Open CASCADE Technology
7.2.0

STEP processor

August 30, 2017

Contents

1 Introduction

STEP is more and more widely used to exchange data between various software, involved in CAD, PDM, Analysis, etc... STEP is far more than an "exchange standard" : it provides a technology and a set of methodologies to describe the data to exchange in a modular and upgradeable way. Regarding OCCT, this mostly applies to CAD data but it is not a limitation, other kinds of data for specific applications can be addressed too.
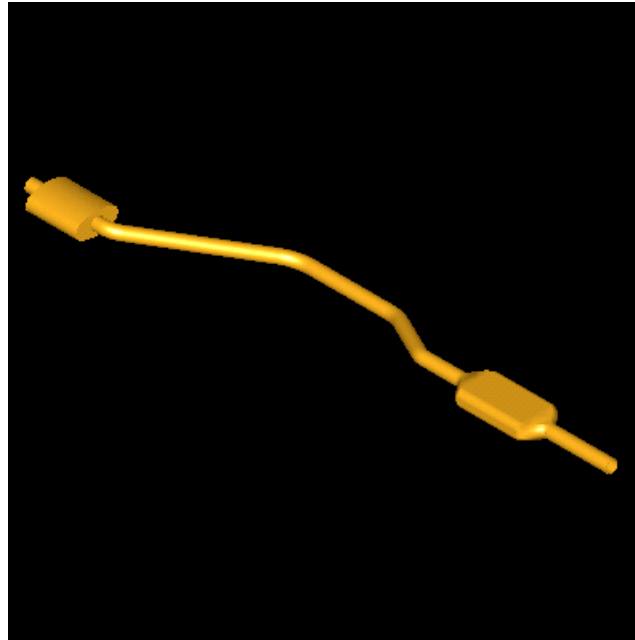
Figure 1: Image imported from STEP

Open Cascade allows its users to employ STEP in the following domains:

- Exchange of data for technical applications, following the state-of-the-art definitions and rules;

- Extension of case coverage, according to specific needs or to the evolution of general business uses;

- Expertise in data architecture of an application, to get experience from STEP definitions and make easier the mapping to them, for a better interoperability with outer world.

This manual is intended to provide technical documentation on the Open CASCADE Technology (**OCCT**) STEP processor and to help Open CASCADE Technology users with the use of the STEP processor (to read and write STEP files).

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.

Shape Healing toolkit provides tools to heal various problems, which may be encountered in translated shapes, and to make them valid in Open CASCADE. The Shape Healing is smoothly connected to STEP translator using the same API, only the names of API packages change.

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.

See also our `E-learning & Training` offerings.

1.1 STEP Exchanges in Open Cascade technology

Beyond the upper level API, which is fitted for an easy end-use, the STEP exchange functions enter in the general frame of Exchanges in Open Cascade, adapted for STEP:

- Specific packages for Data definition and checking;

- Physical Access supported by Drivers (Part 21 file access is embedded);

- Conversion to/from Open Cascade or applicative data supported by drivers (OCC-BREP and XDE ard basically provided);

- Tools for analysis, filtering, etc... including DRAW commands.

These modules share common architecture and capabilities with other exchange modules of Open Cascade, like Shape Healing. Also, built-in Viewer and Converter (as Plugin for Netscape, Internet Explorer ..), are based on the same technology.

In addition, Open Cascade provides tools to process models described using STEP: to reflect EXPRESS descriptions, to read, write and check data, to analyze the whole models ... Their key features are:

- Modularity by sets of data types, which can be hierarchized to reflect the original modularity describing the resources and application protocols;

- Implementation as C++ classes, providing comprehensive access to their members;

- Early binding is basically used, providing good performance, easy installation and use as well as the capability to support non-compiled descriptions.

This provides a natural way to deal with non-supported protocols when they share common definitions, as for geometry, which can then be exploited. The common frame, as the already supported data types, give a good foundation to go towards new uses of STEP, either on data definition (protocols from ISO or from industrial consortia) or on mapping with applicative data.

1.2 STEP Interface

The STEP interface reads STEP files produced in accordance with STEP Application Protocol 214 (Conformance Class 2 both CD and DIS versions of schema) and translates them to Open CASCADE Technology models. STEP Application Protocol 203 and some parts of AP242 are also supported.

The STEP interface also translates OCCT models to STEP files. STEP files that are produced by this interface conform to STEP AP 203 or AP 214 (Conformance Class 2, either CD or DIS version of the schema) depending on the user's option.

