation:

- Translate a root entity identified by its rank with:

```
Standard_Boolean ok = reader.TransferRoot(rank);
```

- Translate an entity identified by its rank with:

```
Standard_Boolean ok = reader.TransferOne(rank);
```

- Translate a list of entities in one operation with (this method returns the number of successful translations):

```
Standard_Integer num = reader.TransferList(list);
```

- Translate all transferable roots with:

```
Standard_Integer NbRoots = reader.NbRootsForTransfer();
Standard_Integer num = reader.TransferRoots();
```

2.3.5. Getting the translation results

Each successful translation operation outputs one shape. A series of translations gives a set of shapes.

Each time you invoke TransferOne(), TransferRoot() or TransferList(), their results are accumulated and the counter of results increases. You can clear the results with:

```
reader.ClearShapes();
```

between two translation operations, if you do not, the results from the next translation will be added to the accumulation.

TransferRoots() operations automatically clear all existing results before they start.

- Get the number of shapes recorded in the result with:

```
Standard_Integer num = reader.NbShapes();
```

- Get the result identified by its rank, where rank is an integer between 1 and NbShapes, with:

```
TopoDS_Shape shape = reader.Shape(rank);
```

- Get the first result of translation with:

```
TopoDS_Shape shape = reader.Shape();
```

- Get all of results in a single shape which is:

  a null shape if there are no results,

  in case of a single result, a shape that is specific to that result,

  a compound that lists the results if there are several results.

```
TopoDS_Shape shape = reader.OneShape();
```

Clearing the accumulation of results

If several individual translations follow each other, the results give a list that can be purged with:

```
reader.ClearShapes();
```

which erases the existing results.

Checking that translation was correctly performed

Each time you invoke Transfer… or TransferRoots(), you can display the related messages with the help of:

```
reader.PrintCheckTransfer(failsonly,mode);
```

This check concerns the last invocation of Transfer… or TransferRoots() only.

2.3.6. Selecting STEP entities for translation

Selection possibilities

There are three selection possibilities. You can select:

- the whole file,
- a list of entities,
- one entity.

Whole file

Transferring the whole file means transferring all root entities. The number of roots can be evaluated when the file is loaded:

```
Standard_Integer NbRoots = reader.NbRootsForTransfer();
Standard_Integer num = reader.TransferRoots();
```

Lists of entities

A list of entities can be formed by invoking STEP214Control_Reader::GiveList (this is a method of the parent class).

Here is a simple example of how a list is translated:

```
Handle(TColStd_HSequenceOfTransient) list = reader.GiveList();
```

The result is a TColStd_HSequenceOfTransient.

You can either translate a list entity by entity or all at once. An entity-by-entity operation lets you check each individual entity translated.

Translating a whole list in one operation

```
Standard_Integer nbtrans = reader.TransferList (list);
```

nbtrans gives the number of items in the list that produced a shape.

Translating a list entity by entity:

```
Standard_Integer i,nb = list->Length();
for (i = 1; i <= nb; i ++) {
 Handle(Standard_Transient) ent = list->Value(i);
 Standard_Boolean OK = reader.TransferEntity (ent);
}
```

Selections

There is a number of predefined operators that can be used. They are:

- step214-placed-items

Selects all mapped_items or context_depended_shape_representations.

- step214-shape-def-repr

Selects all shape_definition_representations.

- step214-shape-repr

Selects all shape_representations.

- step214-type(<entity_type>)

Selects all entities of a given type

- step214-faces

Selects all faces_surface, advanced_face entities and the surface entity or any sub type if these entities are not shared by any face entity or shared by geometric_set entity.

- step214-derived(<entity_type>)

Selects entities of a given type or any subtype.

- step214-GS-curves

Selects all curve entities or any subtype except the composite_curve if these entities are shared by the geometric_set entity.

- step214-assembly

Selects all mapped_items or context_depended_shape_representations involved into the assembly structure.

- xst-model-all

Selects all entities.

- xst-model-roots

Selects all roots.

- xst-shared + <selection>

Selects all entities shared by at least one entity selected by <selection>.

- xst-sharing + <selection>

Selects all entities sharing at least one entity selected by <selection>.

- xst-transferrable-all

Selects all transferable entities.

- xst-transferrable-roots

Selects all translatable roots.

Cumulative lists can be used too.

Single entities

You can select an entity either by its rank or by its handle (an entity's handle can be obtained by invoking the StepData_StepModel::Entity function).

Selection by rank

Use method StepData_StepModel::NextNumberForLabel to find its rank with the following:

```
Standard_CString label = '#...';
StepData_StepModel model = reader.StepModel();
rank = model->NextNumberForLabe(label, 0, Standard_False);
```

Translate an entity specified by its rank:

```
Standard_Boolean ok = reader.Transfer (rank);
```

Direct selection of an entity

<ent> is the entity. The argument is a Handle(Standard_Transient).

```
Standard_Boolean ok = reader.TransferEntity (ent);
```

2.4. Mapping STEP entities to Open CASCADE shapes

Tables given in this paragraph show the mapping of STEP entities to Open CASCADE objects. Only topological and geometrical STEP entities and entities defining assembly structures are described in this paragraph. For a full list of STEP entities please refer to Appendix A.

2.4.1. Assembly structure representation entities

Not all entities defining the assembly structure in the STEP file are translated to Open CASCADE shapes, but they are used to identify the relationships between assemblies and their components. Since the graph of 'natural' dependencies of entities based on direct references between them does not include the references from assemblies to their components, these dependencies are introduced in addition to the former ones. This is made basing on the analysis of the following entities describing the structure of the assembly.

STEP entity type	CASCADE shape	Comments
product_definition	For assemblies, a TopoDS_Compound, for components, a CASCADE shape corresponding to the type of component	Each assembly or component has its own product_definition. It is used as a starting point for translation when read.step.product.mode is ON
product_definition_shape		This entity provides a link between product_definition and corresponding shape_definition_representation, or between next_assembly_usage_occurence and corresponding context_dependent_shape_representation
shape_definition_representation	For assemblies, a TopoDS_Compound, for components, a CASCADE shape corresponding to the type of component	Each assembly or component has its own shape_definition_representation. The graph of dependencies is modified in such a way that shape_definition_representations of all components of the assembly are referred by the shape_definition_representation of the assembly.
next_assembly_usage_occurence		This entity defines a relationship between the assembly and its component. It is used to introduce (in the dependencies graph) the links between shape_definition_representation of the assembly and shape_definition_representations and context_dependent_shape_representations of all its components.
mapped_item	TopoDS_Shape	This entity defines a mapping of the assembly component into the shape_representation of the assembly. The result of translation is a CASCADE shape translated from the component, to which transformation defined by the mapped_item is applied.
context_dependent_shape_representation	TopoDS_Shape	This entity is associated with the next_assembly_usage_occurence entity and defines a placement of the component in the assembly. The graph of dependencies is modified so that each context_dependent_shape_representation is referred by shape_definition_representation of the corresponding assembly.
shape_representation_relationship_with_transformation		This entity is associated with context_dependent_shape_representation and defines a transformation necessary to apply to the component in order to locate it in its place in the assembly.

item_defined_transformation		This entity defines a transformation operator used by shape_representation_relationship_with_transformation or mapped_item entity
cartesian_transformation_operator		This entity defines a transformation operator used by shape_representation_relationship_with_transformation or mapped_item entity

2.4.2. Models

STEP entity type	CASCADE shape	Comments
Solid Models		
brep_with_voids	TopoDS_Solid	
faceted_brep	TopoDS_Solid	
manifold_solid_brep	TopoDS_Solid	
Surface Models		
shell_based_surface_model	TopoDS_Compound	shell_based_surface_model is translated into one or more TopoDS_Shell grouped in a TopoDS_Compound
geometric_set	TopoDS_Compound	TopoDS_Compound contains only TopoDS_Faces, TopoDS_Wires, TopoDS_Edges and/or TopoDS_Vertices
Wireframe Models		
geometric_curve_set	TopoDS_Compound	TopoDS_Compound contains only TopoDS_Wires, TopoDS_Edges and/or TopoDS_Vertices

2.4.3. Topological entities

STEP entity type	CASCADE shape	Comments
Vertices		
vertex_point	TopoDS_Vertex	
Edges		
oriented_edge	TopoDS_Edge	
edge_curve	TopoDS_Edge	
Loops		
face_bound	TopoDS_Wire	
face_outer_bound	TopoDS_Wire	

edge_loop	TopoDS_Wire	
poly_loop	TopoDS_Wire	Each segment of poly_loop is translated into TopoDS_Edge with support of Geom_Line
vertex_loop	TopoDS_Wire	Resulting TopoDS_Wire contains only one degenerated TopoDS_Edge
Faces		
face_surface	TopoDS_Face	
advanced_face	TopoDS_Face	
Shells		
connected_face_set	TopoDS_Shell	
oriented_closed_shell	TopoDS_Shell	
closed_shell	TopoDS_Shell	
open_shell	TopoDS_Shell	

2.4.4. Geometrical entities

3D STEP entities are translated into geometrical objects from the Geom package while 2D entities are translated into objects from the Geom2d package.

STEP entity type	CASCADE object	Comments
Points		
cartesian_point	Geom_CartesianPoint	
	Geom2d_CartesianPoint	
Directions		
direction	Geom_Direction	
	Geom2d_Direction	
Vectors		
vector	Geom_VectorWithMagnitude	
	Geom2d_VectorWithMagnitude	
Placements		
axis1_placement	Geom_Axis1Placement	
axis2_placement_2d	Geom2d_AxisPlacement	
axis2_placement_3d	Geom_Axis2Placement	
Curves		
circle	Geom_Circle	
	Geom2d_Circle	
	Geom2d_BsplineCurve	Circle is translated into

		Geom2d_BSplineCurve when it references the surface of revolution (spherical surface, conical surface, etc.)
ellipse	Geom_Ellipse	
	Geom2d_Ellipse	
	Geom2d_BsplineCurve	Ellipse is translated into Geom2d_BSplineCurve when it references the surface of revolution (spherical surface, conical surface, etc.)
hyperbola	Geom_Hyperbola	
	Geom2d_Hyperbola	
line	Geom_Line	
	Geom2d_Line	
parabola	Geom_Parabola	
	Geom2d_Parabola	
pcurve	Geom2d_Curve	Pcurve in edge
curve_replica	Geom_Curve	Depending on the type of basis curve
	Geom2d_Curve	
offset_curve_3d	Geom_OffsetCurve	
trimmed_curve	Geom_TrimmedCurve	
	Geom2d_BsplineCurve	Only trimmed_curves trimmed by parameters are translated. All trimmed_curves are converted to Geom2d_BSplineCurve.
b_spline_curve	Geom_BsplineCurve	
	Geom2d_BsplineCurve	
b_spline_curve_with_ knots	Geom_BsplineCurve	
	Geom2d_BsplineCurve	
bezier_curve	Geom_BsplineCurve	
	Geom2d_BsplineCurve	
rational_b_spline_curve	Geom_BsplineCurve	
	Geom2d_BsplineCurve	
uniform_curve	Geom_BsplineCurve	
	Geom2d_BsplineCurve	
quasi_ uniform_curve	Geom_BsplineCurve	
	Geom2d_BsplineCurve	

surface_curve	TopoDS_Edge	surface_curve defines geometrical support of an edge and its pcurves.
seam_curve	TopoDS_Edge	the same as surface_curve
composite_curve_segment	TopoDS_Edge	as a segment of composite_curve
composite_curve	TopoDS_Wire	
composite_curve_on_surface	TopoDS_Wire	
boundary_curve	TopoDS_Wire	
Surfaces		
b_spline_surface	Geom_BsplineSurface	
b_spline_surface_with_knots	Geom_BsplineSurface	
bezier_surface	Geom_BSplineSurface	
conical_surface	Geom_ConicalSurface	
cylindrical_surface	Geom_CylindricalSurface	
offset_surface	Geom_OffsetSurface	
surface_replica	Geom_Surface	Depending on the type of basis surface
plane	Geom_Plane	
rational_b_spline_surface	Geom_BSplineSurface	
rectangular_trimmed_ surface	Geom_RectangularTrimmedSurface	
spherical_surface	Geom_SphericalSurface	
surface_of_linear_extrusion	Geom_SurfaceOfLinearExtrusion	
surface_of_revolution	Geom_SurfaceOfRevolution	
toroidal_surface	Geom_ToroidalSurface	
degenerate_toroidal_surface	Geom_ToroidalSurface	
uniform_surface	Geom_BSplineSurface	
quasi_uniform_surface	Geom_BSplineSurface	
rectangular_composite_surface	TopoDS_Compound	Contains TopoDS_Faces
curve_bounded_surface	TopoDS_Face	

2.5. Tolerance management

2.5.1. Values used for tolerances during reading STEP

During the STEP => Open CASCADE translation several parameters are used as tolerances and precisions for different algorithms. Some of them are computed from other tolerances using specific functions.

3D (spatial) tolerance

Package method Precision::Confusion()

Value is 10^{-7}. It is used as the minimal distance between points, which are considered to be distinct.

