Data Exchange

Extended Data Exchange (XDE) User's Guide

Version 6.3 / September 2008

NOTICE FOR USERS:

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

Tour Opus 12

77, Esplanade du Général de Gaulle

92914 PARIS LA DEFENSE

FRANCE

Contents

1. *Introduction*

1.1. Overview of the Extended Data Exchange (XDE)

This manual explains how to use the Extended Data Exchange (XDE). It provides basic documentation on setting up and using XDE. For advanced information on XDE and its applications, see our offerings on our web site at www.opencascade.com/support/training.html

Based on document architecture, XDE allows processing of various types of data to and from external files.

XDE is available for users of Open CASCADE on all supported platforms (Linux, Sun Solaris, Windows NT).

1.1.1. Prerequisite

The Extended Data Exchange (XDE) component requires Advanced Shape Healing for operation.

1.1.2. Environment variables

To use XDE you have to set the environment variables properly. Make sure that two important environment variables are set as follows:

- CSF_PluginDefaults points to sources of %CASROOT%/src/XCAFResources ($CASROOT/src/XCAFResources).
- CSF_XCAFDefaults points to sources of %CASROOT%/src/XCAFResources ($CASROOT/src/XCAFResources).

1.1.3. Basic terms

For better understanding of XDE, certain key terms are defined:

- **Shape** A (simple) shape is a standalone shape, which does not belong to the assembly structure.
- **Instance** An instance (of a shape) is a replication of another shape with a location that can be the same location or a different one.

Assembly An assembly defines a construction that is either a root or a sub-assembly.

1.1.4. XDE Data Types

The following types of data are currently supported:

- assemblies
- validation properties
- names
- colors
- layers

It is also possible to add new types of data by using tools as prototypes. This makes XDE a basically extensible framework.

In addition, XDE provides reading and writing tools to read and write the data supported by STEP and IGES files.

1.1.5. XDE Organization

The basis of XDE, called XCAF, is a framework based on OCAF (Open CASCADE Application Framework) and is intended to be used with assemblies and with various kinds of attached data (attributes). Attributes can be Individual attributes for a shape, specifying some characteristics of a shape, or they can be Grouping attributes, specifying that a shape belongs to a given group whose definition is specified apart from the shapes.

XDE works in an OCAF document with a specific organization defined in a dedicated XCAF module. This organization is used by various functions of XDE to exchange standardized data other than shapes and geometry.

The Assembly Structure and attributes assigned to shapes are stored in the OCAF tree. It is possible to obtain TopoDS representation for each level of the assembly in the form of TopoDS_Compound or TopoDS_Shape using the API.

Basic elements used by XDE are introduced in the XCAF sub-module by the package XCAFDoc. These elements consist in descriptions of commonly used data structures (apart from the shapes themselves) in normalized data exchanges. They are not attached to specific applications and do not bring specific semantics, but are structured according to the use and needs of data exchanges.

The Document used by XDE usually starts as a TDocStd_Document.

1.1.6. Assemblies

XDE supports assemblies by separating shape definitions and their locations. Shapes are simple OCAF objects without a location definition. An assembly consists of several components. Each of these components references one and the same specified shape with different locations. All this provides an increased flexibility in working on multi-level assemblies.

For example, a mechanical assembly can be defined as follows:

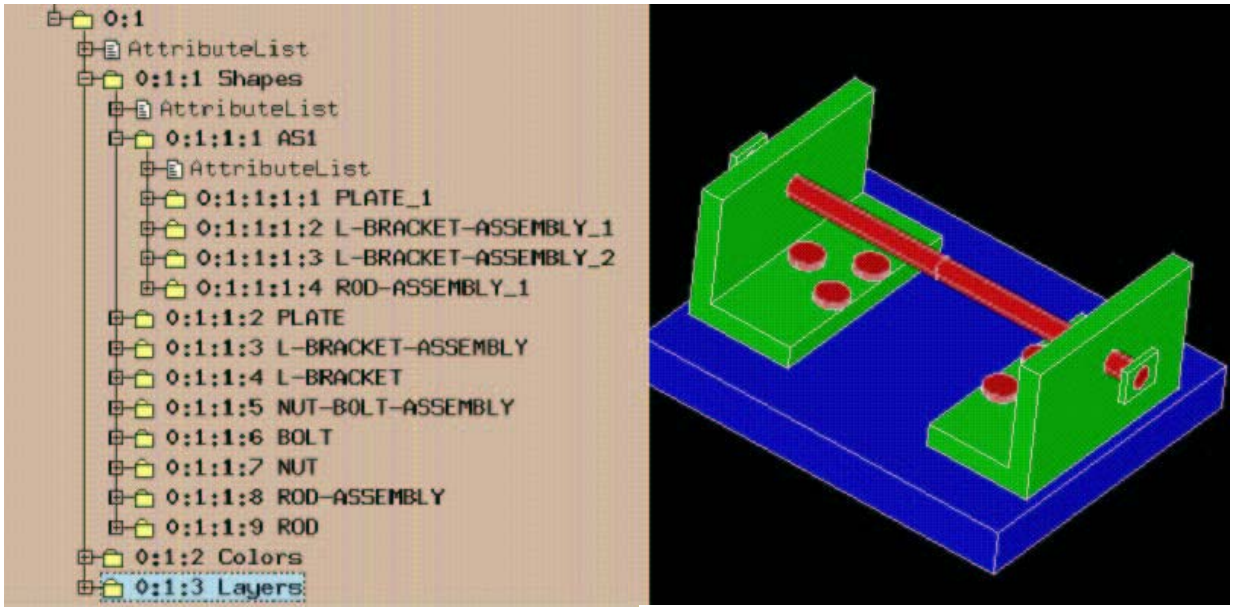

Figure 1. Assembly Description and View

XDE defines the specific organization of the assembly content. Shapes are stored on sub-labels of label 0:1:1. There can be one or more roots (called free shapes) whether they are true trees or simple shapes. A shape can be considered to be an Assembly (such as AS1 under 0:1:1:1 in Figure1) if it is defined with Components (sub-shapes, located or not).

XCAFDoc_ShapeTool is a tool that allows you to manage the Shape section of the XCAF document. This tool is implemented as an attribute and located at the root label of the shape section.

1.1.7. Validation Properties

Validation properties are geometric characteristics of Shapes (volume, centroid, surface area) written to STEP files by the sending system. These characteristics are read by the receiving system to validate the quality of the translation. This is done by comparing the values computed by the original system with the same values computed by the receiving system on the resulting model.

Advanced Data Exchange supports both reading and writing of validation properties, and provides a tool to check them.