Basic interface reads and writes geometrical, topological STEP data and assembly structures.

The interface is able to translate one entity, a group of entities or a whole file.

Other kinds of data such as colors, validation properties, layers, GD&T, names and the structure of assemblies can be read or written with the help of XDE tools: *STEPCAFControl_Reader* and *STEPCAFControl_Writer*.

To choose a translation mode when exporting to a STEP format, use *STEPControl_STEPModelType*.

There is a set of parameters that concern the translation and can be set before the beginning of the translation.

Please, note:

- a STEP model is a STEP file that has been loaded into memory;

- all references to shapes indicate OCCT shapes unless otherwise explicitly stated;

- a root entity is the highest level entity of any given type, i.e. an entity that is not referenced by any other one.

2   Reading STEP

2.1   Procedure

You can translate a STEP file into an OCCT 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 OCCT shapes, and secondly the OCCT 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 OCCT 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 Technology. 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 Technology 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.

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 Mapping STEP entities to Open CASCADE Technology 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 Technology 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.

If *failsonly* is true only fail messages are displayed. 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 OCCT 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. By default it is 0.0001, but can be 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 ..
```

By default this value is 0.0001.

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 ..;
```

Default 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 and 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 Technology installation, by the path *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.

xstep.cascade.unit

This parameter defines units to which a shape should be converted when translated from IGES or STEP to C↩
ASCADE. 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 recog-
  nized 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 incor-
  rect 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) – translates all products;

- 2 (design) – translates only products that have *PRODUCT_DEFINITION_CONTEXT* with field *life_cycle_*↩
  stage set to 'design';

- 3 (analysis) – translates 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 *SHAP↩E_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:

- *Standard_Boolean ok = reader.TransferRoot(rank)* – translates a root entity identified by its rank;

- *Standard_Boolean ok = reader.TransferOne(rank)* – translates an entity identified by its rank;

- *Standard_Integer num = reader.TransferList(list)* – translates a list of entities in one operation (this method returns the number of successful translations);

- *Standard_Integer NbRoots = reader.NbRootsForTransfer()* and *Standard_Integer num = reader.Transfer↩ Roots()* – translate all transferable roots.

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.

- *Standard_Integer num = reader.NbShapes()* – gets the number of shapes recorded in the result;

- *TopoDS_Shape shape = reader.Shape(rank)* – gets the result identified by its rank, where rank is an integer between 1 and NbShapes;

- *TopoDS_Shape shape = reader.Shape()* – gets the first result of translation;

- *TopoDS_Shape shape = reader.OneShape()* – gets all 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.

Clearing the accumulation of results

If several individual translations follow each other, the results give a list that can be purged with *reader.Clear↩ Shapes()*, 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.