Uncertainty in STEP file

This parameter is attached to each shape_representation entity in a STEP file and defined as length_measure in uncertainty_measure_with_unit. It is used as a fundamental value of precision during translation.

User - defined variable read.precision.val

It is used instead of uncertainty from a STEP file when parameter read.precision.mode is 1 (“User”).

2D (parametric) tolerances

Package method Precision::PConfusion()

This is a value of 0.01* Precision::Confusion(). It is used to compare parametric bounds of curves.

Methods UResolution, VResolution (tolerance3d) of the class GeomAdaptor_Surface or BRepAdaptor_Surface

Return tolerance in parametric space of a surface computed from 3d tolerance.

NOTE

When one tolerance value is to be used for both U and V parametric directions, the maximum or the minimum value of UResolution and VResolution is used.

Methods Resolution (tolerance3d) of the class GeomAdaptor_Curve or BRepAdaptor_Curve

Returns tolerance in parametric space of a curve computed from 3d tolerance.

2.5.2. Initial setting of tolerances in translating objects

In the STEP processor, the basic value of tolerance is set in method STEPControl_ActorRead::Transfer() to either value of uncertainty in shape_representation in STEP file (if parameter read.precision.mode is 0), or to a value of parameter read.precision.val (if read.precision.mode is 1 or if the uncertainty is not attached to the current entity in the STEP file).

Translation starts from one entity translated as a root. Function which performs the translation (STEPControl_ActorRead::Transfer()) creates an object of the type StepToTopoDS_Builder, which is intended to translate topology.

This object gets the initial tolerance value that is equal to read.precision.val or the uncertainty from shape_representation. During the translation of the entity, new objects of types StepToTopoDS_Translate... are created for translating sub-entities. All of them use the same tolerances as a StepToTopoDS_Builder object.

2.5.3. Transfer process

Evolution of shape tolerances during transfer

Let us follow the evolution of tolerances during the translation of STEP entities into an Open CASCADE shape.

If the starting STEP entity is a geometric_curve_set all the edges and vertices are constructed with Precision::Confusion().

If the starting STEP entity is not a geometric_curve_set the sub-shapes of the resulting shape have the following tolerance:

- all the faces are constructed with Precision::Confusion(),
- edges are constructed with Precision::Confusion(). It can be modified later by:

  1.ShapeFix::SameParameter() - the tolerance of edge shows real deviation of the 3D curve and pcurves.

  2.ShapeFix_Wire::FixSelfIntersection() in case if a pcurve of a self-intersecting edge is modified.
- vertices are constructed with Precision::Confusion(). It can be modified later by:

  1.StepToTopoDS_TranslateEdge

  2.ShapeFix::SameParameter()

  3.ShapeFix_Wire::FixSelfIntersection()

  4.ShapeFix_Wire::FixLacking()

  5.ShapeFix_Wire::Connected()

So, the final tolerance of sub-shapes shows the real local geometry of shapes (distance between vertices of adjacent edges, deviation of a 3D curve of an edge and its parametric curves and so on) and may be less or greater than the basic value of tolerance in the STEP processor.

Translating into Geometry

Geometrical entities are translated by classes StepToGeom_Make... Methods of these classes translate STEP geometrical entities into Open CASCADE geometrical objects. Since these objects are not BRep objects, they do not have tolerances. Tolerance is used only as precision for detecting bad cases (such as points coincidence).

Translating into Topology

STEP topological entities are translated into Open CASCADE shapes by use of the following classes (package StepToTopoDS):

- StepToTopoDS_TranslateVertex
- StepToTopoDS_TranslateEdge
- StepToTopoDS_TranslateVertexLoop
- StepToTopoDS_TranslateEdgeLoop
- StepToTopoDS_TranslatePolyLoop
- StepToTopoDS_TranslateFace
- StepToTopoDS_TranslateShell
- StepToTopoDS_TranslateCompositeCurve
- StepToTopoDS_TranslateCurveBoundedSurface
- StepToTopoDS_Builder

- StepToTopoDS_MakeTransformed

Although in a STEP file the uncertainty value is assigned to shape_representation entities and this value is applied to all entities in this shape_representation, Open CASCADE shapes are produced with different tolerances. As a rule, updating the tolerance is fulfilled according to the local geometry of shapes (distance between vertices of adjacent edges, deviation of edge's 3D curve and its parametric curves and so on) and may be either less or greater than the uncertainty value assigned to the entity.

The following paragraphs show what default tolerances are used when creating shapes and how they are updated during translation.

Class StepToTopoDS_TranslateVertex

TopoDS_Vertex is constructed from a STEP vertex_point entity with Precision::Confusion().

Class StepToTopoDS_TranslateVertexLoop

Degenerated TopoDS_Edge in TopoDS_Wire is created with tolerance Precision::Confusion(). TopoDS_Vertex of a degenerated edge is constructed with the initial value of tolerance.

Class StepToTopoDS_TranslateEdge

TopoDS_Edge is constructed only on the basis of 3D curve with Precision::Confusion().. Tolerance of the vertices can be increased up to a distance between their positions and ends of 3D curve.

Class StepToTopoDS_TranslateEdgeLoop

TopoDS_Edges in TopoDS_Wire are constructed with the help of class StepToTopoDS_TranslateEdge. Pcurves from a STEP file are translated if they are present and read.surfacecurve.mode is 0. For each edge method ShapeFix_Edge::FixSameParameter() is called. If the resulting tolerance of the edge is greater than the maximum value between 1.0 and 2*Value of basis precision, then the pcurve is recomputed. The best of the original and the recomputed pcurve is put into TopoDS_Edge. The resulting tolerance of TopoDS_Edge is a maximal deviation of its 3D curve and its pcurve(s).

Class StepToTopoDS_TranslatePolyLoop

TopoDS_Edges in TopoDS_Wire are constructed with the help of class StepToTopoDS_TranslateEdge. Their tolerances are not modified inside this method.

Class StepToTopoDS_TranslateFace

TopoDS_Face is constructed with the initial value of tolerance.

TopoDS_Wire on TopoDS_Face is constructed with the help of classes StepToTopoDS_TranslatePolyLoop, StepToTopoDS_TranslateEdgeLoop or StepToTopoDS_TranslateVertexLoop.

Class StepToTopoDS_TranslateShell

This class calls StepToTopoDS_TranslateFace::Init for each face. This class does not modify the tolerance value.

Class StepToTopoDS_TranslateCompositeCurve

TopoDS_Edges in TopoDS_Wire are constructed with the help of class BRepAPI_MakeEdge and have a tolerance 10^{-7}. Pcurves from a STEP file are translated if they are present and if read.surfacecurve.mode is not -3. The connection between segments of a composite curve (edges in the wire) is ensured by call to method ShapeFix_Wire::FixConnected() with a precision equal to the initial value of tolerance.

Class StepToTopoDS_TranslateCurveBoundedSurface

TopoDS_Face is constructed with tolerance Precision::Confusion().

TopoDS_Wire on TopoDS_Face is constructed with the help of class StepToTopoDS_TranslateCompositeCurve. Missing pcurves are computed using projection algorithm

with the help of method ShapeFix_Face::FixPcurves(). For resulting face method ShapeFix::SameParameter() is called. It calls standard BRepLib::SameParameter for each edge in each wire, which can either increase or decrease the tolerances of the edges and vertices. SameParameter writes the tolerance corresponding to the real deviation of pcurves from 3D curve which can be less or greater than the tolerance in a STEP file.

Class StepToTopoDS_Builder

Class StepToTopoDS_Builder is a high level class. Its methods perform translation with the help of the classes listed above. If the value of read.maxprecision.mode is set to 1 then the tolerance of subshapes of the resulting shape is limited by 0 and read.maxprecision.val. Else this class does not change the tolerance value.

Class StepToTopoDS_MakeTransformed

This class performs a translation of mapped_item entity and indirectly uses class StepToTopoDS_Builder. The tolerance of the resulting shape is not modified inside this method.

Healing of resulting shape in ShapeHealing component

Class ShapeFix_Wire

1. ShapeFix_Wire::FixSelfIntersection()

This method is intended for detecting and fixing self-intersecting edges and intersections of adjacent edges in a wire. It fixes self-intersections by cutting edges at the intersection point and/or by increasing the tolerance of the vertex (so that the vertex comprises the point of intersection). There is a maximum tolerance that can be set by this method transmitted as a parameter, currently is read.maxprecision.value.

When a self-intersection of one edge is found, it is fixed by one of the two methods:

- tolerance of the vertex of that edge which is nearest to the point of self-intersection is increased so that it comprises both its own old position and the intersection point
- the self-intersecting loop on the pcurve is cut out and a new pcurve is constructed. This can increase the tolerance of the edge.

The method producing a smaller tolerance is selected.

When an intersection of two adjacent edges is detected, edges are cut at that point. Tolerance of the common vertex of these edges is increased in order to comprise both the intersection point and the old position.

This method can increase the tolerance of the vertex up to a value of read.maxprecision.value.

2. ShapeFix_Wire::FixLacking()

This method is intended to detect gaps between pcurves of adjacent edges (with the precision of surface UVResolution computed from tolerance of a corresponding vertex) and to fix these gaps either by increasing the tolerance of the vertex, or by inserting a new degenerated edge (straight in parametric space).

If it is possible to compensate a gap by increasing the tolerance of the vertex to a value of less than the initial value of tolerance, the tolerance of the vertex is increased. Else, if the vertex is placed in a degenerated point then a degenerated edge is inserted.

3. ShapeFix_Wire::FixConnected()

This method is intended to force two adjacent edges in the wire to share the same vertex. This method can increase the tolerance of the vertex. The maximal value of tolerance is read.maxprecision.value.

2.6. Code architecture

2.6.1. List of the classes

package STEPControl

STEPControl_Reader

STEPControl_ActorRead

package StepToTopoDS

StepToTopoDS_Builder

StepToTopoDS_MakeTransformed

StepToTopoDS_TranslateShell

StepToTopoDS_TranslateFace

StepToTopoDS_TranslateEdgeLoop

StepToTopoDS_TranslatePolyLoop

StepToTopoDS_TranslateVertexLoop

StepToTopoDS_TranslateEdge

StepToTopoDS_TranslateVertex

package StepToGeom

StepToGeom_MakeCartesianPoint

StepToGeom_MakeCurve

StepToGeom_MakeLine

StepToGeom_MakeSurface

For more information refer to CDL.

2.6.2. API classes

package STEPControl

STEPControl_Controller

STEPControl_Reader

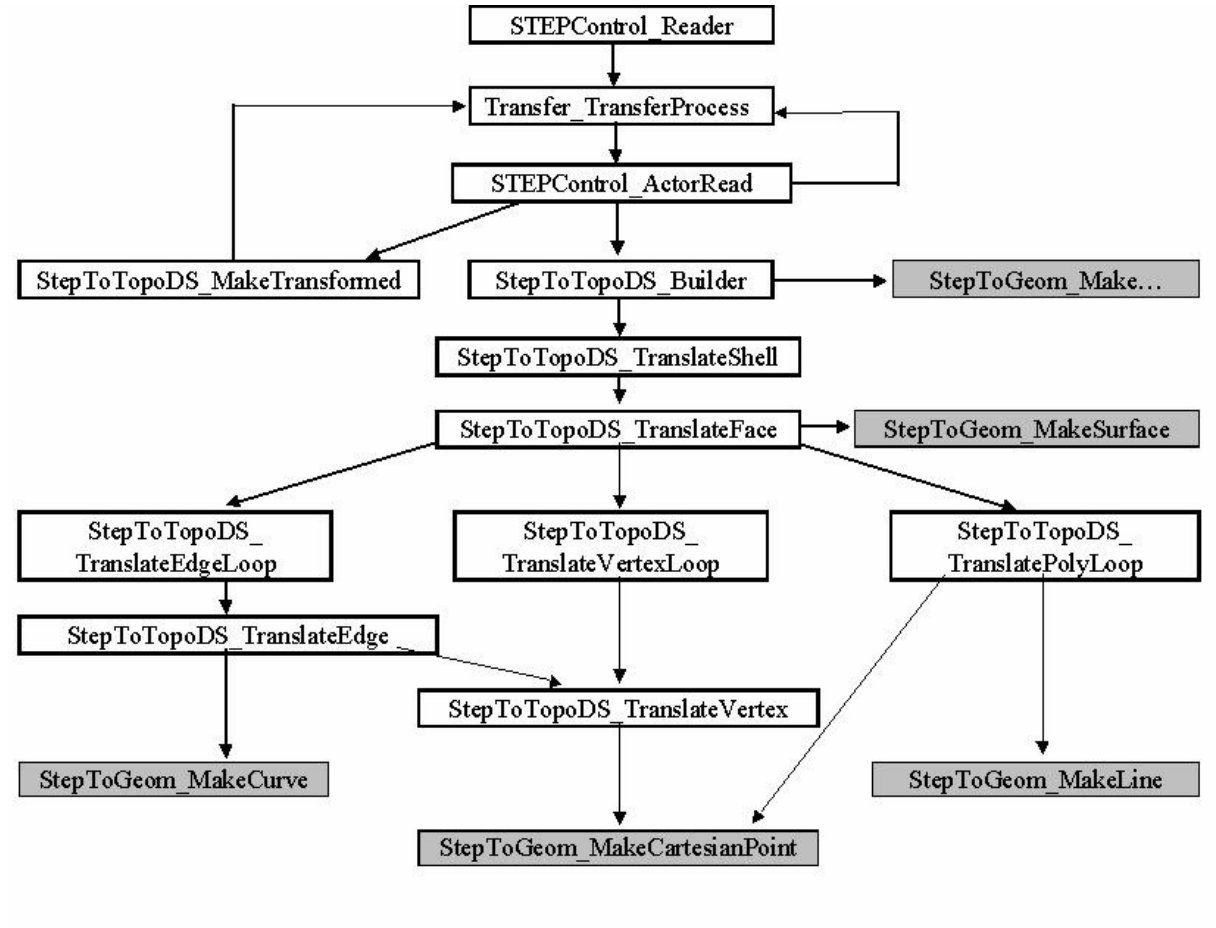
STEPControl_ActorRead

For a description of these classes refer to the chapter 4. API for reading/writing STEP.