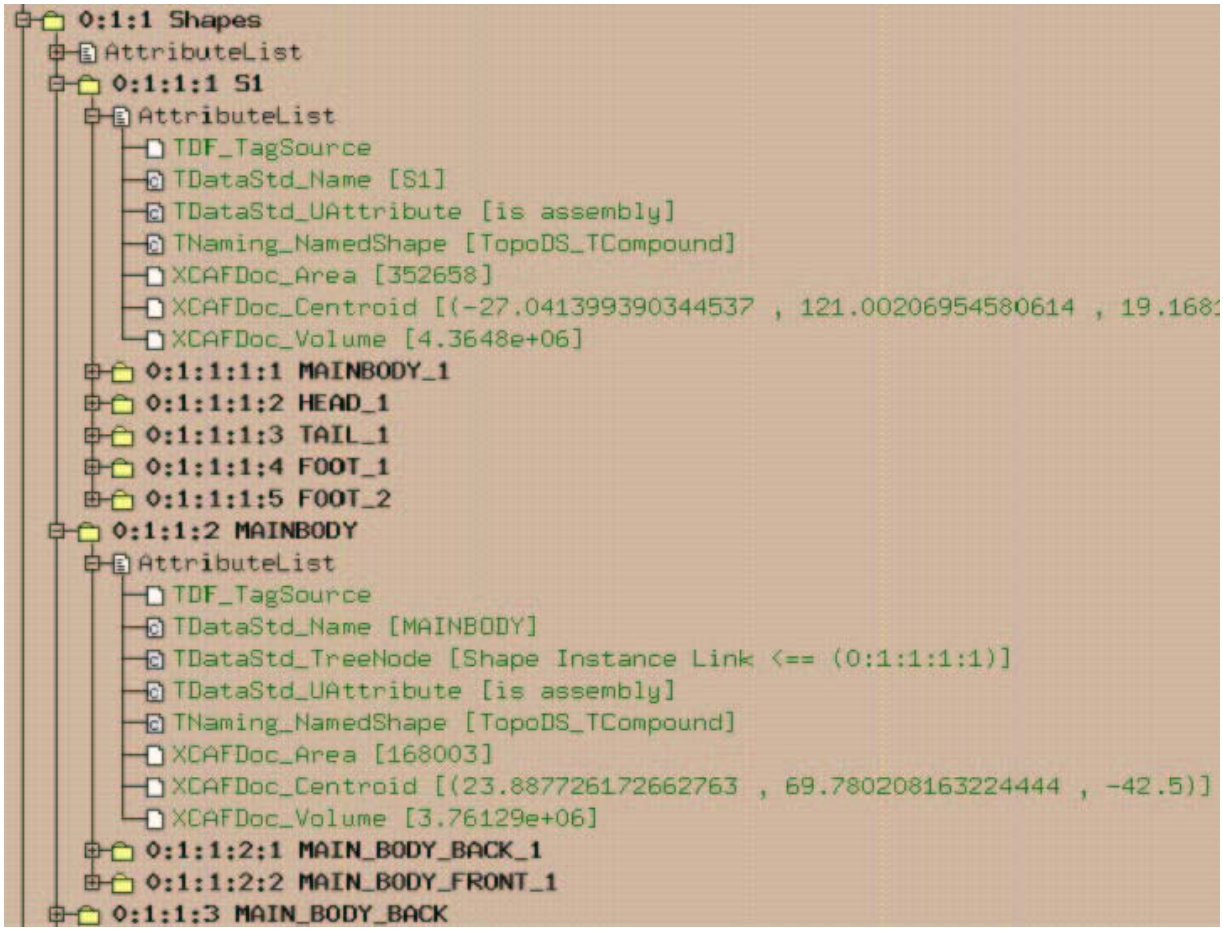

Figure 2. Validation Property Descriptions

Check logs contain deviations of computed values from the values stored in a STEP file. A typical example appears as follows:

Label	Area defect	Volume defect	dX	dY	DZ	Name
0:1:1:1	312.6 (0%)	-181.7 (0%)	0.00	0.00	0.00	"S1"
0:1:1:2	-4.6 (0%)	-191.2 (0%)	-0.00	0.00	-0.00	"MAINBODY"
0:1:1:3	-2.3 (0%)	-52.5 (0%)	-0.00	0.00	0.00	"MAIN_BODY_BACK"
0:1:1:4	-2.3 (0%)	-51.6 (0%)	0.00	0.00	-0.00	"MAIN_BODY_FRONT"
0:1:1:5	2.0 (0%)	10.0 (0%)	-0.00	0.00	-0.00	"HEAD"
0:1:1:6	0.4 (0%)	0.0 (0%)	0.00	-0.00	-0.00	"HEAD_FRONT"
0:1:1:7	0.4 (0%)	0.0 (0%)	0.00	-0.00	-0.00	"HEAD_BACK"
0:1:1:8	-320.6 (0%)	10.9 (0%)	-0.00	0.00	0.00	"TAIL"
0:1:1:9	0.0 (0%)	0.0 (0%)	-0.00	-0.00	0.00	"TAIL_MIDDLE"
0:1:1:10	-186.2 (0%)	4.8 (0%)	-0.00	0.00	-0.00	"TAIL_TURBINE"
0:1:1:11	0.3 (0%)	-0.0 (0%)	-0.00	-0.00	0.00	"FOOT"
0:1:1:12	0.0 (0%)	-0.0 (0%)	0.00	-0.00	-0.00	"FOOT_FRONT"
0:1:1:13	0.0 (0%)	0.0 (0%)	-0.00	0.00	0.00	"FOOT_BACK"

In our example, it can be seen that no errors were detected for either area, volume or positioning data.

1.1.8. Names

XDE supports reading and writing the names of shapes to and from IGES and STEP file formats. This functionality can be switched off if you do not need this type of data, thereby reducing the size of the document.

1.1.9. Colors and Layers

XDE can read and write colors and layers assigned to shapes or their subparts (down to the level of faces and edges) to and from both IGES and STEP formats. Three types of colors are defined in the enumeration XCAFDoc_ColorType:

- generic color (XCAFDoc_ColorGen)
- surface color (XCAFDoc_ColorSurf)
- curve color (XCAFDoc_ColorCurv)

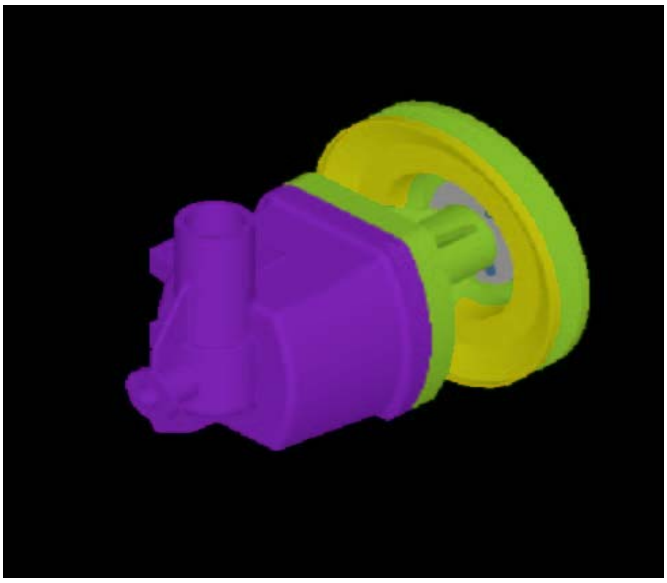

Figure 3. Colors and Layers

2. *Basic Concepts*

2.1. Overview

As explained in the last chapter, XDE uses TDocStd_Documents as a starting point. The general purpose of XDE is:

- Checking if an existing document is fit for XDE
- Getting an application and initialized document
- Initializing a document to fit it for XDE
- Adding, setting and finding data
- Querying and managing shapes
- Attaching properties to shapes

The Document used by XDE usually starts as a TDocStd_Document.

2.1.1. General Check

Before working with shapes, properties, and other types of information, the global organization of an XDE Document can be queried or completed to determine if an existing Document is actually structured for use with XDE.

To find out if an existing TDocStd_Document is suitable for XDE, use:

```
Handle(TDocStd_Document) doc...
if ( XCAFDoc_DocumentTool::IsXCAFDocument (doc) ) { .. yes .. }
```

If the Document is suitable for XDE, you can perform operations and queries explained in this guide. However, if a Document is not fully structured for XDE, it must be initialized. See 2.2.1 Initializing an XDE Document (Shapes).

2.1.2. Getting an Application or an Initialized Document

If you want to retrieve an existing application or an existing document (known to be correctly structured for XDE), use:

```
Handle(TDocStd_Document) aDoc;
Handle(XCAFApp_Application) anApp =
XCAFApp_Application::GetApplication();
anApp->NewDocument("MDTV-XCAF",aDoc);
```

2.2. Shapes and Assemblies

2.2.1. Initializing an XDE Document (Shapes)

An XDE Document begins with a TDocStd_Document. Assuming you have a TDocStd_Document already created, you can ensure that it is correctly structured for XDE by initializing the XDE structure as follows:

```
Handle(TDocStd_Document) doc...
```

```
Handle (XCAFDoc_ShapeTool) myAssembly =
XCAFDoc_DocumentTool::ShapeTool (Doc->Main());
TDF_Label aLabel = myAssembly->NewShape()
```

NOTE *The method XCAFDoc_DocumentTool::ShapeTool returns the XCAFDoc_ShapeTool. The first time this method is used, it creates the XCAFDoc_ShapeTool. In our example, a handle is used for the TDocStd_Document.*

2.2.2. Getting a Node considered as an Assembly

To get a node considered as an Assembly from an XDE structure, you can use the Label of the node. Assuming that you have a properly initialized TDocStd_Document, use:

```
Handle(TDocStd_Document) doc...
Handle(XCAFDoc_ShapeTool) myAssembly =
XCAFDoc_DocumentTool::ShapeTool (aLabel);
```

In the previous example, you can also get the Main Item of an XDE document, which records the root shape representation (as a Compound if it is an Assembly) by using ShapeTool(Doc->Main()) instead of ShapeTool(aLabel).

You can then query or edit this Assembly node, the Main Item or another one (<myAssembly> in our examples).