The 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();
```

List 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 as well.

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 Technology shapes

Tables given in this paragraph show the mapping of STEP entities to OCCT 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 OCCT 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	A *TopoDS_Compound* for assemblies, a CASCADE shape corresponding to the component type of for components,	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	A TopoDS_Compound for assemblies, a CASCADE shape corresponding to the component type for components.	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.

STEP entity type	CASCADE shape	Comments
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

Topology	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.

Geometry	STEP entity type	CASCADE object	Comments
Points	cartesian_point	Geom_CartesianPoint, Geom2d_CartesianPoint	
Directions	direction	Geom_Direction, Geom2d_↵Direction	
Vectors	vector	Geom_VectorWithMagnitude, Geom2d_VectorWith↵Magnitude	
Placements	axis1_placement	Geom_Axis1Placement	
	axis2_placement_2d	Geom2d_AxisPlacement	
	axis2_placement_3d	Geom_Axis2Placement	
Curves	circle	Geom_Circle, Geom2d_↵Circle, Geom2d_Bspline↵Curve	Circle is translated into *Geom2d_BSplineCurve* when it references the surface of revolution (spherical surface, conical surface, etc.)
	ellipse	Geom_Ellipse, Geom2d_↵Ellipse, Geom2d_Bspline↵Curve	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 or Geom2d_↵Curve	Depending on the type of the base curve
	offset_curve_3d	Geom_OffsetCurve	
	trimmed_curve	Geom_TrimmedCurve or Geom2d_BsplineCurve	Only trimmed_curves trimmed by parameters are translated. All *trimmed_curves* are converted to *Geom2d_BSpline↵Curve*.
	b_spline_curve	Geom_BsplineCurve or Geom2d_BsplineCurve	
	b_spline_curve_with_knots	Geom_BsplineCurve or Geom2d_BsplineCurve	
	bezier_curve	Geom_BsplineCurve or Geom2d_BsplineCurve	
	rational_b_spline_curve	Geom_BsplineCurve or Geom2d_BsplineCurve	
	uniform_curve	Geom_BsplineCurve or Geom2d_BsplineCurve	
	quasi_ uniform_curve	Geom_BsplineCurve or Geom2d_BsplineCurve	
	surface_curve	TopoDS_Edge	*surface_curve* defines geometrical support of an edge and its pcurves.

Geometry	STEP entity type	CASCADE object	Comments
	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_Rectangular↩ TrimmedSurface	
	spherical_surface	Geom_SphericalSurface	
	surface_of_linear_extrusion	Geom_SurfaceOfLinear↩ Extrusion	
	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 to OCCT 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 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* is used instead of uncertainty from a STEP file when parameter *read.precision.mode* is 1 (User).

2D (parametric) tolerances

- Package method *Precision::PConfusion()* is a value of *0.01∗Precision::Confusion()*. It is used to compare parametric bounds of curves.

- Methods *UResolution* and *VResolution (tolerance3d)* of the class *GeomAdaptor_Surface* or *BRepAdaptor↩ _Surface* return tolerance in parametric space of a surface computed from 3d tolerance. 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* return 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. *STEPControl_ActorRead::Transfer()*, function which performs the translation 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 OCCT 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:

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

- *ShapeFix_Wire::FixSelfIntersection()* if a pcurve of a self-intersecting edge is modified.

- vertices are constructed with Precision::Confusion(). It can be modified later by: *StepToTopoDS_↩ TranslateEdge ShapeFix::SameParameter() ShapeFix_Wire::FixSelfIntersection() ShapeFix_Wire::Fix↩ Lacking() 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 OCCT 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 OCCT shapes by use of classes from package *StepToTopoDS*.

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, OCCT 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 default tolerances are used when creating shapes and how they are updated during translation.

- *StepToTopoDS_TranslateVertex* constructs *TopoDS_Vertex* from a STEP *vertex_point* entity with *Precision↩ ::Confusion()*.

- *StepToTopoDS_TranslateVertexLoop* creates degenerated *TopoDS_Edge* in *TopoDS_Wire* with tolerance *Precision::Confusion()*. *TopoDS_Vertex* of a degenerated edge is constructed with the initial value of tolerance.

- *StepToTopoDS_TranslateEdge* constructs *TopoDS_Edge* 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.

- *StepToTopoDS_TranslateEdgeLoop* constructs *TopoDS_Edges* in *TopoDS_Wire* with help of class *StepTo↩ TopoDS_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).

- *StepToTopoDS_TranslatePolyLoop* constructs *TopoDS_Edges* in *TopoDS_Wire* with help of class *StepTo↩ TopoDS_TranslateEdge*. Their tolerances are not modified inside this method.

- *StepToTopoDS_TranslateFace* constructs *TopoDS_Face* 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*.

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

- *StepToTopoDS_TranslateCompositeCurve* constructs *TopoDS_Edges* in *TopoDS_Wire* with help of class *B↩ RepAPI_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 provided by calling method *ShapeFix_Wire::FixConnected()∗* with a precision equal to the initial value of tolerance.

- *StepToTopoDS_TranslateCurveBoundedSurface* constructs *TopoDS_Face* 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.

- *StepToTopoDS_Builder* 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.

- *StepToTopoDS_MakeTransformed* 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

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*.

ShapeFix_Wire::FixLacking()

This method is intended to detect gaps between pcurves of adjacent edges (with the precision of surface U$\hookleftarrow$ VResolution 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.

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

The following diagram illustrates the structure of calls in reading STEP. The highlighted classes are intended to translate geometry
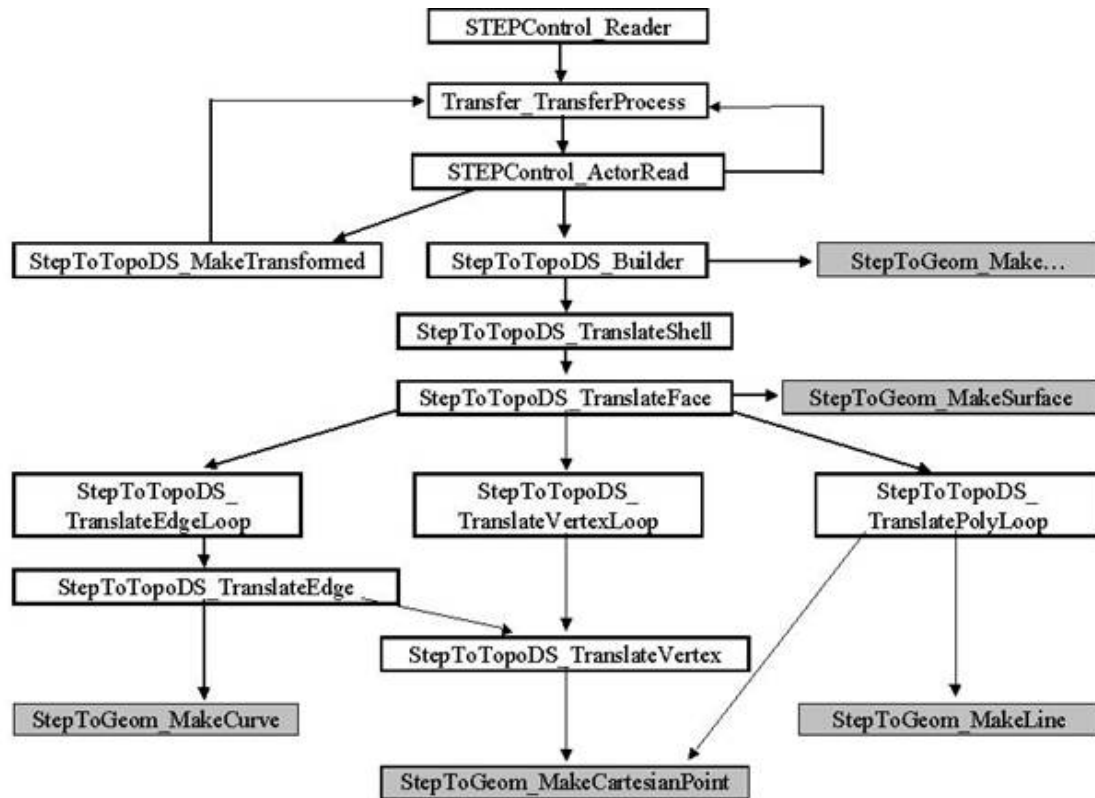
Figure 2: The structure of calls in reading STEP

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: ; NbRootsendl;

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

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

  . . .

}
```