2.6.3. Graph of calls

The following diagram illustrates the structure of calls in reading STEP.

The highlighted classes are intended to translate geometry.



2.7. Example

```
#include <STEPControl_Reader.hxx>
#include <TopoDS_Shape.hxx>
#include <BRepTools.hxx>

Standard_Integer main()
{
  STEPControl_Reader reader;
  reader.ReadFile("MyFile.stp");

  // Loads file MyFile.stp
  Standard_Integer NbRoots = reader.NbRootsForTransfer();
```

```
  // gets the number of transferable roots
  cout<<"Number of roots in STEP file: "<< NbRoots<<endl;

  Standard_Integer NbTrans = reader.TransferRoots();
  // translates all transferable roots, and returns the number of
  //successful translations
  cout<<"STEP roots transferred: "<< NbTrans<<endl;
  cout<<"Number of resulting shapes is: "<<reader.NbShapes()<<endl;

  TopoDS_Shape result = reader.OneShape();
  // obtain the results of translation in one Open CASCADE shape

  . . .

}
```

3. Writing STEP

3.1. Procedure

You can translate Open CASCADE shapes into STEP entities in the following steps:

1. initialize the process,
2. set the translation parameters,
3. perform the shape translation,
4. write the output file.

You can translate several shapes before writing a file. All these translations output a separate shape_representation entity in STEP file.

The user-defined option (parameter "write.step.schema") is provided to define which version of schema (AP214 CD or DIS, or AP203) is used for the output STEP file.

3.2. Domain covered

3.2.1. Writing geometry and topology

There are two families of Open CASCADE objects that can be translated:

- geometrical objects,
- topological shapes.

3.2.2. Writing assembly structures

The shapes organized in a structure of nested compounds can be translated either as simple compound shapes, or into the assembly structure, depending on the parameter "write.step.assembly", which is described below.

The assembly structure placed in the produced STEP file corresponds to the structure described in the ProSTEP Agreement Log (item 21) as the second alternative (assembly structure through representation_relationship / item_defined_transformation). To represent an assembly it uses entities of the representation_relationship_with_transformation type. Transformation operators used for locating assembly components are represented by item_defined_transformation entities.

If mode "write.step.assembly" is set to the values <ON> or <Auto> then an OCC shape consisting of nested compounds will be written as an assembly, otherwise it will be written as separate solids.

Please see also the sub-chapter "Mapping Open CASCADE shapes to STEP entities".

3.3. Description of the process

3.3.1. Initializing the process

Before performing any other operation you have to create a writer object:

```
STEPControl_Writer writer;
```

3.3.2. Setting the translation parameters

The following parameters are used for the Open CASCADE-to-STEP translation.

write.precision.mode

writes the precision value.

"Least" (-1) : the uncertainty value is set to the minimum tolerance of an Open CASCADE shape

"Average" (0) : the uncertainty value is set to the average tolerance of an Open CASCADE shape.

"Greatest" (1) : the uncertainty value is set to the maximum tolerance of an Open CASCADE shape

"Session" (2) : the uncertainty value is that of the write.precision.val parameter.

Read this parameter with:

```
Standard_Integer ic = Interface_Static::IVal("write.precision.mode");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("write.precision.mode",1))
.. error ..
```

Default value is 0.

write.precision.val

a user-defined precision value. This parameter gives the uncertainty for STEP entities constructed from Open CASCADE shapes when the write.precision.mode parameter value is 1.

- 0.0001: default
- any real positive (non null) value.

This value is stored in shape_representation in a STEP file as an uncertainty.

Read this parameter with:

```
Standard_Real rp = Interface_Static::RVal("write.precision.val");
```

Modify this parameter with:

```
if(!Interface_Static::SetRVal("write.precision.val",0.01))
.. error ..
```

Default value is 0.0001.

write.step.assembly

writing assembly mode.

0 (Off) : (default) writes STEP files without assemblies.

1 (On) : writes all shapes in the form of STEP assemblies.

2 (Auto) : writes shapes having a structure of (possibly nested) TopoDS_Compounds in the form of STEP assemblies, single shapes are written without assembly structures.

Read this parameter with:

```
Standard_Integer rp = Interface_Static::IVal("write.step.assembly");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("write.step.assembly",1))
.. error ..
```

Default value is 0.

write.step.schema

defines the version of schema used for the output STEP file:

1 or "AP214CD" (default): AP214, CD version (dated 26 November 1996),

2 or "AP214DIS": AP214, DIS version (dated 15 September 1998).

3 or "AP203": AP203, possibly with modular extensions (depending on data written to a file).

4 or “AP214IS”: AP214, IS version (dated 2002)

This parameter affects the following entities written to the STEP file:

<table>
<tr><th>AP214, CD version</th><th>AP214, DIS version</th><th>AP214,IS version</th><th>AP203</th></tr>
<tr><td>FILE_SCHEMA((`AUTOMOTIVE_DESIGN_CC2 { 1 2 10303 214 -1 1 5 4}'))</td><td>FILE_SCHEMA( (`AUTOMOTIVE_DESIGN { 1 2 10303 214 0 1 1 1}'))</td><td>FILE_SCHEMA(('AUTOMOTIVE_DESIGN { 1 0 10303 214 1 1 1 1 }'))</td><td>FILE_SCHEMA((`CONFIG_CONTROL_DESIGN'))</td></tr>
<tr><td>APPLICATION_PROTOCOL_ DEFINITION `committee draft', `automotive_design' ,1997,##)</td><td>APPLICATION_PROTOCOL_ DEFINITION('draft international standard', 'automotive_design', 1998,##)</td><td>APPLICATION_PROTOCOL_DEFINITION ('international standard', ‘automotive_design', 2000,##);</td><td>APPLICATION_PROTOCOL_ DEFINITION ('international standard', 'config_control_design', 1994,##)</td></tr>
<tr><td colspan="3">APPLICATION_CONTEXT('core data for automotive mechanical design processes')</td><td>APPLICATION_CONTEXT(‘configuration controlled 3D designs of mechanical parts and assemblies')</td></tr>
<tr><td>PRODUCT_TYPE(`part',$,(##))</td><td colspan="3">PRODUCT_RELATED_PRODUCT_CATEGORY( `part',$,(##))</td></tr>
<tr><td>MECHANICAL_CONTEXT(`',##, 'mechanical')</td><td>PRODUCT_CONTEXT (`',##, 'mechanical')</td><td>PRODUCT_CONTEXT ('',##,'mechanical');</td><td>MECHANICAL_CONTEXT(`',##, 'mechanical')</td></tr>
</table>

In addition, in AP203 mode more product and organizational entities are generated (entities like PERSON_AND_ORGANIZATION, SECURITY_CLASSIFICATION etc., as required by AP203).

Read this parameter with:

```
TCollection_AsciiString schema =
Interface_Static::CVal("write.step.schema");
```

Modify this parameter with:

```
if(!Interface_Static::SetCVal("write.step.schema","DIS"))
.. error ..
```

Default value is 1 ("CD").

NOTE

For the parameter "write.step.schema" to take effect, method STEPControl_Writer::Model(Standard_True) should be called after changing this parameter (corresponding command in DRAW is "newmodel").

write.step.product.name

Defines the text string that will be used for field ‘name’ of PRODUCT entities written to the STEP file.

Default value: “Open CASCADE STEP translator 6.1” (the 6.1 stands for Open CASCADE version number).

write.surfacecurve.mode

This parameter indicates whether parametric curves (curves in parametric space of surface) should be written into the STEP file. This parameter can be set to Off in order to minimize the size of the resulting STEP file.

Off (0) : writes STEP files without pcurves. This mode decreases the size of the resulting STEP file .

On (1) : (default) writes pcurves to STEP file

Read this parameter with:

```
Standard_Integer wp =
Interface_Static::IVal("write.surfacecurve.mode");
```

Modify this parameter with:

```
if(!Interface_Static::SetIVal("write.surfacecurve.mode",1))
.. error ..
```

Default value is On.

write.step.unit

Defines a unit in which the STEP file should be written. If set to unit other than MM, the model is converted to these units during the translation.

Default value is MM.

write.step.resource.name

write.step.sequence

These two parameters define the name of the resource file and the name of the sequence of operators (defined in that file) for Shape Processing, which is automatically performed by the STEP translator before translating a shape to a STEP file. Shape Processing is a user-configurable step, which is performed before the translation and consists in applying a set of operators to a resulting shape. This is a very powerful tool allowing customizing the shape and adapting it to the needs of a receiving application. By default the sequence consists of two operators: SplitCommonVertex and DirectFaces, which convert some geometry and topological constructs valid in Open CASCADE but not in STEP to equivalent definitions conforming to STEP format.

See description of parameter read.step.resource.name above for more details on using resource files.

Default values: read.step.resource.name - STEP, read.step.sequence - ToSTEP.

3.3.3. Performing the Open CASCADE shape translation

An Open CASCADE shape can be translated to STEP using one of the following models (shape_representations):

- manifold_solid_brep (advanced_brep_shape_representation)
- brep_with_voids (advanced_brep_shape_representation)
- faceted_brep (faceted_brep_shape_representation)
- shell_based_surface_model (manifold_surface_shape_representation)
- geometric_curve_set (geometrically_bounded_wireframe_shape_representation)

The enumeration **STEPControl_StepModelType** is intended to define a particular transferring model.

The following values of enumeration are allowed:

STEPControl_AsIs	Translator selects the resulting representation automatically, according to the type of CASCADE shape to translate it in its highest possible model
STEPControl_ManifoldSolidBrep	resulting entity is manifold_solid_brep or brep_with_voids
STEPControl_FacetedBrep	resulting entity is faceted_brep or faceted_brep_and_brep_with_voids Note that only planar-face shapes with linear edges can be written
STEPControl_ShellBasedSurfaceModel	resulting entity is shell_based_surface_model
STEPControl_GeometricCurveSet	resulting entity is geometric_curve_set

The following table shows which shapes can be translated in which mode:

STEP214Control_AsIs	any Open CASCADE shape
STEP214Control_ ManifoldSolidBrep	TopoDS_Solid, TopoDS_Shell, TopoDS_Compound (if it contains TopoDS_Solids and TopoDS_Shells.
STEP214Control_ FacetedBrep	TopoDS_Solid or TopoDS_Compound containing TopoDS_Solids if all its surfaces are Geom_Planes and all curves are Geom_Lines.
STEP214Control_ ShellBasedSurfaceMo del	TopoDS_Solid, TopoDS_Shell, TopoDS_Face and TopoDS_Compound (if it contains all mentioned shapes)
STEP214Control_ GeometricCurveSet	any Open CASCADE shape

If TopoDS_Compound contains any other types besides the ones mentioned in the table, these sub-shapes will be ignored.

In case if an Open CASCADE shape cannot be translated according to its mode the result of translation is void.

For further information see “Mapping Open CASCADE shapes to STEP entities” below.

```
STEP214Control_StepModelTope mode = STEP214Control_ManifoldSolidBrep;
IFSelect_ReturnStatus stat = writer.Transfer(shape,mode);
```

3.3.4. Writing the STEP file

Write the STEP file with:

```
IFSelect_ReturnStatus stat = writer.Write("filename.stp");
```

to give the file name.

3.4. Mapping Open CASCADE shapes to STEP entities

NOTE

Only STEP entities that have a corresponding Open CASCADE object and mapping of assembly structures are described in this paragraph. For a full list of STEP entities please refer to Appendix A.

3.4.1. Assembly structures and product information

The assembly structures are written to the STEP file if parameter write.step.assembly is 1 or 2.

Each TopoDS_Compound is written as an assembly with subshapes of that compound being components of the assembly. The structure of nested compounds is translated to the structure of nested assemblies. Shared subshapes are translated into shared components of assemblies. Shapes that are not compounds are translated into subtypes of shape_representation according to their type (see the next subchapter for details).