NOTE *For the examples in the rest of this guide, <myAssembly> is always presumed to be accessed this way, so this information will not be repeated.*

2.2.3. Updating the Assembly after Filling or Editing

Some actions in this chapter affect the content of the document, considered as an Assembly. As a result, you will sometimes need to update various representations (including the compounds).

To update the representations, use:

```
myAssembly->UpdateAssembly(aLabel);
```

Since this call is always used by the editing functions, you need not apply it for such functions. However, you will need this call if special edits, not using XCAF functions, are used on the document.

2.2.4. Adding or Setting Top Level Shapes

Shapes can be added as top-level shapes. Top level means that they can be added to an upper level assembly or added on their own at the highest level as a component or referred by a located instance. Therefore two types of top-level shapes can be added:

- shapes with upper level references
- free shapes (that correspond to roots) without any upper reference

NOTE *Several top-level shapes can be added to the same component.*

A shape to be added can be defined as a compound (if required), with the following interpretations:

- The Shape is a compound

According to the user choice, it may or may not be interpreted as representing an Assembly. If it is an Assembly, each of its subshapes defines a sub-label.

- The Shape is not a compound

The Shape is taken as a whole, without breaking it down.

To break down a Compound in the assembly structure, use:

```
Standard_Boolean makeAssembly;
// True to interpret a Compound as an Assembly,
// False to take it as a whole
aLabel = myAssembly->AddShape(aShape, makeAssembly);
```

Each node of the assembly therefore refers to its sub-shapes.

Concerning located instances of sub-shapes, the corresponding shapes, (without location) appear at distinct sub-labels. They are referred to by a shape instance, which associates a location.

2.2.5. Setting a given Shape at a given Label

A top-level shape can be changed. In this example, no interpretation of compound is performed:

```
Standard_CString LabelString ...;
// identifies the Label (form "0:i:j...")
TDF_Label aLabel...;
// A label must be present
myAssembly->SetShape(aLabel, aShape);
```

2.2.6. Getting a Shape from a Label

To get a shape from its Label from the top-level, use:

```
TDF_Label aLabel...
// A label must be present
if (aLabel.IsNull()) {
  // no such label : abandon
}
TopoDS_Shape aShape;
aShape = myAssembly->GetShape(aLabel);
if (aShape.IsNull()) {
  // this label is not for a Shape
}
```

NOTE: *If the label corresponds to an assembly, then the result is a compound.*

2.2.7. Getting a Label from a Shape

To get a Label which is attached to a Shape from the top-level, use:

```
Standard_Boolean findInstance = Standard_False;
// (this is default value)
aLabel = myAssembly->FindShape(aShape [,findInstance]);
if (aLabel.IsNull()) {
  // no label found for this shape
```

```
}
```

If <findInstance> is True, a search is made for the shape with the same location. If False (default value), a search is made among original, non-located shapes.

2.2.8. Other Queries on a Label

Various other queries can be made from a Label within the Main Item of XDE:

Main Shapes

To determine if a Shape is recorded (or not), use:

```
if ( myAssembly->IsShape(aLabel) ) { .. yes .. }
```

To determine if the shape is "top-level" (was added by the AddShape method), use:

```
if ( myAssembly->IsTopLevel(aLabel) ) { .. yes .. }
```

To get a list of top-level shapes (added by the AddShape method), use:

```
TDF_LabelSequence frshapes;
myAssembly->GetShapes(frshapes);
```

To get all free shapes at once if the list above has only one item, use:

```
TopoDS_Shape result = myAssembly->GetShape(frshapes.Value(1));
```

If there is more than one item, you must create and fill a compound, use:

```
TopoDS_Compound C;
BRep_Builder B;
B.MakeCompound(C);
for(Standard_Integer i=1; i<=frshapes.Length(); i++) {
  TopoDS_Shape S = myAssembly->GetShape(frshapes.Value(i));
  B.Add(C,S);
}
```

In our example, the result is the compound C.

To determine if a shape is a free shape (no reference or super-assembly), use:

```
if ( myAssembly->IsFree(aLabel) ) { .. yes .. }
```

To get a list of Free Shapes (roots), use:

```
TDF_LabelSequence frshapes;
myAssembly->GetFreeShapes(frshapes);
```

To get the shapes, which use a given shape as a component, use:

```
TDF_LabelSequence users;
Standard_Integer nbusers = myAssembly->GetUsers(aLabel,users);
```

The count of users is contained with nbusers. It contains 0 if there are no users.

Assembly and Components

To determine if a label is attached to the main part or to a sub-part (component), use:

```
if (myAssembly->IsComponent(aLabel)) { .. yes .. }
```

To determine whether a label is a node of a (sub-) assembly or a simple shape, use:

```
if ( myAssembly->IsAssembly(aLabel) ) { .. yes .. }
```

If the label is a node of a (sub-) assembly, you can get the count of components, use:

```
Standard_Boolean subchilds = Standard_False; //default
Standard_Integer nbc = myAssembly->NbComponents (aLabel
[,subchilds]);
```

If <subchilds> is True, commands also consider sub-levels. By default, only level one is checked.

To get component Labels themselves, use:

```
Standard_Boolean subchilds = Standard_False; //default
TDF_LabelSequence comps;
Standard_Boolean isassembly = myAssembly->GetComponents
(aLabel,comps[,subchilds]);
```

2.2.9. Instances and References for Components

To determine if a label is a simple shape, use:

```
if ( myAssembly->IsSimpleShape(aLabel) ) { .. yes .. }
```

To determine if a label is a located reference to another one, use:

```
if ( myAssembly->IsReference(aLabel) ) { .. yes .. }
```

If the label is a located reference, you can get the location, use:

```
TopLoc_Location loc = myAssembly->GetLocation (aLabel);
```

To get the label of a referenced original shape (also tests if it is a reference), use:

```
Standard_Boolean isref = myAssembly->GetReferredShape
(aLabel, refLabel);
```

NOTE *<isref> returns False if <aLabel> is not for a reference.*

2.3. Editing Shapes

In addition to the previously described AddShape and SetShape, several shape edits are possible.

To remove a Shape, and all its sub-labels, use:

```
Standard_Boolean remsh = myAssembly->RemoveShape(aLabel);
// remsh is returned True if done
```

NOTE: *This operation will fail if the shape is neither free nor top level.*

To add a Component to the Assembly, from a new shape, use:

```
Standard_Boolean expand = Standard_False; //default
TDF_Label aLabel = myAssembly->AddComponent (aShape
[,expand]);
```

If <expand> is True and if <aShape> is a Compound, it is broken down to produce sub-components, one for each of its sub-shapes.

To add a component to the assembly, from a previously recorded shape (the new component is defined by the label of the reference shape, and its location), use:

```
TDF_Label refLabel ...; // the label of reference shape
TopLoc_Location loc ...; // the desired location
TDF_Label aLabel = myAssembly->AddComponent (refLabel, loc);
```

To remove a component from the assembly, use:

```
myAssembly->RemoveComponent (aLabel);
```

2.4. Management of Sub-Shapes

In addition to components of a (sub-)assembly, it is possible to have individual identification of some sub-shapes inside any shape. Therefore, you can attach specific attributes such as Colors. Some additional actions can be performed on sub-shapes that are neither top-level, nor components:

To add a sub-shape to a given Label, use:

```
TDF_Label subLabel = myAssembly->AddSubShape (aLabel,
subShape);
```

To find the Label attached to a given sub-shape, use:

```
TDF_Label subLabel; // new label to be computed
if ( myAssembly-> FindSubShape (aLabel, subShape,
subLabel)) { .. yes .. }
```

If the sub-shape is found (yes), <subLabel> is filled by the correct value.

To find the top-level simple shape (not a compound whether free or not), which contains a given sub-shape, use:

```
TDF_Label mainLabel = myAssembly->FindMainShape(subShape);
```

NOTE: *There should be only one shape for a valid model. In any case, the search stops on the first one found.*

To get the sub-shapes of a shape, which are recorded under a label, use:

```
TDF_LabelSequence subs;
Standard_Boolean hassubs = myAssembly->GetSubShapes
(aLabel,subs);
```

2.5. Properties

Some properties can be attached directly to shapes. These properties are:

- Name (standard definition from OCAF)
- Centroid (for validation of transfer)
- Volume (for validation of transfer)
- Area (for validation of transfer)

Some other properties can also be attached, and are also managed by distinct tools for Colors and Layers. Colors and Layers are managed as an alternative way of organizing data (by providing a way of identifying groups of shapes).