3 Writing STEP

3.1 Procedure

You can translate OCCT 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 OCCT 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 ProST↩ EP 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 Mapping OCCT 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 OCCT-to-STEP translation.

write.precision.mode

writes the precision value.

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

- Average (0) : the uncertainty value is set to the average tolerance of an OCCT shape.

- Greatest (1) : the uncertainty value is set to the maximum tolerance of an OCCT 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 OCCT 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)

- 5 or *AP242DIS*: AP242, DIS version.

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;). 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: OCCT STEP translator (current OCCT 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 and 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 Technology 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.

write.step.vertex.mode

This parameter indicates which of free vertices writing mode is switch on.

- 0 (One Compound) : (default) All free vertices are united into one compound and exported in one SHAPE DEFINITION REPRESENTATION (vertex name and style are lost).

- 1 (Single Vertex) : Each vertex exported in its own SHAPE DEFINITION REPRESENTATION (vertex name and style are not lost, but size of STEP file increases).

Read this parameter with:

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

Modify this parameter with:

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

Default value is 0.

3.3.3 Performing the Open CASCADE Technology shape translation

An OCCT 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 C↩ASCADE 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 list shows which shapes can be translated in which mode:

- *STEP214Control_AsIs* – any OCCT 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_ShellBasedSurfaceModel* – *TopoDS_Solid, TopoDS_Shell, TopoDS_Face* and *TopoDS_↩ Compound* (if it contains all mentioned shapes)