A set of STEP entities describing general product information is written to the STEP file together with the entities describing the product geometry, topology and assembly structure. Most of these entities are attached to the entities being subtypes of shape_representation, but some of them are created only one per STEP file.

The table below describes STEP entities, which are created when the assembly structure and product information are written to the STEP file, and shows how many of these entities are created. Note that the appearance of some of these entities depends on the version of the schema (AP214, CD, DIS or IS, or AP203).

CASCADE shape	STEP entity	Comments
	application_protocol_definition	One per STEP file, defines the application protocol used (depends on the schema version)
	application_context	One per STEP file, defines the application generating the file (AP214 or AP203)
TopoDS_Compound	shape_representation	Empty shape_representation describing the assembly. The components of that assembly are written as subtypes of shape_representation and are included to the assembly using next_assembly_usage_occurence entities.
TopoDS_Shape	subtypes of shape_representation	Depending on the shape type, see the tables below for mapping details
	next_assembly_usage_occurence	Describes the instance of component in the assembly by referring corresponding product_definitions. If the same component is included in the assembly several times (for

		example, with different locations), several next_assembly_usage_occurences are created.
	context_dependent_shape_representation	Describes the placement of a component in the assembly. One context_dependent_shape_representation corresponds to each next_assembly_usage_occurence entity.
	shape_representation_relationship_with_transformation	Together with the context_dependent_shape_representation describes the location of a component in the assembly.
	item_defined_transformation	Defines a transformation used for the location of a component in the assembly. Is referred by shape_representation_relationship_with_transformation
	shape_definition_representation	One per shape_representation
	product_definition_shape	One per shape_definition_representation and context_dependent_shape_representation
	product_definition	Defines a product, one per shape_definition_representation
	product_definition_formation	One per product_definition. All product_definition_formations in the STEP file have unique names.
	Product	One per product_definition_formation. All products in the STEP file have unique names.
	product_type (CD) or product_related_product_category (DIS,IS)	One per product
	Mechanical_context (CD) or product_context (DIS,IS)	One per product.
	product_definition_context	One per product_definition.

3.4.2. Topological shapes

CASCADE shape	STEP entity	Comments
TopoDS_Compound	geometric_curve_set	If the write mode is STEP214Control_GeometricCurveSet only 3D curves of the edges found in TopoDS_Compound and all its subshapes are translated
	manifold_solid_brep	If the write mode is STEP214Control_AsIs and TopoDS_Compound consists only of TopoDS_Solids
	shell_based_surface_m	If the write mode is STEP214Control_AsIs and

	odel	TopoDS_Compound consists of TopoDS_Solids, TopoDS_Shells and TopoDS_Faces
	geometric_curve_set	If the write mode is STEP214Control_AsIs and TopoDS_Compound contains TopoDS_Wires, TopoDS_Edges, TopoDS_Vertices
		If the write mode is not STEP214Control_AsIs or STEP214Control_GeometricCurveSet TopoDS_Solids, TopoDS_Shells and TopoDS_Faces are translated according to this table.
TopoDS_Solid	manifold_solid_brep	If the write mode is STEP214Control_AsIs or STEP214Control_ManifoldSolidBrep and CASCADE TopoDS_Solid has no voids.
	faceted_brep	If the write mode is STEP214Control_FacetedBrep.
	brep_with_voids	If the write mode is STEP214Control_AsIs or STEP214Control_ManifoldSolidBrep and CASCADE TopoDS_Solid has voids.
	shell_based_surface_model	If the write mode is STEP214Control_ShellBasedSurfaceModel.
	geometric_curve_set	If the write mode is STEP214Control_GeometricCurveSet. Only 3D curves of the edges are translated.
TopoDS_Shell in a TopoDS_Solid	closed_shell	If TopoDS_Shell is closed shell.
TopoDS_Shell	manifold_solid_brep	If the write mode is STEP214Control_ManifoldSolidBrep.
	shell_based_surface_model	If the write mode is STEP214Control_AsIs or STEP214Control_ShellBasedSurfaceModel.
	geometric_curve_set	If the write mode is STEP214Control_GeometricCurveSet. Only 3D curves of the edges are translated.
TopoDS_Face	advanced_face	
TopoDS_Wire in a TopoDS_Face	face_bound	The resulting face_bound contains poly_loop if write mode is faceted_brep or edge_loop if not .
TopoDS_Wire	geometric_curve_set	If the write mode is STEP214Control_GeometricCurveSet. Only 3D curves of the edges are translated.
TopoDS_Edge	oriented_edge	
TopoDS_Vertex	vertex_point	

3.4.3. Geometrical objects

CASCADE object	STEP entity	Comments

Points		
Geom_CartesianPoint	cartesian_point	
Geom2d_CartesianPoint		
TColgp_Array1OfPnt	polyline	
TColgp_Array1OfPnt2d		
Placements		
Geom_Axis1Plasement	axis1_placement	
Geom2d_AxisPlacement		
Geom_Axis2Placement	axis2_placement_3d	
Directions		
Geom_Direction	direction	
Geom2d_Direction		
Vectors		
Geom_Vector	vector	
Geom2d_Vector		
Curves		
Geom_Circle	circle	
Geom2d_Circle	circle	
	rational_b_spline_curve	
Geom_Ellipse	Ellipse	
Geom2d_Ellipse	Ellipse	
	rational_b_spline_curve	
Geom_Hyperbola	Hyperbola	
Geom2d_Hyperbola		
Geom_Parabola	Parabola	
Geom2d_Parabola		
Geom_BSplineCurve	b_spline_curve_with_knots	
	rational_b_spline_curve	if Geom_BsplineCurve is a rational BSpline
Geom2d_BSplineCurve	b_spline_curve_with_knots	
	b_spline_curve_with_knots_and_rational_b_spline_curve	if Geom2d_BSplineCurve is a rational Bspline
Geom_BezierCurve	b_spline_curve_with_knots	
Geom_Line	Line	
Geom2d_Line		

Surfaces		
Geom_Plane	Plane	
Geom_OffsetSurface	offset_surface	
Geom_ConicalSurface	conical_surface	
Geom_CylindricalSurface	cylindrical_surface	
Geom_OffsetSurface	offset_surface	
Geom_RectangularTrimmedSurface	rectangular_trimmed_surface	
Geom_SphericalSurface	spherical_surface	
Geom_SurfaceOfLinearExtrusion	surface_of_linear_extrusion	
Geom_SurfaceOfRevolution	surface_of_revolution	
Geom_ToroidalSurface	toroidal_surface	
	degenerate_toroidal_surface	if the minor radius is greater then the major one
Geom_BezierSurface	b_spline_surface_with_knots	
Geom_BsplineSurface	b_spline_surface_with_knots	
	b_spline_surface_with_knots_and_rational_b_spline_surface	if Geom_BSplineSurface is a rational Bspline

3.5. Tolerance management

There are four possible values for the uncertainty when writing a STEP file:

- user-defined value of the uncertainty
- minimal value of sub-shapes tolerances
- average value of sub-shapes tolerances
- maximal value of sub-shapes tolerances

The chosen value of the uncertainty is the final value that will be written into the STEP file.

See parameter write.precision.mode.

3.6. Code architecture

3.6.1. List of the classes

package STEPControl

STEPControl_Controller

STEPControl_Writer

STEPControl_ActorWrite

package TopoDSToStep

TopoDSToStep_Builder

TopoDSToStep_WireframeBuilder

TopoDSToStep_MakeBrepWithVoids

TopoDSToStep_MakeFacetedBrep

TopoDSToStep_MakeFacetedBrepAndBrepWithVoids

TopoDSToStep_MakeGeometricCurveSet

TopoDSToStep_MakeManifoldSolidBrep

TopoDSToStep_MakeShellBasedSurfaceModel

TopoDSToStep_FacetedTool

TopoDSToStep_MakeStepFace

TopoDSToStep_MakeStepWire

TopoDSToStep_MakeStepEdge

TopoDSToStep_MakeStepVertex

package GeomToStep

GeomToStep_MakeCartesianPoint

GeomToStep_MakeCurve

GeomToStep_MakePolyline

GeomToStep_MakeSurface

For more information refer to CDL.

3.6.2. API classes

package STEPControl

STEPControl_Controller

STEPControl_Writer

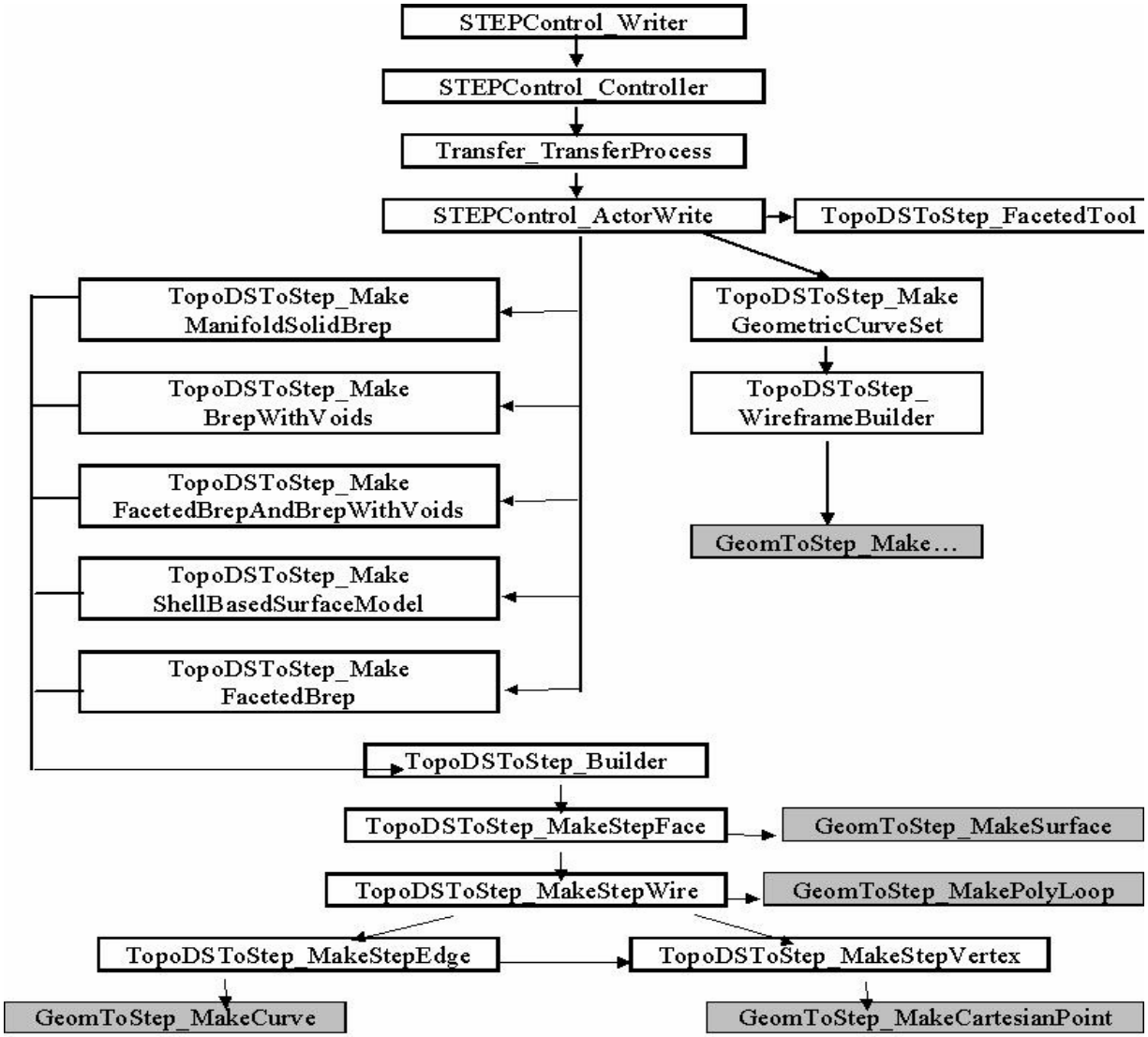
STEPControl_ActorWrite

For a description of these classes refer to the chapter 4. API for reading/writing STEP.

3.6.3. Graph of calls

The following diagram illustrates the structure of calls in writing STEP.

The highlighted classes are intended to translate geometry.



3.7. Example

```
#include <STEPControl.hxx>
#include <STEPControl_Writer.hxx>
#include <TopoDS_Shape.hxx>
#include <BRepTools.hxx>
#include <BRep_Builder.hxx>

Standard_Integer main()
{
TopoDS_Solid source;
. . .

STEPControl_Writer writer;
writer.Transfer(source, STEPControl_ManifoldSolidBrep);

// Translates TopoDS_Shape into manifold_solid_brep entity
writer.Write("Output.stp");
// writes the resulting entity in the STEP file

}
```

4. API for reading/writing STEP

4.1. Overview

This chapter contains a description of packages, which provide API for translating Open CASCADE shapes into STEP files and vice versa.