Colors are put into a table of colors while shapes refer to this table. There are two ways of attaching a color to a shape:

- By attaching an item from the table.
- Adding the color directly.

When the color is added directly, a search is performed in the table of contents to determine if it contains the requested color. Once this search and initialize operation is done, the first way of attaching a color to a shape is used.

2.5.1. Name

Name is implemented and used as a TDataStd_Name, which can be attached to any label. Before proceeding, consider that:

- In IGES, every entity can have a name with an optional numeric part called a Subscript Label. For example, "MYCURVE" is a name, and "MYCURVE(60)" is a name with a Subscript Label.
- In STEP, there are two levels: Part Names and Entity Names:

Part Names are attached to "main shapes" such as parts and assemblies. These Part Names are specifically supported by XDE.

Entity Names can be attached to every Geometric Entity. This option is rarely used, as it tends to overload the exploitation of the data structure. Only some specific cases justify using this option: for example, when the sending system can really ensure the stability of

an entity name after each STEP writing. If such stability is ensured, you can use this option to send an Identifier for external applications using a database.

NOTE: *Both IGES or STEP files handle names as pure ASCII strings.*

These considerations are not specific to XDE. What is specific to data exchange is the way names are attached to entities.

To get the name attached to a label (as a reminder using OCAF), use:

```
Handle(TDataStd_Name) N;
if ( !aLabel.FindAttribute(TDataStd_Name::GetID(),N)) {
  // no name is attached
}
TCollection_ExtendedString name = N->Get();
```

Don't forget to consider Extended String as ASCII, for the exchange file.

To set a name to a label (as a reminder using OCAF), use:

```
TCollection_ExtendedString aName ...;
// contains the desired name for this Label (ASCII)
TDataStd_Name::Set (aLabel, aName);
```

2.5.2. Centroid

A Centroid is defined by a Point to fix its position. It is handled as a property, item of the class XCAFDoc_Centroid, sub-class of TDF_Attribute. However, global methods give access to the position itself.

This notion has been introduced in STEP, together with that of Volume, and Area, as defining the "Validation Properties": this feature allows exchanging the geometries and some basic attached values, in order to perform a synthetic checking on how they are maintained after reading and converting the exchange file. This exchange depends on reliable exchanges of Geometry and Topology. Otherwise, these values can be considered irrelevant.

Along with Volume (see 2.5.4), it can be determined at any level of an assembly, thereby allowing a check of both individual simple shapes and their combinations including locations.

To get a Centroid attached to a Shape, use:

```
gp_Pnt pos;
Handle(XCAFDoc_Centroid) C;
aLabel.FindAttribute ( XCAFDoc_Centroid::GetID(), C );
if ( !C.IsNull() ) pos = C->Get();
```

To set a Centroid to a Shape, use:

```
gp_Pnt pos (X,Y,Z);
// the position previously computed for the centroid
XCAFDoc_Centroid::Set ( aLabel, pos );
```

2.5.3. Area

An Area is defined by a Real, it corresponds to the computed Area of a Shape, provided that it contains surfaces. It is handled as a property, item of the class XCAFDoc_Area, sub-class of TDF_Attribute.

This notion has been introduced in STEP but it is usually disregarded for a Solid, as Volume is used instead. In addition, it is attached to simple shapes, not to assemblies.

To get an area attached to a Shape, use:

```
Standard_Real area;
Handle(XCAFDoc_Area) A;
L.FindAttribute ( XCAFDoc_Area::GetID(), A );
if ( !A.IsNull() ) area = A->Get();
```

To set an area value to a Shape, use:

```
Standard_Real area ...;
// value previously computed for the area
XCAFDoc_Area::Set ( aLabel, area );
```

2.5.4. Volume

A Volume is defined by a Real and corresponds to the computed volume of a Shape, provided that it contains solids. It is handled as a property, an item of the class XCAFDoc_Volume, sub-class of TDF_Attribute.

This notion has been introduced in STEP. It may be attached to simple shapes or their assemblies for computing cumulated volumes and centers of gravity (see also 2.5.2 Centroid).

To get a Volume attached to a Shape, use:

```
Standard_Real volume;
Handle(XCAFDoc_Volume) V;
L.FindAttribute ( XCAFDoc_Volume::GetID(), V );
if ( !V.IsNull() ) volume = V->Get();
```

To set a volume value to a Shape, use:

```
Standard_Real volume ...;
// value previously computed for the volume
XCAFDoc_Volume::Set ( aLabel, volume );
```

2.6. Colors

In an XDE document, colors are managed by the class XCAFDoc_ColorTool. This is done with the same principles as for ShapeTool with Shapes, and with the same capability of having a tool on the Main Label, or on any sub-label. The Property itself is defined as an XCAFDoc_Color, sub-class of TDF_Attribute.

Colors are stored in a child of the starting document label: it is the second level (0.1.2), while Shapes are at the first level. Each color then corresponds to a dedicated label, the property itself is a Quantity_Color, which has a name and value for Red, Green, Blue. A Color may be attached to Surfaces (flat colors) or to Curves (wireframe colors), or to both. A Color may be attached to a sub-shape. In such a case, the sub-shape (and its own sub-shapes) takes its own Color as a priority.

Colors and Shapes are related to by Tree Nodes.

These definitions are common to various exchange formats, at least for STEP and IGES.

2.6.1. Initialization

To query, edit, or initialize a Document to handle Colors of XCAF, use:

```
Handle(XCAFDoc_ColorTool) myColors =
XCAFDoc_DocumentTool::ColorTool(Doc->Main ());
```

This call can be used at any time. The first time it is used, a relevant structure is added to the document. This definition is used for all the following color calls and will not be repeated for these.

2.6.2. Adding a Color

There are two ways to add a color. You can:

- add a new Color defined as Quantity_Color and then directly set it to a Shape (anonymous Color)
- define a new Property Color, add it to the list of Colors, and then set it to various shapes.

When the Color is added by its value Quantity_Color, it is added only if it has not yet been recorded (same RGB values) in the Document.

To set a Color to a Shape using a label, use:

```
Quantity_Color Col (red,green,blue);
XCAFDoc_ColorType ctype ..;
// can take one of these values :
// XCAFDoc_ColorGen : all types of geometries
// XCAFDoc_ColorSurf : surfaces only
// XCAFDoc_ColorCurv : curves only
myColors->SetColor ( aLabel, Col, ctype );
```

Alternately, the Shape can be designated directly, without using its label, use:

```
myColors->SetColor ( aShape, Col, ctype );
// Creating and Adding a Color, explicitly
Quantity_Color Col (red,green,blue);
TDF_Label ColLabel = myColors->AddColor ( Col );
```

NOTE: *this Color can then be named, allowing later retrieval by its Name instead of its Value.*

To set a Color, identified by its Label and already recorded, to a Shape, use:

```
XCAFDoc_ColorType ctype ..; // see above
if ( myColors->SetColors ( aLabel, ColLabel, ctype) ) {
.. it is done .. }
```

In this example, <aLabel> can be replaced by <aShape> directly.

2.6.3. Queries on Colors

Various queries can be performed on colors. However, only specific queries are included in this section, not general queries using names.

To determine if a Color is attached to a Shape, for a given color type (<ctype>), use:

```
if ( myColors->IsSet (aLabel , ctype)) {
  // yes, there is one ..
}
```

In this example, <aLabel> can be replaced by <aShape> directly.

To get the Color attached to a Shape (for any color type), use:

```
Quantity_Color col;
// will receive the recorded value (if there is some)
if ( !myColors->GetColor(aLabel, col) ) {
// sorry, no color ..
}
// color name can also be queried from : col.StringName
// or col.Name
```

In this example, <aLabel> can be replaced by <aShape> directly.