- *STEP214Control_GeometricCurveSet* – any OCCT 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 OCCT shape cannot be translated according to its mode the result of translation is void.

```
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 Technology shapes to STEP entities

Only STEP entities that have a corresponding OCCT 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 *TopoD↩ S_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.

CASCADE shape	STEP entity	Comments
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_As↵Is* and *TopoDS_Compound* consists only of *TopoDS_Solids*.
	shell_based_surface_model	If the write mode is *STEP214Control_AsIs* and *TopoDS_Compound* consists of *Topo↵DS_Solids*, *TopoDS_Shells* and *TopoDS_↵Faces*.

CASCADE shape	STEP entity	Comments
	geometric_curve_set	If the write mode is *STEP214Control↩_AsIs* and *TopoDS_Compound* contains *TopoDS_Wires, TopoDS_Edges, Topo↩DS_Vertices*. If the write mode is not *STEP214Control_AsIs* or *STEP214↩Control_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_ShellBasedSurface↩Model*.
	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 it is 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

Geometry	CASCADE object	STEP entity	Comments
Points	Geom_CartesianPoint, Geom2d_CartesianPoint	cartesian_point	
	TColgp_Array1OfPnt, T↩Colgp_Array1OfPnt2d	polyline	
Placements	Geom_Axis1Plasement, Geom2d_AxisPlacement	axis1_placement	
	Geom_Axis2Placement	axis2_placement_3d	
Directions	Geom_Direction, Geom2d_↩Direction	direction	

Geometry	CASCADE object	STEP entity	Comments
Vectors	Geom_Vector, Geom2d_↩ Vector	vector	
Curves	Geom_Circle	circle	
	Geom2d_Circle	circle, rational_b_spline_↩ curve	
	Geom_Ellipse	Ellipse	
	Geom2d_Ellipse	Ellipse, rational_b_spline_↩ curve	
	Geom_Hyperbola, Geom2d↩ _Hyperbola	Hyperbola	
	Geom_Parabola, Geom2d_↩ Parabola	Parabola	
	Geom_BSplineCurve	b_spline_curve_with_knots or rational_b_spline_curve	*rational_b_spline_curve* is produced if *Geom_Bspline↩ Curve* is a rational BSpline
	Geom2d_BSplineCurve	b_spline_curve_with_knots or rational_b_spline_curve	*rational_b_spline_curve* is produced if *Geom2d_↩ BsplineCurve* is a rational BSpline
	Geom_BezierCurve	b_spline_curve_with_knots	
	Geom_Line or Geom2d_Line	Line	
Surfaces	Geom_Plane	Plane	
	Geom_OffsetSurface	offset_surface	
	Geom_ConicalSurface	conical_surface	
	Geom_CylindricalSurface	cylindrical_surface	
	Geom_OffsetSurface	offset_surface	
	Geom_Rectangular↩ TrimmedSurface	rectangular_trimmed_surface	
	Geom_SphericalSurface	spherical_surface	
	Geom_SurfaceOfLinear Extrusion	surface_of_linear_extrusion	
	Geom_SurfaceOf Revolution	surface_of_revolution	
	Geom_ToroidalSurface	toroidal_surface or degenerate_toroidal_surface	*degenerate_toroidal_surface* is produced if the minor radius is greater then the major one
	Geom_BezierSurface	b_spline_surface_with_knots	
	Geom_BsplineSurface	b_spline_surface_with_knots or rational_b_spline_surface	*rational_b_spline_surface* is produced if *Geom_B↩ SplineSurface* 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 Graph of calls

The following diagram illustrates the structure of calls in writing STEP. The highlighted classes are intended to translate geometry.
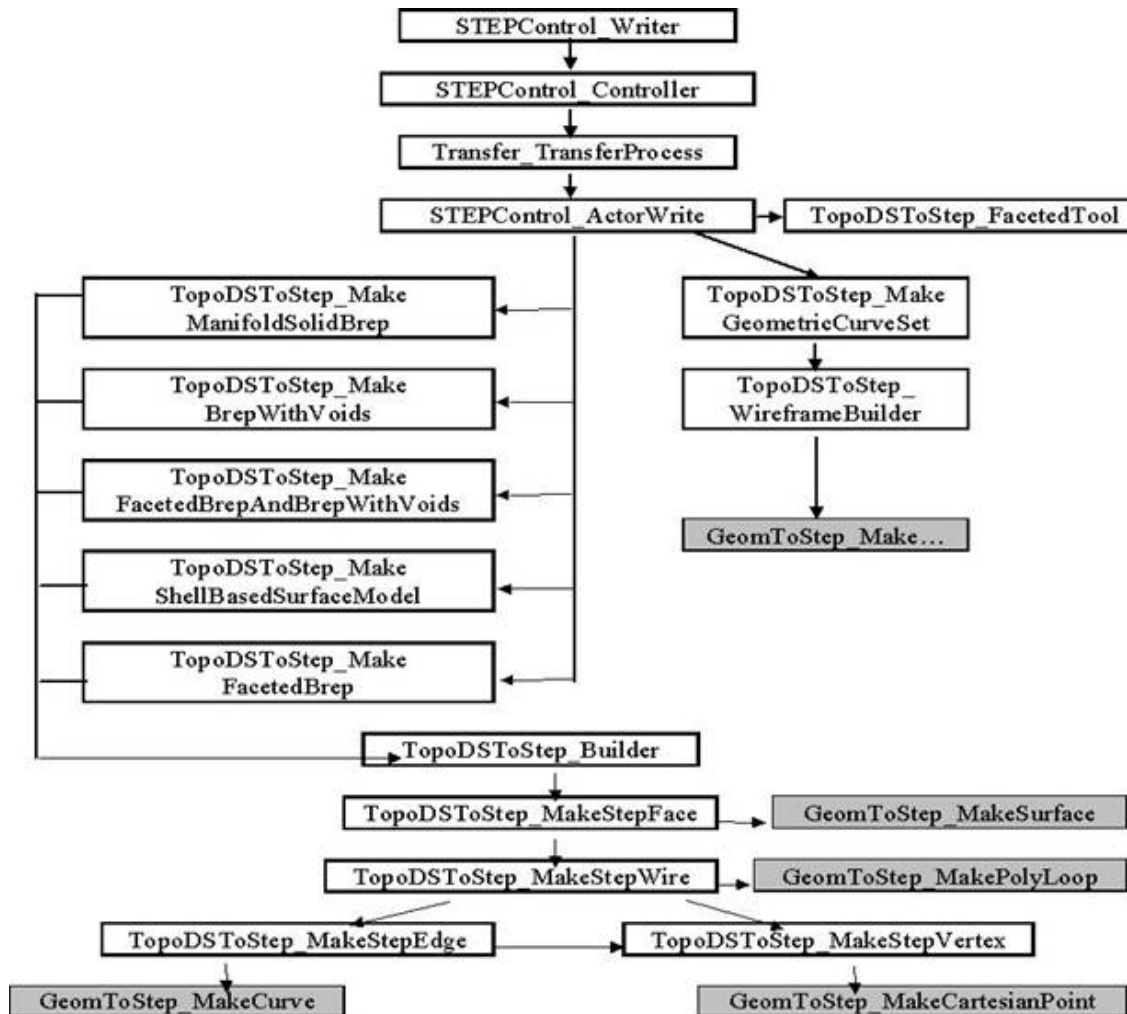
Figure 3: The structure of calls in writing STEP

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

}
```