Package STEPControl provides API for reading and writing STEP files. This package deals with Open CASCADE shapes. It includes five API classes:

- STEPControl_Controller - for initializing the STEP norm,
- STEPControl_Reader - for reading STEP files,
- STEPControl_Writer - for writing STEP files,
- STEPControl_ActorRead - actor which performs translation of a STEP entity into an Open CASCADE shape,
- STEPControl_ActorRead – actor, which performs translation of an Open CASCADE shape into a STEP entity.

4.2. Package STEPControl

4.2.1. General description

This package provides tools to translate STEP entities to Open CASCADE shapes and vice versa.

STEP AP 214 and AP 203 entities can be read and translated.

File that is produced by this component conforms to either STEP AP214 (CD, DIS or IS version) or AP203 schema, according to parameter write.step.schema.

The result of importing a STEP file is an Open CASCADE shape. The result of exporting an Open CASCADE geometrical or topological object is a STEP file that conforms to AP 214 or AP203.

The package contains the following classes:

class STEPControl_Controller

class STEPControl_Reader

class STEPControl_Writer

class STEPControl_ActorRead

class STEPControl_ActorWrite

4.2.2. Enumeration STEPControl_StepModelType

enum STEPControl_StepModelType {

STEPControl_AsIs,

STEPControl_ManifoldSolidBrep,

STEPControl_BrepWithVoids,

STEPControl_FacetedBrep,

STEPControl_FacetedBrepAndBrepWithVoids,

STEPControl_ShellBasedSurfaceModel,

STEPControl_GeometricCurveSet,

STEPControl_Hybrid

}

Purpose: This enumeration is intended to define a particular transferring model for writing STEP. The following values of enumeration are allowed:

STEPControl_AsIs	translator selects the resulting representation automatically, according to the type of CASCADE shape to translate it in its highest possible model
STEPControl_ManifoldSolidBrep	resulting entity is manifold_solid_brep or brep_with_voids (only for solids, shells and compounds of them)
STEPControl_FacetedBrep	resulting entity is faceted_brep or faceted_brep_and_brep_with_voids (only for solids, shells or compounds of them and only if all surfaces are planes and curves are straight lines)
STEPControl_ShellBasedSurfaceModel	resulting entity is shell_based_surface_model (for solids, shells, faces or compounds of them)
STEPControl_GeometricCurveSet	resulting entity is geometric_curve_set (for any type of shape containing edges)

4.2.3. Class STEPControl_Controller

General description

This class is intended to provide an appropriate initialization of the STEP norm, namely it includes a set of necessary parameters for STEP translation.

After completing the initialization the STEP norm can be used.

```
Inheritance
        XSControl_Controller
        MMgt_TShared
        Standard_Trancient
```

Methods

Constructors

- STEPControl_Controller();

Purpose: Initializes the use of the STEP norm in the first call. Creates an object with default parameters.

Method for performing initialization

- STEPControl_Controller::Init

```
static Standard_Boolean Init();
```

Purpose: Performs standard initialization creating a controller object for the STEP norm. Returns True.

Method for creating a STEP Model object

- STEPControl_Controller::NewModel

```
Handle_Interface_InterfaceModel NewModel() const;
```

Purpose: Creates a new empty model ready to receive data of the STEP norm. Sets the Header of the model to default values from the STEP template).

.

Method for getting the actor object

- STEPControl_Controller::ActorRead

```
Handle_Transfer_ActorOfTransientProcess ActorRead(
const Handle(Interface_InterfaceModel)& model) const;
```

Purpose: Returns the actor object for reading (actually, it is of the type STEPControl_ActorRead).

Method for translating an Open CASCADE shape

- STEPControl_Controller::TransferWriteShape

```
virtual IFSelect_ReturnStatus TransferWriteShape(const TopoDS_Shape& shape,
                        const Handle(Transfer_FinderProcess)& FP,
                        const Handle(Interface_InterfaceModel)& model,
                        const Standard_Integer modetrans = 0) const;
```

Purpose: Translates an Open CASCADE shape into a STEP entity and adds it to the interface model <model>. <modetrans> defines the type of the resulting STEP entity:

0 - autodetect according to the type of the shape.

1 - faceted_brep

2 - shell_based_surface_model

3 - manifold_solid_brep

4 - geometric_curve_set

Returns:

IFSelect_RetDone: OK

IFSelect_RetError: if <modetrans> is not equal to 0,1,2,3,4, or <model> is not a STEP one.

IFSelect_Fail: if result is null.

4.2.4. Class STEPControl_Reader

General description

This high level class performs the reading of a STEP file and translating its contents into an Open CASCADE shape. Only geometrical and topological entities conformant to the STEP AP 214 and the AP 203 formats can be translated into Open CASCADE shapes. The result of translation is an Open CASCADE topological shape.

```
Inheritance
        XSControl_Reader
```

This class complements the XSControl_Reader class (deals directly with WorkSession object).

Methods

Constructors

```
STEPControl_Reader();
```

Purpose: Creates a reader and initializes a new controller with the help of STEPControl_Controller::Init().

```
STEPControl_Reader( const Handle(XSControl_WorkSession)& WS,
                    const Standard_Boolean scratch = Standard_True);
```

Purpose: Creates a reader and defines its work session. If <scratch> is true the method sets a new interface model.

Method for dealing with STEP Model

- STEPControl_Reader::StepModel

```
Handle_StepData_StepModel StepModel() const;
```

Purpose: Returns a STEP Model.

Method for performing translation

- STEPControl_Reader::TransferRoot

```
Standard_Boolean TransferRoot(const Standard_Integer num = 1)
```

Purpose: Translates a STEP root specified by its rank. Performs translation with the help of method XSControl_Reader::TransferOneRoot.

4.2.5. Class STEPControl_Writer

General description

This class is intended to create and write a file compliant to STEP AP 214 or to AP203 out of Open CASCADE shapes. The file produced by this component conforms to STEP AP214 or AP203. This component gives a possibility to write a STEP file containing the following models (shape_representations):

faceted_brep

shell_based_surface_model

manifold_solid_brep

geometric_curve_set

The write mode is selected by specifying the appropriate parameter.

Export into a STEP file can be performed on the basis of either an already existing STEP model (its representation in the memory) or a new one.

The translation of Open CASCADE shapes (which can be a 2D or a 3D geometrical or topological object) into a STEP file is fulfilled in the following steps:

- Setting the translation parameters
- The translation. It can be performed in one or several operations. Each operation defines a root in the STEP file.
- Writing the STEP file. The output file is specified by its name.

Methods

Constructors

```
STEPControl_Writer();
```

Purpose: Creates a writer, initializes a default STEPControl_Controller and sets a new work session.

```
STEPControl_Writer( const Handle(XSControl_WorkSession)& WS,
const Standard_Boolean scratch = Standard_True);
```

Purpose: Creates a writer, initializes a default STEPControl_Controller with an already existing WorkSession object. If <scratch> if True, this method clears the already recorded data.

Methods for dealing with the uncertainty value

- STEPControl_Writer::SetTolerance

```
void SetTolerance(const Standard_Real Tol) ;
```

Purpose: Sets the value of tolerance, which will be written into a file as a value of uncertainty.

- STEPControl_Writer::UnsetTolerance

```
void UnsetTolerance() ;
```

Purpose: Unsets the tolerance set by method SetTolerance. The value of uncertainty will be then calculated as an average tolerance of the Open CASCADE shape.

Methods for dealing with a WorkSession object

- STEPControl_Writer::SetWS

```
void SetWS( const Handle(XSControl_WorkSession)& WS,
const Standard_Boolean scratch = Standard_True) ;
```

Purpose: Sets an existing WorkSession WS.

- STEPControl_Writer::WS

```
Handle_XSControl_WorkSession WS() const;
```

Purpose: Returns a current WorkSession object.

Method for obtaining a model object

- STEPControl_Writer::Model

```
Handle_StepData_StepModel Model(
const Standard_Boolean newone = Standard_False) ;
```

Purpose: Returns a produced Model. Produces a new one if such is not yet produced or if <newone> is True.

Method for performing the transfer process

- STEPControl_Writer::Transfer

```
IFSelect_ReturnStatus Transfer( const TopoDS_Shape& sh,
const STEPControl_StepModelType mode) ;
```

Purpose: Translates a shape <sh> to STEP entities. Allowed modes are:

STEPControl_AsIs	to select the resulting entity type automatically, according to the type of the Open CASCADE shape to translate it into its highest possible model,

STEPControl_ManifoldSolidBrep	to translate into manifold_solid_brep or brep_with_voids,
STEPControl_ShellBasedSurfaceModel	to translate into shell_based_surface_model,
STEPControl_FacetedBrep	to translate into faceted_brep,
STEPControl_GeometricCurveSet	to translate into geometric_curve_set.

Returns the transfer status (see Controller from STEPControl)

Method for creating a STEP file

- STEPControl_Writer::Write

```
IFSelect_ReturnStatus Write(const Standard_CString filename) ;
```

Purpose: Writes a model into the file <filename>. Performs this operation with the help of method XSControl_WorkSession::SendAll.

Method for obtaining statistics

- STEPControl_Writer::PrintStatsTransfer

```
void PrintStatsTransfer( const Standard_Integer what,
const Standard_Integer mode = 0) const;
```

Purpose: Intended to display all statistics on the last translation performed. Calls method XSControl_TransferWriter::PrintStats.

4.2.6. Class STEPControl_ActorRead

General description

This class is intended for translation of AP214 and AP203 entities of the following types:

faceted_brep

brep_wiht_voids

manifold_solid_brep

shell_based_surface_model

geometric_curve_set

mapped_item

face_surface

It can also translate assembly structures, shape_definition_representation and shape_representation referencing to the mentioned entities.

The result of translation is Transfer Binder with the resulting Open CASCADE shape.

```
Inheritance
        Transfer_ActorOfTransientProcess
        Transfer_ActorOfProcessForTransient
        MMgt_TShared
        Standard_Trancient
```

Methods

Constructor

```
STEPControl_ActorRead();
```

Purpose: Empty constructor.

Method for recognizing entities

- STEPControl_ActorRead::Recognize

```
virtual Standard_Boolean Recognize(
const Handle(Standard_Transient)& start) ;
```

Purpose: Returns True if entity <start> can be translated by this Actor class, i.e. if entity type is one of the types listed above.

Method for performing translation

- STEPControl_ActorRead::Transfer

```
virtual Handle_Transfer_Binder Transfer(
            const Handle(Standard_Transient)& start,
            const Handle(Transfer_TransientProcess)& TP) ;
```

Purpose: Performs access to transferable entities and translates them with the help of the class StepToTopoDS_Builder. Sets translation parameters with the help of method PrepareUnits.

- STEPControl_ActorRead::PrepareUnits

```
void PrepareUnits( const Handle(Standard_Transient)& start,
const Handle(Transfer_TransientProcess)& TP) ;
```

Purpose: If <start> is shape_definition_representation or shape_repesentation entity, this method sets the length and plane angle units taken from <entity> for performing translation. Sets the tolerance value to uncertainty if read.precision.mode is "File".

4.2.7. Class STEPControl_ActorWrite

General description

This class provides an actor for performing the translation of Open CASCADE shapes to STEP AP203 or AP214 entities according to the write mode and schema version (CD , DIS or IS).

An Open CASCADE shape can be translated to STEP using one of the following models (shape_representations):

manifold_solid_brep (advanced_brep_shape_representation)

brep_with_voids (advanced_brep_shape_representation)

faceted_brep (faceted_brep_shape_representation)

shell_based_surface_model (manifold_surface_shape_representation)

geometric_curve_set (geometrically_bounded_wireframe_shape_representation)

During the translation the actor computes the uncertainty value to be applied to the resulting model (shape_representation).

This actor also writes STEP assembly structures (according to group mode, which is initialised by the value of parameter write.step.assembly).

This parameter is used to create a STEP entity with or without assemblies. If write.step.assembly is 1, assemblies are created, if it is 0 assemblies are not created. If it is 2, assemblies are created for compound shapes containing more than one component.

The transfer mode is enumeration STEPControl_StepModeType, which defiles the resulting model.

```
Inheritance
```

```
Transfer_ActorOfFinderProcess
Transfer_ActorOfProcessForFinder
MMgt_TShared
Standard_Transient
```

Methods

Constructors

```
STEPControl_ActorWrite();
```

Purpose: Sets the write mode to shell_based_surface_model, group mode to 0 and the default value for the uncertainty that will be computed as an average tolerance for each shape.

Method for setting and obtaining translation parameters:

- STEPControl_ActorWrite::SetMode

```
void SetMode(const STEPControl_StepModelType M) ;
```

Purpose: Sets the write mode. <M> can be of the following values:

STEPControl_AsIs	to select the resulting entity type automatically, according to the type of Open CASCADE shape to translate it into its highest possible model,
STEPControl_ManifoldSolidBrep	to translate into manifold_solid_brep or brep_with_voids,
STEPControl_ShellBasedSurfaceModel	to translate into shell_based_surface_model,
STEPControl_FacetedBrep	to translate into faceted_brep,
STEPControl_GeometricCurveSet	to translate into geometric_curve_set.

- STEPControl_ActorWrite::Mode

```
STEPControl_StepModelType Mode();
```

Purpose: Returns the write mode.