To get the Color attached to a Shape, with a specific color type, use:

```
XCAFDoc_ColorType ctype ..;
// the desired color type
Quantity_Color col;
// will receive the recorded value (if there is some)
if ( !myColors->GetColor(aLabel, ctype, col) ) {
// sorry, no color ..
}
```

To get all the Colors recorded in the Document, use:

```
Quantity_Color col; // to receive the values
TDF_LabelSequence ColLabels;
myColors->GetColors(ColLabels);
Standard_Integer i, nbc = ColLabels.Length();
```

```
for (i = 1; i <= nbc; i ++) {
  aLabel = Labels.Value(i);
  if ( !myColors->GetColor(aLabel, col) ) continue;
  // <col> receives the color n0 i ..
}
```

To find a Color from its Value, use:

```
Quantity_Color Col (red,green,blue);
TDF_Label ColLabel = myColors->FindColor (Col);
if ( !ColLabel.IsNull() ) { .. found .. }
```

2.6.4. Editing Colors

Besides adding colors, the following attribute edits can be made:

To unset a Color on a Shape, use:

```
XCAFDoc_ColorType ctype ...;
// desired type (XCAFDoc_ColorGen for all )
myColors->UnSetColor (aLabel,ctype);
```

To remove a Color and all the references to it (so that the related shapes will become colorless), use:

```
myColors->RemoveColor(ColLabel);
```

2.7. Layers

Layers are handled using the same principles as for Colors (see 2.6 Colors). Simply replace "Color" with "Layer" when dealing with Layers. Layers are supported by the class XCAFDoc_LayerTool.

The class of the property is XCAFDoc_Layer, sub-class of TDF_Attribute while its definition is a TCollection_ExtendedString. Integers are generally used when dealing with Layers. The general cases are:

- IGES has LevelList as a list of Layer Numbers (not often used)
- STEP identifies a Layer (not by a Number, but by a String), to be more general.

2.7.1. Reading and Writing STEP or IGES

Note that saving and restoring the document itself are standard OCAF operations. As the various previously described definitions enter into this frame, they will not be explained any further.

The same can be said for Viewing: presentations can be defined from Shapes and Colors.

There are several important points to consider:

- Previously defined Readers and Writers for dealing with Shapes only, whether Standard or Advanced, remain unchanged in their form and in their dependencies. In addition, functions other than mapping are also unchanged.
- XDE provides mapping with data other than Shapes. Names, Colors, Layers, Validation Properties (Centroid, Volume, Area), and Assembly Structure are hierarchic with rigid motion.
- XDE mapping is relevant for use within the Advanced level of Data Exchanges, rather than Standard ones, because a higher level of information is better suited to a higher quality of shapes. In addition, this allows to avoid the multiplicity of combinations between various options. Note that this choice is not one of architecture but of practical usage and packaging.
- Reader and Writer classes for XDE are generally used like those for Shapes. However, their use is adapted to manage a Document rather than a Shape.

The packages to manage this are IGESCAFControl for IGES, and STEPCAFControl for STEP.

2.7.2. Reading a STEP file

To read a STEP file by itself, use:

```
STEPCAFControl_Reader reader;
IFSelect_ReturnStatus readstat = reader.ReadFile(filename);
// The various ways of reading a file are available here too :
// to read it by the reader, to take it from a WorkSession ...
Handle(TDocStd_Document) doc...
// the document referred to is already defined and
// properly initialized.
// Now, the transfer itself
if ( !reader.Transfer ( doc ) ) {
  cout<<"Cannot read any relevant data from the STEP file"<<endl;
  // abandon ..
}
// Here, the Document has been filled from a STEP file,
// it is ready to use
```

In addition, the reader provides methods that are applicable to document transfers and for directly querying of the data produced.

2.7.3. Writing a STEP file

To write a STEP file by itself, use:

```
STEPControl_StepModelType mode =
STEPControl_AsIs;
// Asis is the recommended value, others are available
// Firstly, perform the conversion to STEP entities
STEPCAFControl_Writer writer;
//(the user can work with an already prepared WorkSession or
create a //new one)
```

```
Standard_Boolean scratch = Standard_False;
STEPCAFControl_Writer writer ( WS, scratch );
// Translating document (conversion) to STEP
if ( ! writer.Transfer ( Doc, mode ) ) {
  cout<<"The document cannot be translated or gives no
result"<<endl;
  // abandon ..
}
// Writing the File
IFSelect_ReturnStatus stat = writer.Write(file-name);
```

2.7.4. Reading an IGES File

Use the same procedure as for a STEP file but with IGESCAFControl instead of STEPCAFControl.

2.7.5. Writing an IGES File

Use the same procedure as for a STEP file but with IGESCAFControl instead of STEPCAFControl.

2.8. Using an XDE Document

There are several ways of exploiting XDE data from an application, you can:

1. Get the data relevant for the application by mapping XDE/Appli, then discard the XDE data once it has been used.
2. Create a reference from the Application Document to the XDE Document, to have its data available as external data.
3. Embed XDE data inside the Application Document (see the following section for details).
4. Directly exploit XDE data such as when using file checkers.

2.8.1. XDE Data inside an Application Document

To have XCAF data elsewhere than under label 0.1, you use the DocLabel of XDE. The method DocLabel from XCAFDoc_DocumentTool determines the relevant Label for XCAF. However, note that the default is 0.1.

In addition, as XDE data is defined and managed in a modular way, you can consider exclusively Assembly Structure, only Colors, and so on.

As XDE provides an extension of the data structure, for relevant data in standardized exchanges, note the following:

- This data structure is fitted for data exchange, rather than for use by the final application.
- The provided definitions are general, for common use and therefore do not bring strongly specific semantics.

As a result, if an application works on Assemblies, on Colors or Layers, on Validation Properties (as defined in STEP), it can rely on all or a part of the XDE definitions, and

include them in its own data structure.

In addition, if an application has a data structure far from these notions, it can get data (such as Colors and Names on Shapes) according to its needs, but without having to consider the whole.

3. *Package XCAFDoc*

3.1. General description

This package is intended for definition of general structure of an XDE document and tools to work with it.

The document is composed of sections, each section storing its own kind of data and managing by a corresponding tool.

The API enumeration is

- XCAFDoc_ColorType.

The API classes are the following:

- XCAFDoc_DocumentTool
- XCAFDoc_Location
- XCAFDoc_Color
- XCAFDoc_Volume
- XCAFDoc_ Area
- XCAFDoc_ Centroid
- XCAFDoc_ ShapeTool
- XCAFDoc_ ColorTool
- XCAFDoc_ LayerTool
- XCAFDoc_ GraphNode

3.2. Enumeration XCAFDoc_ColorType

Definition:

```
enumeration ColorType is
ColorGen,   -- simple color
ColorSurf,  -- color of surfaces
ColorCurv   -- color of curves
end ColorType;
```

Purpose: Defines types of color assignments.

3.3. Class XCAFDoc_DocumentTool

3.3.1. General description

Purpose: Attribute marking an OCAF document as being an XDE document.

Creates the sections structure of the document.

3.3.2. Methods

The methods are the following:

Constructors

```
XCAFDoc_DocumentTool()
```

Purpose: Empty constructor

Methods:

- XCAFDoc_DocumentTool::Init

```
void XCAFDoc_DocumentTool::Init() const
```

Purpose: To be called when reading this attribute from a file.

- XCAFDoc_DocumentTool::Set

```
Handle(XCAFDoc_DocumentTool) XCAFDoc_DocumentTool::Set(
const TDF_Label& L, const Standard_Boolean IsAcces)
```

- Purpose: Creates (if does not exist) a DocumentTool attribute on 0.1 label if <IsAcces> is true, else on <L> label. This label will be returned by DocLabel(); If the attribute is already set it will not be reset on <L> even if <IsAcces> is false. ColorTool and ShapeTool attributes are also set by this method. Returns DocumentTool from XCAFDoc;

- XCAFDoc_DocumentTool::DocLabel

```
TDF_Label XCAFDoc_DocumentTool::DocLabel(
const TDF_Label& acces)
```

- Purpose: Returns the label where the DocumentTool attribute is or “0.1” if DocumentTool is not yet set.

- XCAFDoc_DocumentTool::ShapesLabel

```
TDF_Label XCAFDoc_DocumentTool::ShapesLabel(
const TDF_Label& acces)
```

Purpose: Returns the shapes sub-label of DocLabel().

- XCAFDoc_DocumentTool::ColorsLabel

```
TDF_Label XCAFDoc_DocumentTool::ColorsLabel(
const TDF_Label& acces)
```

Purpose: Returns the colors sub-label of DocLabel().

- XCAFDoc_DocumentTool::LayersLabel

```
TDF_Label XCAFDoc_DocumentTool::LayersLabel(
const TDF_Label& acces)
```

Purpose: Returns the layers sub-label of DocLabel().

- XCAFDoc_DocumentTool::ShapeTool

```
Handle(XCAFDoc_ShapeTool) XCAFDoc_DocumentTool::ShapeTool(
const TDF_Label& acces)
```

Purpose: Creates (if does not exist) a ShapeTool attribute on ShapesLabel().