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

4   Physical STEP file reading and writing

4.1   Architecture of STEP Read and Write classes

4.1.1   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 the 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 the 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 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.

4.1.2   Complex entities

EXPRESS schema allows multiple inheritance. Entities that are built on the basis of multiple inheritance are called complex entities. 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 C++ class for any possible complex type. Thus, only widespread complex entities have corresponding representing classes and RW-classes that are created manually beforehand.

4.2   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 OCCT objects representing STEP entities 4.↩ Initializing OCCT objects 5.Building a references graph

4.2.1   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.

4.2.2   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.

4.2.3 Creating empty Open CASCADE Technology objects that represent STEP entities

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

4.2.4 Initializing Open CASCADE Technology objects

Each OCCT object (including StepData_UndefinedEntity) is initialized by its parameters with the help of the appropriate RW-class. If an 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.

4.2.5 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).

4.3 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:

- Create a class representing a new entity. This can be *Stepxxx_NewEntity* class where xxx can be one of the following:

  - Basic
  - Geom
  - Shape
  - Visual
  - Repr
  - AP214
  - AP203
  - AP242

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.

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

- 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.

- 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 *WStepAP214_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.

- 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.

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

4.4 Physical file writing

Physical file writing consists of the following steps:

1. Building a references graph. Physical writing starts when STEP model, which was either loaded from a STEP file or created from OCCT shape with the help of translator, is available together with corresponding graph of references. During this step the graph of references can be recomputed.

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

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.

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

If it is necessary to write and translate an OCCT 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 Physical file reading. Then, enhance the *STEPControl_ActorWrite* class i.e. methods *Recognize()* and *Transfer()*, or other classes from *TopoDSToStep*, to translate the OCCT shape into a new STEP entity.

5 Using DRAW

5.1 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.

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.

5.2 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 OCCT 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*;

- *Draw:> xtrace.* – directs all messages to the standard output.

5.3 Reading a STEP file

For a description of parameters used in reading a STEP file refer to Setting the translation parameters section.

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.

Description	Name	Values	Meaning
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 OCCT 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 OCCT shapes is created.