- STEPControl_ActorWrite::SetGroupMode

```
void SetGroupMode(const Standard_Integer mode) ;
```

Purpose. Sets the group mode, which determines whether assemblies should be created or not. <mode> can be one of the following values:

0 - result of translating Open CASCADE shape does not contain assemblies,

1 - result of translating Open CASCADE shape contains assemblies,

>1 - result of translating Open CASCADE shape can either contain assemblies or not, depending on the original shape.

- STEPControl_ActorWrite::GroupMode

```
Standard_Integer GroupMode() const;
```

Purpose: Returns the group mode.

- STEPControl_ActorWrite::SetTolerance

```
void SetTolerance(const Standard_Real Tol) ;
```

Purpose: Sets the value of uncertainty for STEP entities. If <Tol> is more than zero, then <Tol> will be taken as an uncertainty, if <Tol> is less than the zero value of the uncertainty it will be calculated as an average tolerance of the shape.

Method for recognizing entities

- STEPControl_ActorWrite::Recognize

```
virtual Standard_Boolean Recognize(const Handle(Transfer_Finder)&
start) ;
```

Purpose. Checks whether a shape can be translated into a STEP entity according to a defined model. This method returns True if the shape referenced by <start> can be translated and False if it cannot.

TopoDS_Solid, TopoDS_Shell can be translated into manifold_solid_brep.

TopoDS_Solid can be translated into brep_with_voids.

TopoDS_Solid can be translated into faceted_brep if all its surfaces are planes and all its curves are lines.

TopoDS_Solid, TopoDS_Shell, TopoDS_Face can be translated into shell_based_surface_model.

Any Open CASCADE shape can be translated into geometric_curve_set.

Method for performing the translation

- STEPControl_ActorWrite::Transfer

```
virtual Handle_Transfer_Binder Transfer(
                const Handle(Transfer_Finder)& start,
                const Handle(Transfer_FinderProcess)& FP) ;
```

Purpose: Translates a single Open CASCADE shape referenced by <start> into a STEP entity according to the tolerance, the write mode and the group mode.

Performs translation with the help of package TopoDSToStep and its Make... classes. Returns the transfer binder.

See also: Package TopoDSToStep, classes TopoDSToStep_Make...

4.3. Package STEPConstruct

4.3.1. General description

This package defines tools for creation and investigation of specific STEP constructs used for representing various kinds of data, such as product and assembly structure, unit contexts, and associated information. These structures are created according to the current schema (AP203, AP214 CD2, DIS or IS), which is defined by the parameter write.step.schema.

The package contains the following classes:

class STEPConstruct_Styles

class STEPConstruct_Part

4.3.2. Class STEPConstruct_Styles

General description

This class provides a mechanism for reading and writing shape styles (such as color) to and from a STEP file. This tool maintains a list of styles, either taking them from the STEP model (when reading), or filling it by calls to AddStyle or directly (for writing). Some methods deal with general structures of styles and presentations in STEP, but there are methods dealing with a particular implementation of colors (implemented in accordance with the "Recommended Practices for colors and layers").

Methods

Constructors

```
STEPConstruct_Styles();
```

Purpose: Empty constructor

```
STEPConstruct_Styles(const Handle(XSControl_WorkSession)& WS);
```

Purpose: Creates an object and calls Init

Method for initializing the object

- STEPConstruct_Styles::Init

```
Standard_Boolean Init(const Handle(XSControl_WorkSession)& WS) ;
```

Purpose: Initializes a tool; returns True if succeeded. The XSControl_WorkSession object is used to access data

Methods for dealing with styles

- STEPConstruct_Styles::NbStyles

```
Standard_Integer NbStyles() const;
```

Purpose: Returns the number of defined styles.

- STEPConstruct_Styles::Style

```
Handle_StepVisual_StyledItem Style(const Standard_Integer i) const;
```

Purpose: Returns a style with a given index

- STEPConstruct_Styles::ClearStyles

```
void ClearStyles() ;
```

Purpose: Clears all defined styles

- STEPConstruct_Styles::AddStyle

```
void AddStyle(const Handle(StepVisual_StyledItem)& style) ;
```

Purpose: Adds a style to a sequence (defines a style).

- STEPConstruct_Styles::AddStyle

```
Handle_StepVisual_StyledItem AddStyle(
            const Handle(StepRepr_RepresentationItem)& item,
            const Handle(StepVisual_PresentationStyleAssignment)& PSA,
            const Handle(StepVisual_StyledItem)& Override) ;
```

Purpose: Creates a style linking giving PSA (presentation_style_assignment) to the item, and adds it to the sequence of stored styles. If Override is not Null, then the resulting style will be of the subtype overriding_styled_item (else just simple styled_item).

- STEPConstruct_Styles::AddStyle

```
Handle_StepVisual_StyledItem AddStyle( const TopoDS_Shape& Shape,
            const Handle(StepVisual_PresentationStyleAssignment)& PSA,
            const Handle(StepVisual_StyledItem)& Override) ;
```

Purpose: Creates a style linking giving PSA (presentation_style_assignment) to the Shape, and adds it to the sequence of stored styles. If Override is not Null, then the resulting style will be of the subtype

overriding_styled_item (else just simple styled_item). The Shape is used to find a corresponding STEP entity by call to STEPConstruct::FindEntity(), then the previous method is called.

- STEPConstruct_Styles::CreateMDGPR

```
Standard_Boolean CreateMDGPR(const
Handle(StepRepr_RepresentationContext)& Context) ;
```

Purpose: Creates MDGPR (MECHANICAL_DESIGN_GEOMETRIC_PRESENTATION_ REPRESENTATION), fills it with all the styles previously defined, and adds it to the model.

- STEPConstruct_Styles::FindContext

```
Handle_StepRepr_RepresentationContext FindContext(const TopoDS_Shape&
Shape) const;
```

Purpose: Searches the STEP model for the RepresentationContext in which given shape is defined. This context (if found) can be used then in call to CreateMDGPR()

- STEPConstruct_Styles::LoadStyles

```
Standard_Boolean LoadStyles() ;
```

Purpose: Searches the STEP model for the MDGPR (MECHANICAL_DESIGN_GEOMETRIC_ PRESENTATION_REPRESENTATION) or DM (draughting_model) entities (which bring styles) and reads styles from these entities, thus filling the sequence of styles.

Methods for dealing with colors

- STEPConstruct_Styles::MakeColorPSA

```
Handle_StepVisual_PresentationStyleAssignment MakeColorPSA(
        const Handle(StepRepr_RepresentationItem)& item,
        const Handle(StepVisual_Colour)& SurfCol,
        const Handle(StepVisual_Colour)& CurveCol) const;
```

Purpose: Creates a presentation_style_assignment entity defining two colors (for filling surfaces and curves)

- STEPConstruct_Styles::GetColorPSA

```
Handle_StepVisual_PresentationStyleAssignment GetColorPSA(
            const Handle(StepRepr_RepresentationItem)& item,
            const Handle(StepVisual_Colour)& Col) ;
```

Purpose: Returns a PresentationStyleAssignment entity, which defines the surface and curve colors as Col. This PSA is either created or taken from the internal map where all PSAs are created by this method are remembered.

- STEPConstruct_Styles::GetColors

```
Standard_Boolean GetColors(
        const Handle(StepVisual_StyledItem)& style,
        Handle(StepVisual_Colour)& SurfCol,
        Handle(StepVisual_Colour)& BoundCol,
        Handle(StepVisual_Colour)& CurveCol) const;
```

Purpose: Extract color definitions from the style entity. For each type of color supported, the result can be either NULL if it is not defined by that style, or the last definition (if they are one or more)

Methods for converting a STEP and Open CASCADE color definition

- STEPConstruct_Styles::EncodeColor

```
Handle_StepVisual_Colour EncodeColor(const Quantity_Color& Col) ;
```

Purpose: Creates a STEP color entity from a given Quantity_Color. The analysis is performed for whether the color corresponds to the one of standard colors predefined in STEP. In that case, draughting_predefined_colour entity is created instead of rgb_colour.

- STEPConstruct_Styles::DecodeColor

```
Standard_Boolean DecodeColor(const Handle(StepVisual_Colour)&
Colour,Quantity_Color& Col) ;
```

Purpose: Decodes STEP color and fills the Quantity_Color. Returns True if OK or False if color is not recognized

4.3.3. Class STEPConstruct_Part

General description

Provides tools for creating STEP structures associated with shape_definition_representation, such as product, product_definition_formation etc., as required by the current schema (parameter write.step.schema). Also allows investigating and modifying this data.

Methods

Constructors

```
STEPConstruct_Part();
```

Purpose: Empty constructor

Methods for initializing and obtaining shape_definition_representation

- STEPConstruct_Part::MakeSDR

```
void MakeSDR( const Handle(StepShape_ShapeRepresentation)& aShape,
              const Handle(TCollection_HAsciiString)& aName,
              const Handle(StepBasic_ApplicationContext)& AC) ;
```

Purpose: Creates shape_definition_representation according to the currently active schema (AP203 or AP214 CD2, DIS or IS), which is taken from parameter write.step.schema. Creates all necessary product description entities as well.

- STEPConstruct_Part::ReadSDR

```
void ReadSDR(const Handle(StepShape_ShapeDefinitionRepresentation)&
aSDR) ;
```

Purpose: Sets the current SDR (shape_definition_representation) to the specified shape_definition_ representation <aSDR >

- STEPConstruct_Part::SDRValue

```
Handle_StepShape_ShapeDefinitionRepresentation SDRValue() const;
```

Purpose: Returns the current SDR or Null if no SDR is set or created

Method for obtaining the done status

- STEPConstruct_Part::IsDone

```
Standard_Boolean IsDone() const;
```

Purpose: Returns the done status

Method for obtaining shape_representation

- STEPConstruct_Part::SRValue

```
Handle_StepShape_ShapeRepresentation SRValue() const;
```

Purpose: Returns used_representation from the current SDR or Null if not done

Methods for dealing with product_context

- STEPConstruct_Part::PC

```
Handle_StepBasic_ProductContext PC() const;
```

Purpose: Returns the product_context associated with the current SDR

- STEPConstruct_Part::PCname

```
Handle_TCollection_HAsciiString PCname() const;
```

Purpose: Returns the Name of the product_context associated with the current SDR

- STEPConstruct_Part::PCdisciplineType

```
Handle_TCollection_HAsciiString PCdisciplineType() const;
```

Purpose: Returns the discipline type of the product_context associated with the current SDR

- STEPConstruct_Part::SetPCname

```
void SetPCname(const Handle(TCollection_HAsciiString)& name) ;
```

Purpose: Sets the Name of the product_context associated with the current SDR

- STEPConstruct_Part::SetPCdisciplineType

```
void SetPCdisciplineType(const Handle(TCollection_HAsciiString)&
label);
```

Purpose: Sets the discipline type of the product_context associated with the current SDR

Methods dealing with application_context

- STEPConstruct_Part::AC

```
Handle_StepBasic_ApplicationContext AC() const;
```

Purpose: Returns the application_context associated with the current SDR

- STEPConstruct_Part::ACapplication

```
Handle_TCollection_HAsciiString ACapplication() const;
```

Purpose: Returns the application of the application_context associated with the current SDR

- STEPConstruct_Part::SetACapplication

```
void SetACapplication(const Handle(TCollection_HAsciiString)& text) ;
```

Purpose: Sets the application of the application_context associated with the current SDR

- STEPConstruct_Part::PDC

```
Handle_StepBasic_ProductDefinitionContext PDC() const;
```

Purpose: Returns the product_definition_context associated with the current SDR

- STEPConstruct_Part::PDCname

```
Handle_TCollection_HAsciiString PDCname() const;
```

Purpose: Returns the name of the product_definition_context associated with the current SDR

- STEPConstruct_Part::PDCstage

```
Handle_TCollection_HAsciiString PDCstage() const;
```

Purpose: Returns the life cycle stage of the product_definition_context associated with the current SDR

- STEPConstruct_Part::SetPDCname

```
void SetPDCname(const Handle(TCollection_HAsciiString)& label) ;
```

Purpose: Sets the name of the product_definition_context associated with the current SDR

- STEPConstruct_Part::SetPDCstage

```
void SetPDCstage(const Handle(TCollection_HAsciiString)& label) ;
```

Purpose: Sets the life cycle stage of the product_definition_context associated with the current SDR

Methods dealing with the product

- STEPConstruct_Part::Product

```
Handle_StepBasic_Product Product() const;
```

Purpose: Returns the product associated with the current SDR

- STEPConstruct_Part::Pid

```
Handle_TCollection_HAsciiString Pid() const;
```

Purpose: Returns ID of the product associated with the current SDR

- STEPConstruct_Part::Pname

```
Handle_TCollection_HAsciiString Pname() const;
```

Purpose: Returns the name of the product associated with the current SDR

- STEPConstruct_Part::Pdescription

```
Handle_TCollection_HAsciiString Pdescription() const;
```

Purpose: Returns the description of the product associated with the current SDR

- STEPConstruct_Part::SetPid

```
void SetPid(const Handle(TCollection_HAsciiString)& id) ;
```