- XCAFDoc_DocumentTool::ColorTool

```
Handle(XCAFDoc_ColorTool) XCAFDoc_DocumentTool::ColorTool (
```

```
const TDF_Label& acces)
```

Purpose: Creates (if does not exist) a ColorTool attribute on ColorsLabel().

- XCAFDoc_DocumentTool::LayerTool

```
Handle(XCAFDoc_LayerTool) XCAFDoc_DocumentTool::LayerTool (
const TDF_Label& acces)
```

Purpose: Creates (if does not exist) a LayerTool attribute on LayersLabel().

3.4. Class XCAFDoc_Location

3.4.1. General description

Purpose: Attribute to store TopLoc_Location.

Inherits Attribute from TDF

3.4.2. Methods

The methods are the following:

Constructors

```
XCAFDoc_Location()
```

Purpose: Empty constructor.

Methods:

- XCAFDoc_Location::Set

```
void XCAFDoc_Location::Set(const TopLoc_Location& Loc)
```

Purpose: Sets a Location attribute.

```
Handle(XCAFDoc_Location) XCAFDoc_Location::Set(
const TDF_Label& L,const TopLoc_Location& Loc)
```

- Purpose: Finds or creates a Location attribute and sets its value. The Location attribute is returned.
- XCAFDoc_Location::Get

```
TopLoc_Location XCAFDoc_Location::Get() const
```

Purpose: Returns Location from TopLoc.

3.5. Class XCAFDoc_Color

3.5.1. General description

Purpose: Attribute to store color.

Inherits Attribute from TDF

3.5.2. Methods

The methods are the following:

Constructors

`XCAFDoc_Color()`

Purpose: Empty constructor.

Methods:

- XCAFDoc_Color::Set

```
void XCAFDoc_Color::Set(const Quantity_Color& C)
void XCAFDoc_Color::Set(const Quantity_NameOfColor C)
void XCAFDoc_Color::Set(const Standard_Real R,
const Standard_Real G, const Standard_Real B)
```

Purpose: Sets a Color attribute.

```
Handle(XCAFDoc_Color) XCAFDoc_Color::Set(
const TDF_Label& L, const Quantity_Color& C)
Handle(XCAFDoc_Color) XCAFDoc_Color::Set(
const TDF_Label& L, const Quantity_NameOfColor C)
Handle(XCAFDoc_Color) XCAFDoc_Color::Set(
const TDF_Label& L, const Standard_Real R,
const Standard_Real G, const Standard_Real B)
```

Purpose: Finds or creates a Color attribute and sets its value. The Color attribute is returned.

- XCAFDoc_Color::GetColor

```
Quantity_Color XCAFDoc_Color::GetColor() const
```

Purpose: Returns a Color attribute.

- XCAFDoc_Color::GetNOC

```
Quantity_NameOfColor XCAFDoc_Color::GetNOC() const
```

Purpose: Returns a Name of Color.

- XCAFDoc_Color::GetRGB

```
void XCAFDoc_Color::GetRGB(Standard_Real& R,
Standard_Real& G, Standard_Real& B) const
```

Purpose: Returns RGB.

3.6. Class XCAFDoc_Volume

3.6.1. General description

Purpose: Attribute to store volume.

Inherits Attribute from TDF

3.6.2. Methods

The methods are the following:

Constructors

`XCAFDoc_Volume()`

Purpose: Empty constructor.

Methods:

- XCAFDoc_Volume::Set

```
void XCAFDoc_Volume::Set (const Standard_Real V)
```

Purpose: Sets a value of Volume.

```
Handle(XCAFDoc_Volume) XCAFDoc_Volume::Set (
const TDF_Label& L,const Standard_Real V)
```

Purpose: Finds or creates a Volume attribute and sets its value.

- XCAFDoc_Volume::Get

```
Standard_Real XCAFDoc_Volume::Get() const
```

Purpose: Returns a value of Volume

```
Standard_Boolean XCAFDoc_Volume::Get(
const TDF_Label& label,Standard_Real& vol)
```

Purpose: Returns Volume as argument. Returns false if there is no such attribute at the <label>.

3.7. Class XCAFDoc_Area

3.7.1. General description

Purpose: Attribute to store area.

Inherits Attribute from TDF

3.7.2. Methods

The methods are the following:

Constructors

`XCAFDoc_Area()`

Purpose: Empty constructor.

Methods:

- XCAFDoc_Area::Set

```
void XCAFDoc_Area::Set (const Standard_Real V)
```

Purpose: Sets a value of area.

```
Handle(XCAFDoc_Area) XCAFDoc_Area::Set (
const TDF_Label& L,const Standard_Real V)
```

Purpose: Finds or creates an Area attribute and sets its value.

- XCAFDoc_Area::Get

```
Standard_Real XCAFDoc_Area::Get() const
```

Purpose: Returns a value of area.

```
Standard_Boolean XCAFDoc_Area::Get(
const TDF_Label& label, Standard_Real& area)
```

Purpose: Returns the volume of area as an argument and success status returns false if there is no such attribute at the <label>.

3.8. Class XCAFDoc_Centroid

3.8.1. General description

Purpose: Attribute to store Centroid.

Inherits Attribute from TDF

3.8.2. Methods

The methods are the following:

Constructors

```
XCAFDoc_Centroid()
```

Purpose: Empty constructor.

Methods:

- XCAFDoc_Centroid::Set

```
void XCAFDoc_Centroid::Set(const gp_Pnt& pnt)
```

Purpose: Sets a Centroid attribute.

```
Handle(XCAFDoc_Centroid) XCAFDoc_Centroid::Set(
const TDF_Label& L, const gp_Pnt& pnt)
```

Purpose: Finds or creates a Centroid attribute and sets its value. The Centroid attribute is returned.

- XCAFDoc_Area::Get

```
gp_Pnt XCAFDoc_Centroid::Get() const
```

Purpose: Returns a Centroid attribute.

```
Standard_Boolean XCAFDoc_Centroid::Get(
const TDF_Label& label,gp_Pnt& pnt)
```

Purpose: Returns a point as an argument. Returns false if there is no such attribute at the <label>.

3.9. Class XCAFDoc_ShapeTool

3.9.1. General description

Purpose: attribute containing the Shapes section of an XDE document.

Provides tools for management of the Shapes section.

3.9.2. Methods

The methods are the following:

Constructors

`XCAFDoc_ShapeTool()`

Purpose: Empty constructor.

Method Set

- XCAFDoc_ ShapeTool::Set

```
Handle(XCAFDoc_ShapeTool) XCAFDoc_ShapeTool::Set(const TDF_Label&
L)
```

Purpose: Creates (if does not exist) ShapeTool from XCAFDoc on <L>.

Methods for work with top-level structure of shapes

- XCAFDoc_ShapeTool::Search

```
Standard_Boolean XCAFDoc_ShapeTool::Search (
const TopoDS_Shape &S, TDF_Label &L,
const Standard_Boolean findInstance,
const Standard_Boolean findComponent,
const Standard_Boolean findSubShape) const
```

Purpose: General tool to find a (sub) shape in the document

- If <findInstance> is True, and <S> has a non-null location, first tries to find the shape among the top-level shapes with this location

- If not found, and <findComponent> is True, tries to find the shape among the components of assemblies

- If not found, tries to find the shape without location among top-level shapes

- If not found and <findSubshape> is True, tries to find a shape as a subshape of top-level simple shapes

Returns False if nothing is found.

- XCAFDoc_ShapeTool::FindShape

```
Standard_Boolean XCAFDoc_ShapeTool::FindShape (
const TopoDS_Shape& S, TDF_Label& L,
const Standard_Boolean findInstance) const
```

Purpose. Returns the label corresponding to shape <S> (searches among top-level shapes, not including subcomponents of assemblies). If <findInstance> is False (default), searches for the non-located shape (i.e. among original shapes). If <findInstance> is True, searches for the shape with the same location, including shape instances. Returns True if <S> is found.

```
TDF_Label XCAFDoc_ShapeTool::FindShape (const TopoDS_Shape& S,
const Standard_Boolean findInstance) const
```

Purpose: Does the same as the previous method. Returns the Null label if not found.