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

```
Draw:> tpent #*.
```

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

```
Draw:> tpdraw #*.
```

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

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

To clear the map of correspondences between STEP entities and OCCT shapes use the command

```
Draw:> tpclear.
```

5.4 Analyzing the transferred data

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).

5.4.1   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:

- *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:

- Selection Filters – allow selecting subsets of entities of the loaded model;

- Counter – calculates 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:

- *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:

- *xst-types* – calculates how many entities of each OCCT 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.

- *Draw:> elab #* outputs a label for an entity with a known number.
- *Draw:> enum #* prints a number for the entity with a given label.
- *Draw:> entity # <level_of_information>* outputs the contents of a STEP entity.
- *Draw: estat #* outputs the list of entities referenced by a given entity and the list of entities referencing to it.
- *Draw: dumpassembly* prints a STEP assembly as a tree.

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.

5.4.2 Estimating the results of reading STEP

All the following commands are available only after data is converted into OCCT 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 OCCT types, as well as fail and warning messages. The parameter *<symbol>* defines what information will be printed:

- *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.

The command *Draw:> tpstat ∗1* gives statistics on the result of translation of different types of entities (taking check messages into account) and calculates summary translation ratios.

To get information on OCCT shape contents use command *Draw:> statshape <shape_name>* . 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.).

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, use command *Draw:> checkbrep <shape_name> <expurged_↩ shape_name>*. 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 OCCT shape was produced from.

<expurged_shape_name> will contain the original shape without invalid subshapes. To get information on tolerances of the shape use command *Draw:> tolerance <shape_name> [<min> [<max>] [<symbol>]]* . 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.

5.5 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 CASCA↩DE 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 CASCA↩DE 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 following modes are available :

- *a* – "as is" – the mode is selected automatically depending on the type & geometry of the shape;

- *m – manifold_solid_brep* or *brep_with_voids*

- *f – faceted_brep*

- *w – geometric_curve_set*

- *s – shell_based_surface_model*

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* to perform a new translation of the next OCCT shape.

6   Reading from and writing to STEP

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

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

- STEP assemblies, read as OCCT 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 Geometric Dimensions and Tolerances are translated;

- STEP Saved Views are translated.

6.1   Reading from STEP

Load 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.

Check the loaded STEP file

This step is not obligatory. See a description of this step in section Checking the STEP file.

Set parameters for translation to XDE

See a description of this step in section Setting the translation parameters.