Purpose: Sets ID of the product associated with the current SDR

- STEPConstruct_Part::SetPname

```
void SetPname(const Handle(TCollection_HAsciiString)& label) ;
```

Purpose: Sets the name of the product associated with the current SDR

- STEPConstruct_Part::SetPdescription

```
void SetPdescription(const Handle(TCollection_HAsciiString)& text) ;
```

Purpose: Sets the description of the product associated with the current SDR

Methods dealing with product_definition_formation

- STEPConstruct_Part::PDF

```
Handle_StepBasic_ProductDefinitionFormation PDF() const;
```

Purpose: Returns the product_definition_formation associated with the current SDR

- STEPConstruct_Part::PDFid

```
Handle_TCollection_HAsciiString PDFid() const;
```

Purpose: Returns the ID of the product_definition_formation associated with the current SDR

- STEPConstruct_Part::PDFdescription

```
Handle_TCollection_HAsciiString PDFdescription() const;
```

Purpose: Returns the description of the product_definition_formation associated with the current SDR

- STEPConstruct_Part::SetPDFid

```
void SetPDFid(const Handle(TCollection_HAsciiString)& id) ;
```

Purpose: Sets the ID of the product_definition_formation associated with the current SDR

- STEPConstruct_Part::SetPDFdescription

```
void SetPDFdescription(const Handle(TCollection_HAsciiString)& text) ;
```

Purpose: Sets the description of the product_definition_formation associated with the current SDR

Methods dealing with product_definition

- STEPConstruct_Part::PD

```
Handle_StepBasic_ProductDefinition PD() const;
```

Purpose: Returns the product_definition associated with the current SDR

- STEPConstruct_Part::PDdescription

```
Handle_TCollection_HAsciiString PDdescription() const;
```

Purpose: Returns the description of the product_definition associated with the current SDR

- STEPConstruct_Part::SetPDdescription

```
void SetPDdescription(const Handle(TCollection_HAsciiString)& text) ;
```

Purpose: Sets the description of the product_definition associated with the current SDR

Methods dealing with product_definition_shape

- STEPConstruct_Part::PDS

```
Handle_StepRepr_ProductDefinitionShape PDS() const;
```

Purpose: Returns the product_definition_shape associated with the current SDR

- STEPConstruct_Part::PDSname

```
Handle_TCollection_HAsciiString PDSname() const;
```

Purpose: Returns the name of the product_definition_shape associated with the current SDR

- STEPConstruct_Part::PDSdescription

```
Handle_TCollection_HAsciiString PDSdescription() const;
```

Purpose: Returns the description of the product_definition_shape associated with the current SDR

- STEPConstruct_Part::SetPDSname

```
void SetPDSname(const Handle(TCollection_HAsciiString)& label) ;
```

Purpose: Sets the name of the product_definition_shape associated with the current SDR

- STEPConstruct_Part::SetPDSdescription

```
void SetPDSdescription(const Handle(TCollection_HAsciiString)& text) ;
```

Purpose: Sets the description of the product_definition_shape associated with the current SDR

Methods dealing with product_related_product_category

- STEPConstruct_Part::PRPC

```
Handle_StepBasic_ProductRelatedProductCategory PRPC() const;
```

Purpose: Returns the product_related_product_category associated with the current SDR

- STEPConstruct_Part::PRPCname

```
Handle_TCollection_HAsciiString PRPCname() const;
```

Purpose: Returns the name of the product_related_product_category associated with the current SDR

- STEPConstruct_Part::PRPCdescription

```
Handle_TCollection_HAsciiString PRPCdescription() const;
```

Purpose: Returns the description of the product_related_product_category associated with the current SDR

- STEPConstruct_Part::SetPRPCname

```
void SetPRPCname(const Handle(TCollection_HAsciiString)& label) ;
```

Purpose: Sets the name of the product_related_product_category associated with the current SDR

- STEPConstruct_Part::SetPRPCdescription

```
void SetPRPCdescription(const Handle(TCollection_HAsciiString)& text) ;
```

Purpose: Sets the description of the product_related_product_category associated with the current SDR

5. Physical STEP file reading and writing

The following paragraphs describe the loading of data from a physical STEP file into a STEP model and data writing from a STEP model to a STEP file.

5.1. Architecture of STEP Read and Write classes

5.1.1. General principles

To perform data loading from a STEP file and to translate this data it is necessary to create correspondence between the EXPRESS schema and the structure of CDL classes. There are two possibilities to organize such correspondence: the so-called early binding and late binding.

Late binding means that the processor works with a description of the schema. The processor builds a dictionary of entities and can recognize and read any entity that is described in the schema. To change the behavior and the scope of processor based on late binding it is enough to change the description of the schema. However, this binding has some disadvantages (for example low speed of reading process).

In case of early binding, the structure of CDL classes is created beforehand with the help of a specific automatic tool or manually. If the processor finds an entity that is not found in this schema, it will simply be ignored. The processor calls constructors of appropriate classes and their read methods. To add a new type in the scope of the processor it is necessary to create a class corresponding to the new entity.

The STEP processor is based on early binding principles. It means that specific classes for each EXPRESS type have been created with the help of an automatic tool from the EXPRESS schema. There are two CDL classes for each EXPRESS type. The first class (named the representing class) represents the STEP entity in memory. The second one (RW - class) is intended to perform the initialization of the representing class and to output data to an intermediate structure to be written in a STEP file.

5.1.2. Complex entities

EXPRESS schema allows multiple inheritance. Entities that are built on the basis of multiple inheritance are called complex entities. Multiple inheritance is not available in CDL. EXPRESS enables any type of complex entities that can be inherited from any EXPRESS type. In the manner of early binding it is not possible to create a CDL class for any possible complex type. Thus, only widespread complex entities have corresponding representing classes and RW-classes that are created manually beforehand.

5.2. Physical file reading

Physical file reading consists of the following steps:

1. Loading a STEP file and syntactic analysis of its contents
2. Mapping STEP entities to the array of strings
3. Creating empty Open CASCADE objects representing STEP entities
4. Initializing Open CASCADE objects
5. Building a references graph

5.2.1. Loading a STEP file and syntactic analysis of its contents

In the first phase, a STEP file is syntactically checked and loaded in memory as a sequence of strings.

Syntactic check is performed on the basis of rules defined in step.lex and step.yacc files. Files step.lex and step.yacc are located in the StepFile nocdlpack development unit. These files describe text encoding of STEP data structure (for additional information see ISO 10303 Part 21). The step.lex file describes the lexical structure of the STEP file. It describes identifiers, numbers, delimiters, etc. The step.yacc file describes the syntactic structure of the file, such as entities, parameters, and headers.

These files have been created only once and need to be updated only when norm ISO 10303-21 is changed.

5.2.2. Mapping STEP entities to arrays of strings

For each entity specified by its rank number the arrays storing its identifier, STEP type and parameters are filled.

5.2.3. Creating empty Open CASCADE objects that represent STEP entities

For each STEP entity an empty Open CASCADE object representing this entity is created. A map of correspondence between entity rank and Open CASCADE object is created and filled out. If a STEP entity is not recognized by the STEP processor then the StepData_UndefinedEntity object is created.

5.2.4. Initializing Open CASCADE objects

Each Open CASCADE object (including StepData_UndefinedEntity) is initialized by its parameters with the help of the appropriate RW - class. If some entity has another entity as its parameter, the object that represents the latter entity will be initialized immediately. All initialized objects are put into a special map to avoid repeated initialization.

5.2.5. Building a graph

The final phase is building a graph of references between entities. For each entity its RW-class is used to find entities referenced by this entity. Back references are built on the basis of direct references. In addition to explicit references defined in the STEP entities some additional (implicit) references are created for entities representing assembly structures (links from assemblies to their components).

5.3. How to add a new entity in scope of the STEP processor

If it is necessary to read and translate a new entity by the STEP processor the Reader and Actor scope should be enhanced. Note that some actions to be made for adding a new type are different for simple and complex types.

The following steps should be taken:

1. Create a CDL class representing a new entity. This can be the Stepxxx_NewEntity class where xxx currently are the following:

   Basic

   Geom

   Shape

Visual

Repr

AP214

AP203

Each field of a STEP entity should be represented by a corresponding field of this class. The class should have methods for initializing, setting and obtaining fields and it should also have the default constructor.

2. Create the RWStepxxx_RWNewEntity class with a default constructor and methods ReadStep(), WriteStep() and if the entity references other entities, then method Share().

3. Update file StepAP214_Protocol.cxx. In the constructor StepAP214_Protocol:: StepAP214_Protocol() add the new type to the map of registered types and associate the unique integer identifier with this type.

4. Update file RWStepAP214_ReadWriteModule.cxx. The changes should be the following:

For simple types:

- Add a static object of class TCollection_AsciiString with name Reco_NewEntity and initialize it with a string containing the STEP type.
- In constructor RWStepAP214_ReadWriteModule::RWStepAP214_ReadWriteModule() add this object onto the list with the unique integer identifier of the new entity type.
- In function RWStepAP214_ReadWriteModule::StepType() add a new C++ case operator for this identifier.

For complex types:

- In the method RWStepAP214_ReadWriteModule::CaseStep() add a code for recognition the new entity type returning its unique integer identifier.
- In the method RWStepAP214_ReadWriteModule::IsComplex() return True for this type.
- In the method RWStepAP214_ReadWriteModule::ComplexType() fill the list of subtypes composing this complex type.

For both simple and complex types:

- In function RWStepAP214_ReadWriteModule::ReadStep() add a new C++ case operator for the new identifier and call the RWStepxxx_RWNewEntity class, method ReadStep to initialize the new class.

5. Update file RWStepAP214_GeneralModule.cxx. Add new C++ case operators to functions NewVoid() and FillSharedCase(), and in the method CategoryNumber() add a line defining a category of the new type.

6. Enhance the STEPControl_ActorRead class (methods Recognize() and Transfer()), or class(es) translating some entities, to translate the new entity into an Open CASCADE shape.

5.4. Physical file writing

Physical file writing consists of the following steps:

1. Building a references graph
2. Transferring data from a model to a sequence of strings
3. Writing the sequence of strings into the file

5.4.1. Building a references graph

Physical writing starts when STEP model, which was either loaded from a STEP file or created from Open CASCADE shape with the help of translator, is available together with corresponding graph of references.

During this step the graph of references can be recomputed.

5.4.2. Transferring data from the model to a sequence of strings

For each representing entity from the model a corresponding RW - class is called. RW - class performs the writing of data that is contained in the representing class into an intermediate data structure. The mentioned structure is a sequence of strings in memory.

5.4.3. Writing the sequence of strings into the file

The sequence of strings is written into the file. This is the last phase of physical STEP writing.

5.5. How to add a new entity to write in the STEP file.

If it is necessary to write and translate an Open CASCADE shape into a new entity by the STEP processor the Writer and Actor scope should be enhanced.

For a description of steps, which should be taken for adding a new entity type to the STEP processor, see the previous chapter "Physical file reading". Then, enhance the STEPControl_ActorWrite class i.e. methods Recognize() and Transfer(), or other classes from TopoDSToStep, to translate the Open CASCADE shape into a new STEP entity.

6. Using DRAW

6.1. DRAW STEP Commands Overview

TKXSDRAW toolkit provides commands for testing XSTEP interfaces interactively in the DRAW environment. It provides an additional set of DRAW commands specific for data exchange tasks, which allows loading and writing data files and an analysis of the resulting data structures and shapes.

This section is divided into five parts. Two of them deal with reading and writing a STEP file and are specific for the STEP processor. The first and the forth parts describe some general tools for setting parameters and analyzing the data. Most of them are independent of the norm being tested. Additionally, a table of mentioned DRAW commands is provided.

NOTE

In the description of commands, square brackets ([]) are used to indicate optional parameters. Parameters given in the angle brackets (<>) and sharps (#) are to be substituted by an appropriate value. When several exclusive variants are possible, a vertical dash (|) is used.

6.2. Setting the interface parameters

A set of parameters for importing and exporting STEP data is defined in the XSTEP resource file. In XSDRAW, these parameters can be viewed or changed using the command

```
Draw:> param [<parameter_name> [<value>]]
```

Command param with no arguments gives a list of all parameters with their values. When the argument <parameter_name> is specified, information about this parameter is printed (current value and short description).

The third argument is used to set a new value of the given parameter. The result of the setting is printed immediately.

During all interface operations, the protocol of the process (fail and warning messages, mapping of loaded entities into Open CASCADE shapes etc.) can be output to the trace file. Two parameters are defined in the DRAW session: trace level (integer value from 0 to 9, default is 0), and trace file (default is standard output).

Command xtrace is intended to view and change these parameters:

```
Draw:> xtrace
```

- prints current settings (e.g.: `Level=1 - Standard Output');

```
Draw:> xtrace #
```

- sets trace level to the value #;

```
Draw:> xtrace tracefile.log
```

- sets the trace file as tracefile.log; and

```
Draw:> xtrace .
```

- directs all messages to the standard output.