- XCAFDoc_ShapeTool::GetShape

```
Standard_Boolean XCAFDoc_ShapeTool::GetShape (
const TDF_Label& L, TopoDS_Shape& S)
```

Purpose: To get TopoDS_Shape from the shape's label. For component, returns a new shape with a correct location. Returns False if the label does not contain a shape.

```
TopoDS_Shape XCAFDoc_ShapeTool::GetShape(const TDF_Label& L)
```

Purpose: To get TopoDS_Shape from the shape's label. For component, returns a new shape with a correct location. Returns a Null shape if the label does not contain a shape.

- XCAFDoc_ShapeTool::NewShape

```
TDF_Label XCAFDoc_ShapeTool::NewShape() const
```

Purpose: Creates a new (empty) top-level shape. Initially it holds an empty TopoDS_Compound.

- XCAFDoc_ShapeTool::SetShape

```
void XCAFDoc_ShapeTool::SetShape (const TDF_Label& L,
const TopoDS_Shape& S) const
```

Purpose: Sets representation (TopoDS_Shape) for the top-level shape.

- XCAFDoc_ShapeTool::AddShape

```
TDF_Label XCAFDoc_ShapeTool::AddShape (const TopoDS_Shape& S,
const Standard_Boolean makeAssembly) const
```

Purpose: Adds a new top-level (creates and returns a new label). If <makeAssembly> is True, treats TopAbs_COMPOUND shapes as assemblies (creates assembly structure).

- XCAFDoc_ShapeTool::RemoveShape

```
Standard_Boolean XCAFDoc_ShapeTool::RemoveShape (
const TDF_Label& L) const
```

Purpose: Removes a shape (whole label and all its sublabels). Returns False (and does nothing) if the shape is not free or is not a top-level shape.

- XCAFDoc_ShapeTool::GetShapes

```
void XCAFDoc_ShapeTool::GetShapes (TDF_LabelSequence& Labels)
const
```

Purpose: Returns a sequence of all top-level shapes.

- XCAFDoc_ShapeTool::GetFreeShapes

```
void XCAFDoc_ShapeTool::GetFreeShapes (
TDF_LabelSequence& FreeLabels) const
```

Purpose: Returns a sequence of all top-level shapes, which are free (i.e. not referred by any other shape).

- XCAFDoc_ShapeTool::GetUsers

```
Standard_Integer XCAFDoc_ShapeTool::GetUsers (const TDF_Label& L,
TDF_LabelSequence& Labels,
const Standard_Boolean getsubchilds)
```

Purpose: Returns a list of labels, which refer to shape <L> as a component. Returns the number of users (0 if the shape is free).

- XCAFDoc_ShapeTool::GetLocation

```
TopLoc_Location XCAFDoc_ShapeTool::GetLocation (const TDF_Label&
L)
```

Purpose: Returns the location of instance.

- XCAFDoc_ShapeTool::GetReferredShape

```
Standard_Boolean XCAFDoc_ShapeTool::GetReferredShape (
const TDF_Label& L, TDF_Label& Label)
```

Purpose: Returns the label, which corresponds to a shape referred by <L>. Returns False if the label is not a reference.

Methods for analysis

- XCAFDoc_ShapeTool::IsTopLevel

```
Standard_Boolean XCAFDoc_ShapeTool::IsTopLevel (const TDF_Label&
L) const
```

Purpose: Returns True if the label is a label of a top-level shape, as opposed to a component of an assembly or a subshape.

- XCAFDoc_ShapeTool::IsShape

```
Standard_Boolean XCAFDoc_ShapeTool::IsShape (const TDF_Label& L)
```

Purpose: Returns True if the label represents a shape (simple shape, assembly or reference).

- XCAFDoc_ShapeTool::IsSimpleShape

```
Standard_Boolean XCAFDoc_ShapeTool::IsSimpleShape (const
TDF_Label& L)
```

Purpose: Returns True if the label is a label of a simple shape.

- XCAFDoc_ShapeTool::IsReference

```
Standard_Boolean XCAFDoc_ShapeTool::IsReference (const TDF_Label&
L)
```

Purpose: Returns true if <L> is a located instance of another shape i.e. a reference.

XCAFDoc_ShapeTool::IsAssembly

```
Standard_Boolean XCAFDoc_ShapeTool::IsAssembly (const TDF_Label&
L)
```

Purpose: Returns True if the label is a label of assembly, i.e. contains sublabels which are assembly components. This is relevant only if IsShape() is True.

- XCAFDoc_ShapeTool::IsComponent

```
Standard_Boolean XCAFDoc_ShapeTool::IsComponent (const TDF_Label&
L)
```

Purpose: Returns true if <L> is a reference serving as a component of an assembly.

- XCAFDoc_ShapeTool::IsSubShape

```
Standard_Boolean XCAFDoc_ShapeTool::IsSubShape (const TDF_Label&
L)
```

Purpose: Returns true if <L> is a subshape of a top-level shape.

```
Standard_Boolean XCAFDoc_ShapeTool::IsSubShape (
const TDF_Label &shapeL,
const TopoDS_Shape &sub) const
```

Purpose: Checks whether shape <sub> is a subshape of a shape stored on label <shapeL>.

- XCAFDoc_ShapeTool::IsFree

```
Standard_Boolean XCAFDoc_ShapeTool::IsFree (const TDF_Label& L)
```

Purpose: Returns True if the label is not used by any assembly, i.e. contains sublabels which are assembly components. This is relevant only if IsShape() is True (There is no Father TreeNode on this <L>).

Methods for work with assembly structure

- XCAFDoc_ShapeTool::NbComponents

```
Standard_Integer XCAFDoc_ShapeTool::NbComponents (const TDF_Label&
L,
const Standard_Boolean getsubchilds)
```

Purpose: Returns the number of Assembles components.

- XCAFDoc_ShapeTool::GetComponents

```
Standard_Boolean XCAFDoc_ShapeTool::GetComponents (const
TDF_Label& L,
TDF_LabelSequence& Labels,
const Standard_Boolean getsubchilds)
```

Purpose: Returns a list of components of an assembly. Returns False if the label is not an assembly.

- XCAFDoc_ShapeTool::AddComponent

```
TDF_Label XCAFDoc_ShapeTool::AddComponent (const TDF_Label&
assembly,
const TDF_Label& compL, const TopLoc_Location &Loc) const
```

Purpose: Adds a component given by its label and location to the assembly. Note: the assembly must be IsAssembly() or IsSimpleShape().

- XCAFDoc_ShapeTool::AddComponent

```
TDF_Label XCAFDoc_ShapeTool::AddComponent (const TDF_Label&
assembly,
const TopoDS_Shape& comp,
const Standard_Boolean expand) const
```

Purpose: Adds a shape (located) as a component to the assembly. If necessary, creates an additional top-level shape for the component and returns the Label of component. If <expand> is True and the component is Compound, it will be created as an assembly also. Note: the assembly must be IsAssembly() or IsSimpleShape().

- XCAFDoc_ShapeTool::RemoveComponent

```
void XCAFDoc_ShapeTool::RemoveComponent (const TDF_Label& comp)
const
```

Purpose: Removes a component from its assembly.

- XCAFDoc_ShapeTool::UpdateAssembly

```
void XCAFDoc_ShapeTool::UpdateAssembly (const TDF_Label& L) const
```

Purpose: Updates an assembly at label <L>.

Methods for work with sub-shapes of shape

- XCAFDoc_ShapeTool::FindSubShape

```
Standard_Boolean XCAFDoc_ShapeTool::FindSubShape (
const TDF_Label &shapeL, const TopoDS_Shape &sub,
TDF_Label &L) const
```

Purpose: Finds a label for subshape <sub> of the shape stored on label <shapeL>. Returns a Null label if it is not found.

- XCAFDoc_ShapeTool::AddSubShape

```
TDF_Label XCAFDoc_ShapeTool::AddSubShape (const TDF_Label &shapeL,
const TopoDS_Shape &sub) const
```

Purpose: Adds a label for subshape <sub> of the shape stored on label <shapeL>. Returns a Null label if it is not a subshape.

- XCAFDoc_ShapeTool::FindMainShape

```
TDF_Label XCAFDoc_ShapeTool::FindMainShape (
const TopoDS_Shape &sub) const
```

Purpose: Performs a search among top-level shapes to find the shape containing <sub> as a subshape. Checks only simple shapes, and returns the first found label (which should be the only one for valid model).