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
  ```

Translate 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)*.

6.2 Attributes read from STEP

Colors

Colors are implemented in accordance with `Recommended practices for model styling and organization` sections 4 and 5.

The following attributes are imported from STEP file:

- colors linked to assemblies, solids, shells, faces/surfaces, wireframes, edges/curves and vertices/points;

- information about invisibility.

The following attributes are mentioned in the Recommended Practices, but not handled by OCCT:

- styling different sides of surfaces with different colors;

- transparency and reflectance for surfaces;

- curve styles;

- point markers.

Layers

Layers are implemented in accordance with `Recommended practices for model styling and organization` section 6. All layers are imported, but invisibility styles are skipped.

Materials

Materials are implemented in accordance with `Recommended practices for material identification and density` section 4. OCCT translator processes materials attached to solids in shape representations. The name, description and density (name and value) are imported for each material.

Validation properties

Validation properties are implemented in accordance with `Recommended practices for geometric and assembly validation properties` section 4 for AP214. OCCT processes several types of geometric validation properties for solids, shells and geometric sets:

- area;

- volume;

- centroid.

Geometric dimensions and tolerances

General types of STEP entities imported by OCCT are listed in the table below:

STEP entity	OCCT attribute
Dimensional_Size	XCAFDoc_Dimension
Dimensional_Location	XCAFDoc_Dimension
Dimensional_Size_With_Path	XCAFDoc_Dimension
Dimensional_Location_With_Path	XCAFDoc_Dimension
Angular_Size	XCAFDoc_Dimension

STEP entity	OCCT attribute
Angular_Location	XCAFDoc_Dimension
Geometric_Tolerance and subtypes	XCAFDoc_GeometricTolerance
Datum	XCAFDoc_Datum
Datum_Feature	XCAFDoc_Datum
Datum_Target	XCAFDoc_Datum

Processing of GD&T is realized in accordance with `Recommended practices for the Representation and Presentation of Product Manufacturing` for AP242. The general restriction is that OCCT S↵ TEP Reader imports GD&T assigned only to shapes (faces, edges, vertices, etc) or to shape groups from general shape model i.e. any constructive geometries are not translated as referenced shapes.

Dimensions

Dimensions are implemented according to section 5 of the latter document. Additionally to the reference shapes, the Reader imports from STEP file some auxiliary geometry for dimensional line building: connection points and line orientation, if exist.

The following values and modifiers described in sections 5.2 and 5.3 can be imported from STEP file:

- qualifiers (minimum, maximum and average);

- plus/minus bounds;

- value range;

- class of tolerance;

- text notes, attached to dimension value;

- dimension modifiers type 2 (Table 8);

- number of decimal places.

Datums

Datums are implemented in accordance with sections 6.5 and 6.6.1-6.6.2. Each datum can have one or several datum features (shapes from the model, to which the datum is linked) and datum targets (auxiliary geometry: point, line, rectangle, circle or area).

Tolerances

Tolerances are implemented in accordance with sections 6.7-6.9 with several restrictions.

Types of imported tolerances:

- simple tolerances (see Table 10);

- tolerance with modifiers (section 6.9.3);

- tolerance with maximum value (section 6.9.5);

- tolerance with datums (section 6.9.7 (simple datums and datum with modifiers) and 6.9.8 (common datums));

- superposition of the mentioned types.

Not all tolerance zones can be imported by OCCT STEP Reader, only the Tolerance Zones with associated symbols from *Table 11, Projected tolerance zone* (section 6.9.2.2) and *Runout zone* definition.

Presentations

Each semantic representation of GD&T (Dimension, Tolerance, Datum Feature or Datum Target) can have a presentation; its processing by OCCT is implemented in accordance with sections 7.3, 8 and 9.1-9.2. Presentations have several types:

- *Graphic Presentation* (polylines or tessellated wireframes) - partially implemented in OCCT;

- *Minimal Presentation* (position and orientation) - implemented in OCCT as a part of Graphic presentation;

- *Character-based Presentation* (3D Text with information about fonts, curve styles etc.) - not handled by OCCT.

Note, that separate Minimal presentation and Character-based Presentation are not described in any Recommended Practices, so there is no agreement about how such information should be saved in STEP file.

OCCT STEP Reader imports only Annotation Planes, outline/stroked Polylines and Tessellated wireframes, i.e. all styling information (color, curve style, etc.) and filled characters are missed.

OCCT STEP Reader also handles Annotations, linked directly to shapes (section 9.3.1), processing of these presentations is subject to the same restrictions as the processing of presentations, linked to GD&T semantic.

Geometric dimensions and tolerances AP214

Simple types of GD&T (Dimensions, Tolerances and Datums without presentations or any types of modifiers) are also handled in AP214. However, according to the Recommended Practices for the Representation and Presentation of Product Manufacturing, this implementation is obsolete.

Saved views

Saved views are implemented in accordance with `Recommended practices for the Representation and Presentation of Product Manufacturing` section 9.4.1-9.4.4. For each Saved View OCCT S←TEP Reader will retrieve the following attributes:

- set of displayed shape representations;

- set of displayed PMI presentations;

- projection point;

- view direction;

- up direction of view window;

- horizontal size of view window;

- vertical size of view window;

- zoom factor;

- clipping planes (single plane of combination of planes);

- front and back plane clipping.

6.3 Writing to STEP

The translation from XDE to STEP can be initialized as follows:

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

Set parameters for translation from XDE to STEP

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

- For transferring colors:

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

- For transferring names:

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

Translate 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)*.

Write 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*.

6.4 Attributes written to STEP

Colors

The following attributes are exported to STEP file:

- colors linked to assemblies, solids, shells, faces/surfaces, wireframes, edges/curves;

- information about visibility.

Restrictions:

- colors and visibility information for points is not exported by default, it is necessary to use *write.step.vertex.↩ mode* parameter;

- all colors are always applied to both sides of surfaces;

- all curves are exported with 'continuous' curve style.

Layers

All layers are exported, but invisibility styles can be connected only to shapes.

Materials

For solids with materials, the material is exported to STEP file (name, description and density (name and value)).

Validation properties

Geometric validation properties, such as volume, area and centroid, which are attached to shape, are exported to STEP file.

Geometric dimensions and tolerances

All entities, which can be imported from STEP, can be exported too. Please see the same item in section Reading from STEP to find more information.

Note: OCCT use AP214 by default, so for GD&T exporting AP242 should be set manually:

```
Interface_Static::SetCVal("write.step.schema", "AP242DIS"));
```

or

```
Interface_Static::SetIVal("write.step.schema", 5));
```

Saved views

Saved Views are not exported by OCCT.