6.3. Reading a STEP file

For a description of parameters used in reading a STEP file refer to 2.3.3

For reading a STEP file, the following parameters are defined (see above, the command param):

Description	Name	Values	Meaning
Precision for input entities	read.precision.mode	0 or 1	If 0 (File), precision of the input STEP file will be used for the loaded shapes If 1 (Session), the following parameter will be used as the precision value
	read.precision.val	real	Value of precision (used if the previous parameter is 1)
Surface curves	read.surfacecurve.mode	0 or 3	Defines a preferable way of representing surface curves (2d or 3d representation). If 0, no preference.
Maximal tolerance	read.maxprecision.mode	0 or 1	If 1, maximum tolerance is used as a rigid limit If 0, maximum tolerance is used as a limit but can be exceeded by some algorithms
	read.maxprecision.val	real	Value of maximum precision

It is possible either only to load a STEP file into memory (i.e. fill the *InterfaceModel* with data from the file), or to read it (i.e. load and convert all entities to Open CASCADE shapes).

Loading is done by the command

```
Draw:> xload <file_name>
```

Once the file is loaded, it is possible to investigate the structure of the loaded data. To find out how you do it, look in the beginning of the analysis subsection.

Reading a STEP file is done by the command

```
Draw:> stepread <file_name> <result_shape_name> [selection]
```

Here a dot can be used instead of a filename if the file is already loaded by xload or stepread.

The optional selection (see below for a description of selections) specifies a set of entities to be translated. If an asterisk `*' is given, all transferable roots are translated. If a selection is not given, the user is prompted to define a scope of transfer interactively:

N	Mode	Description
0	End	Finish transfer and exit **stepread**
1	root with rank 1	Transfer first root
2	root by its rank	Transfer root specified by its rank
3	One entity	Transfer entity with a number provided by the user
4	Selection	Transfer only entities contained in selection

* root is an entity in the STEP file which is not referenced by another entities

Second parameter of the stepread command defines the name of the loaded shape.

During the STEP translation, a map of correspondence between STEP entities and Open CASCADE shapes is created.

To get information on the result of translation of a given STEP entity use the command

```
Draw:> tpent #
```

is used.

To create an Open CASCADE shape, corresponding to a STEP entity, use the command

```
Draw:> tpdraw #
```

is used.

To get the number of a STEP entity, corresponding to an Open CASCADE shape, use the command

```
Draw:> fromshape <shape_name>
```

is used.

To clear the map of correspondences between STEP entities and Open CASCADE shapes the command

```
Draw:> tpclear
```

is used.

6.4. Analyzing the data transferred

The procedure of analysis of data import can be divided into two stages:

1. to check the file contents,
2. to estimate the translation results (conversion and validated ratios).

6.4.1. Checking file contents

General statistics on the loaded data can be obtained by using the command

```
Draw:> data <symbol>
```

Information printed by this command depends on the symbol specified:

Symbol	Output
g	Prints the information contained in the header of the file
c or **f**	Prints messages generated during the loading of the STEP file (when the procedure of the integrity of the loaded data check is performed) and the resulting statistics (**f** works only with fails while **c** with both fail and warning messages)
t	The same as **c** or **f**, with a list of failed or warned entities
m or **l**	The same as **t** but also prints a status for each entity
e	Lists all entities of the model with their numbers, types, validity status etc.
R	The same as **e** but lists only root entities

There is a set of special objects, which can be used to operate with a loaded model. They can be of the following types:

Special object type	Operation
Selection	Filters - allow to select subsets of entities of the loaded model
Counter	Calculate some statistics on the model data

A list of these objects defined in the current session can be printed in DRAW by command

```
Draw:> listitems
```

Command

```
Draw:> givelist <selection_name>
```

prints a list of a subset of loaded entities defined by the selection argument:

Selection	**Description**
xst-model-all	all entities of the model
xst-model-roots	all roots
xst-pointed	(Interactively) pointed entities (not used in DRAW)
xst-transferrable-all	all transferable (recognized) entities
xst-transferrable-roots	Transferable roots

The command listtypes gives a list of entity types, which were encountered in the last loaded file (with a number of STEP entities of each type).

The list cannot be shown for all entities but for a subset of them. This subset is defined by an optional selection argument (for the list of possible values for STEP, see the table above).

Two commands are used to calculate statistics on the entities in the model:

```
Draw:> count <counter> [<selection>]
Draw:> listcount <counter> [<selection>]
```

The former only prints a count of entities while the latter also gives a list of them.

The optional selection argument, if specified, defines a subset of entities, which are to be taken into account. The first argument should be one of the currently defined counters:

Counter	**Operation**
xst-types	Calculates how many entities of each Open CASCADE type exist
step214-types	Calculates how many entities of each STEP type exist

Entities in the STEP file are numbered in the succeeding order. An entity can be identified either by its number or by its label. Label is the letter `#' followed by the rank. To get a label for an entity with a known number, command

```
Draw:> elab #
```

can be used.

In the same way, command

```
Draw:> enum #
```

prints a number for the entity with a given label.

The contents of a STEP entity can be obtained by command

```
Draw:> entity # <level_of_information>
```

The list of entities referenced by a given entity and the list of entities referencing to it can be obtained by command

```
Draw:> estat #
```

A STEP assembly can be printed as a tree using the following DRAW command:

```
Draw:> dumpassembly
```

Information about product names, next_assembly_usage_occurence, shape_definition_representation, context_dependent_shape_representation or mapped_item entities that are involved into the assembly structure will be printed.

6.4.2. Estimating the results of reading STEP

All the following commands are available only after data is converted into Open CASCADE shapes (i.e. after command 214read).

Command

```
Draw:> tpstat [*|?]<symbol> [<selection>]
```

is provided to get all statistics on the last transfer, including a list of transferred entities with mapping from STEP to Open CASCADE types, as well as fail and warning messages. The parameter symbol defines what information will be printed:

Symbol	Output
g	General statistics (a list of results and messages)
c	Count of all warning and fail messages
C	List of all warning and fail messages
f	Count of all fail messages
F	List of all fail messages
n	List of all transferred roots
s	The same, with types of source entity and the type of result
b	The same, with messages
t	Count of roots for geometrical types
r	Count of roots for topological types
l	The same, with the type of the source entity

The sign `*' before parameters n, s, b, t, r makes it work on all entities (not only on roots). The sign '?' before n, s, b, t limits the scope of information to invalid entities.

Optional argument <selection> can limit the action of the command to the selection, not to all entities.

To get help, run this command without arguments.

Example: The following command gives statistics on the result of translation of different types of entities (taking check messages into account) and calculates summary translation ratios.

```
Draw:> tpstat *l
```

To get information on Open CASCADE shape contents the command

```
Draw:> statshape <shape_name>
```

is used.

It outputs the number of each kind of shapes (vertex, edge, wire, etc.) in the shape and some geometrical data (number of C0 surfaces, curves, indirect surfaces, etc.).

Note

The number of faces is returned as a number of references. To obtain the number of single instances, the standard command (from TTOPOLOGY executable) nbshapes can be used.

To analyze the internal validity of the shape, command

```
Draw:> checkbrep <shape_name> <expurged_shape_name>
```

is used. It checks shape geometry and topology for different cases of inconsistency, like self-intersecting wires or wrong orientation of trimming contours. If an error is found, it copies bad parts of the shape with the names "expurged_subshape_name _#" and generates an appropriate message. If possible this command also tries to find STEP entities the Open CASCADE shape was produced from.

<expurged_shape_name> will contain the original shape without invalid subshapes.

To get information on tolerances of the shape the command

```
Draw:> tolerance <shape_name> [<min> [<max>] [<symbol>]]
```

is used. It outputs maximum, average and minimum values of tolerances for each kind of subshapes having tolerances and for the whole shape in general.

When specifying <min> and <max> arguments this command saves shapes with tolerances in the range [min, max] with names <shape_name>_... and gives their total number.

<Symbol> is used for specifying the kind of sub-shapes to analyze: v - for vertices, e - for edges, f - for faces, c - for shells and faces.

6.5. Writing a STEP file

For writing shapes to a STEP file, the following parameters are defined (see above, the command param):

Description	**Name**	**Values**	**Meaning**
Uncertainty for resulting entities	Write.precision.mode	-1, 0, 1 or 2	If -1 the uncertainty value is set to the minimal tolerance of CASCADE subshapes. If 0 the uncertainty value is set to the average tolerance of CASCADE subshapes. If 1 the uncertainty value is set to the maximal tolerance of CASCADE subshapes. If 2 the uncertainty value is set to write.precision.val
Value of uncertainty	Write.precision.val	real	Value of uncertainty (used if previous parameter is 2)

Several shapes can be written in one file. To start writing a new file, enter command

```
Draw:> newmodel
```

Actually, command newmodel will clear the InterfaceModel to empty it, and the next command will convert the specified shape to STEP entities and add them to the InterfaceModel:

```
Draw:> stepwrite <mode> <shape_name> [<file_name>]
```

The available modes are following:

a - "as is" mode

m - manifold_solid_brep or brep_with_voids

f - faceted_brep

w - geometric_curve_set

s - shell_based_surface_model

For further information, see "Performing the Open CASCADE shape translation".

After a successful translation, if <file_name> parameter is not specified, the procedure asks you whether to write a STEP model in the file or not:

execution status : 1

Mode (0 end, 1 file) :

It is necessary to call command

```
newmodel
```

in order to perform a new translation of the next Open CASCADE shape.

6.6. Index of useful XSDRAW commands

Command	**Description**
	Setting general parameters
xtrace [#|<file>|.]	View and set parameters of the trace file
param [<param> [<val>]]	View and set parameters of transfer
	Reading and writing a STEP file
xload <file>	Load a file into memory
stepread {<file>|.} <name>	Convert data from the file into shapes
newmodel	Creates an empty Interface model in memory
stepwrite <mode> <shape>	Prepares shapes for writing and writes data to the file
	Checking the results of the load procedure
data <symbol>	Get the statistics on the loaded file
entity {n|#n}	Dump data loaded for a particular entity
estatus {n|#n}	Get the load status of the entity and its references
	Checking the results of conversion
tpstat [*| ?]<symbol>	Get statistics on the converted entities
fromshape <shape>	Get the source STEP entity for the specified shape
	Analyzing the loaded shapes
checkbrep <shape> <expurged_shape>	Check the shape for internal errors
statshape <shape>	Get statistics on the shape
tolerance <shape>	Calculate tolerances for the given shape

7. Reading from and writing to XDE

The STEPCAFControl package (TKXDESTEP toolkit) provides tools to read and write STEP files to and from XDE format (see XDE User's Guide).

In addition to the translation of shapes implemented in basic translator, it provides the following:

- STEP assemblies, read as Open CASCADE compounds by basic translator, are translated to XDE assemblies
- Names of products are translated and assigned to assembly components and instances in XDE
- STEP external references are recognized and translated (if external documents are STEP files)
- Colors, layers, materials and validation properties assigned to parts or subparts are translated
- STEP dimensional tolerances are translated

7.1. Description of the process

7.1.1. Loading a STEP file

Before performing any other operation, you must load a STEP file with:

```
STEPCAFControl_Reader reader(XSDRAW::Session(), Standard_False);
IFSelect_ReturnStatus stat = reader.ReadFile("filename.stp");
```

Loading the file only memorizes the data, it does not translate it.

7.1.2. Checking the loaded STEP file

This step is not obligatory. See a description of this step in paragraph 2.3.2.

7.1.3. Setting the parameters for translation to XDE

See a description of this step in paragraph 2.3.3.

In addition, the following parameters can be set for XDE translation of attributes:

• Parameter for transferring colors:

```
reader.SetColorMode(mode);
// mode can be Standard_True or Standard_False
```

• Parameter for transferring names:

```
reader.SetNameMode(mode);
// mode can be Standard_True or Standard_False
```

7.1.4. Performing the translation of a STEP file to XDE

The following function performs a translation of the whole document:

```
Standard_Boolean ok = reader.Transfer(doc);
```

where "doc" is a variable which contains a handle to the output document and should have a type `Handle(TDocStd_Document)`.

7.1.5. Initializing the process of translation from XDE to STEP

Here is how to initialize the process:

```
STEPCAFControl_Writer aWriter(XSDRAW::Session(),Standard_False);
```

7.1.6. Setting the parameters for translation from XDE to STEP

The following parameters can be set for a translation of attributes to STEP:

• Parameter for transferring colors:

```
aWriter.SetColorMode(mode);
// mode can be Standard_True or Standard_False
```

• Parameter for transferring names:

```
aWriter.SetNameMode(mode);
// mode can be Standard_True or Standard_False
```

7.1.7. Performing the translation of an XDE document to STEP

You can perform the translation of document by calling the function:

```
IFSelect_ReturnStatus aRetSt = aWriter.Transfer(doc);
```

where "doc" is a variable, which contains a handle to the input document for transferring and should have a type `Handle(TDocStd_Document)`.

7.1.8. Writing a STEP file

Write a STEP file with:

```
IFSelect_ReturnStatus statw = aWriter.WriteFile("filename.stp");
```

or

```
IFSelect_ReturnStatus statw = writer.WriteFile (S);
```

where S is OStream