- XCAFDoc_ShapeTool::GetSubShapes

```
Standard_Boolean XCAFDoc_ShapeTool::GetSubShapes (
const TDF_Label &L, TDF_LabelSequence& Labels)
```

Purpose: Returns the list of labels identifying subshapes of the given shape. Returns False if no subshapes are placed on that label.

Auxiliary methods

- XCAFDoc_ShapeTool::MakeReference

```
void XCAFDoc_ShapeTool::MakeReference (const TDF_Label &L,
const TDF_Label &refL,     const TopLoc_Location &loc)
```

Purpose: Makes a shape on label <L> to be a reference to shape <refL> with location <loc>.

- XCAFDoc_ShapeTool::BaseLabel

```
TDF_Label XCAFDoc_ShapeTool::BaseLabel () const
```

Purpose: returns the label under which shapes are stored.

3.10.Class XCAFDoc_ColorTool

3.10.1. General description

Purpose: Attribute containing Colors section of an XDE document.

Provides tools for management of the Colors section of a document.

3.10.2. Methods

The methods are the following:

Constructors

`XCAFDoc_ShapeTool()`

Purpose: Empty constructor.

Method Set

- XCAFDoc_ColorTool::Set

```
Handle(XCAFDoc_ColorTool) XCAFDoc_ColorTool::Set(const TDF_Label&
L)
```

Purpose: Creates (if does not exist) a ColorTool.

Methods for general structure

- XCAFDoc_ColorTool::BaseLabel

```
TDF_Label XCAFDoc_ColorTool::BaseLabel() const
```

Purpose: Returns the label under which colors are stored

- XCAFDoc_ColorTool::ShapeTool

```
const Handle(XCAFDoc_ShapeTool)& XCAFDoc_ColorTool::ShapeTool()
```

Purpose: Returns the internal XCAFDoc_ShapeTool tool

Methods for color table management

- XCAFDoc_ColorTool::IsColor

```
Standard_Boolean XCAFDoc_ColorTool::IsColor (const TDF_Label& lab)
const
```

Purpose: Returns True if the label belongs to a colortable and is a color definition

- XCAFDoc_ColorTool::GetColor

```
Standard_Boolean XCAFDoc_ColorTool::GetColor (const TDF_Label&
lab,
Quantity_Color& col) const
```

Purpose: Returns a color defined by label lab. Returns False if the label is not in the colortable or does not define a color.

- XCAFDoc_ColorTool::FindColor

```
Standard_Boolean XCAFDoc_ColorTool::FindColor (
const Quantity_Color& col,
TDF_Label& lab) const
```

Purpose: Finds a color definition in the colortable and returns its label if found. Returns False if the color is not found in the colortable.

- XCAFDoc_ColorTool::FindColor

```
TDF_Label XCAFDoc_ColorTool::FindColor (const Quantity_Color& col)
const
```

Purpose: Finds a color definition in the colortable and returns its label if found (or Null label otherwise)

- XCAFDoc_ColorTool::AddColor

```
TDF_Label XCAFDoc_ColorTool::AddColor (const Quantity_Color& col)
const
```

Purpose: Adds a color definition to the colortable and returns its label (returns the existing label if the same color is already defined).

- XCAFDoc_ColorTool::RemoveColor

```
void XCAFDoc_ColorTool::RemoveColor (const TDF_Label& lab) const
```

Purpose: Removes color from the colortable.

- XCAFDoc_ColorTool::GetColors

```
void XCAFDoc_ColorTool::GetColors (TDF_LabelSequence& Labels)
const
```

Purpose: Returns a sequence of colors currently stored in the colortable.

Methods for assignment of colors to labels

- XCAFDoc_ColorTool::SetColor

```
void XCAFDoc_ColorTool::SetColor (const TDF_Label& L,
const TDF_Label& colorL, const XCAFDoc_ColorType type) const
```

Purpose: Sets a link with a GUID defined by <type> (see XCAFDoc::ColorRefGUID()) from label <L> to a color defined by <colorL>.

```
void XCAFDoc_ColorTool::SetColor (const TDF_Label& L,
const Quantity_Color& Color,
const XCAFDoc_ColorType type) const
```

Purpose: Sets a link with a GUID defined by <type> (see XCAFDoc::ColorRefGUID()) from label <L> to color <Color> in the colortable. Adds a color as necessary.

- XCAFDoc_ColorTool::UnSetColor

```
void XCAFDoc_ColorTool::UnSetColor (const TDF_Label& L,
const XCAFDoc_ColorType type) const
```

Purpose: Removes a link with a GUID defined by <type> (see XCAFDoc::ColorRefGUID()) from label <L> to color <Color>.

- XCAFDoc_ColorTool::IsSet

```
Standard_Boolean XCAFDoc_ColorTool::IsSet (const TDF_Label& L,
const XCAFDoc_ColorType type) const
```

Purpose: Returns True if label <L> has a color assignment of the type <type>.

- XCAFDoc_ColorTool::GetColor

```
Standard_Boolean XCAFDoc_ColorTool::GetColor (const TDF_Label& L,
const XCAFDoc_ColorType type, TDF_Label& colorL)
```

Purpose: Returns a label with a color assigned to <L> as <type>. Returns False if no such color is assigned.

```
Standard_Boolean XCAFDoc_ColorTool::GetColor (const TDF_Label& L,
const XCAFDoc_ColorType type, Quantity_Color& color)
```

Purpose: Returns a color assigned to <L> as <type>. Returns False if no such color is assigned.

Methods for assignment of colors to shapes in the Shapes section

- XCAFDoc_ColorTool::SetColor

```
Standard_Boolean XCAFDoc_ColorTool::SetColor (const TopoDS_Shape&
S,
const TDF_Label& colorL, const XCAFDoc_ColorType type)
```

Purpose: Sets a link with a GUID defined by <type> (see XCAFDoc::ColorRefGUID()) from label <L> to the color defined by <colorL>. Returns False if cannot find a label for shape S.

```
Standard_Boolean XCAFDoc_ColorTool::SetColor (const TopoDS_Shape&
S,
const Quantity_Color& Color, const XCAFDoc_ColorType type)
```

Purpose: Sets a link with a GUID defined by <type> (see XCAFDoc::ColorRefGUID()) from label <L> to color <Color> in the colortable. Adds a color as necessary. Returns False if cannot find a label for shape S.

- XCAFDoc_ColorTool::UnSetColor

```
Standard_Boolean XCAFDoc_ColorTool::UnSetColor (const
TopoDS_Shape& S,
const XCAFDoc_ColorType type)
```

Purpose: Removes a link with a GUID defined by <type> (see XCAFDoc::ColorRefGUID()) from label <L> to color <Color>. Returns True if such link existed.

- XCAFDoc_ColorTool::IsSet

```
Standard_Boolean XCAFDoc_ColorTool::IsSet (const TopoDS_Shape& S,
const XCAFDoc_ColorType type)
```

Purpose: Returns True if label <L> has a color assignment of the type <type>.

- XCAFDoc_ColorTool::GetColor

```
Standard_Boolean XCAFDoc_ColorTool::GetColor (const TopoDS_Shape&
S,
const XCAFDoc_ColorType type, TDF_Label& colorL)
```

Purpose: Returns label with a color assigned to <L> as <type>. Returns False if no such color is assigned.

```
Standard_Boolean XCAFDoc_ColorTool::GetColor (const TopoDS_Shape&
S,
const XCAFDoc_ColorType type, Quantity_Color& color)
```

Purpose: Returns a color assigned to <L> as <type>. Returns False if no such color is assigned.

3.11. Class XCAFDoc_LayerTool;

3.11.1. General description

Purpose: Attribute containing Layers section of an XDE document.

Provides tools for management of the Layers section of a document.

3.11.2. Methods

The methods are the following:

Constructors

`XCAFDoc_LayerTool()`

Purpose: Empty constructor.

Method Set

- XCAFDoc_LayerTool::Set

Handle(XCAFDoc_LayerTool) XCAFDoc_LayerTool::Set(const TDF_Label& L)

Method of general structure

- XCAFDoc_LayerTool::BaseLabel

```
TDF_Label XCAFDoc_LayerTool::BaseLabel() const
```

Purpose: Returns the label under which Layers are stored.

- XCAFDoc_LayerTool::ShapeTool

```
const Handle(XCAFDoc_ShapeTool)& XCAFDoc_LayerTool::ShapeTool()
```

Purpose: Returns the internal XCAFDoc_ShapeTool tool.

Methods for Layer table management

- XCAFDoc_LayerTool::IsLayer

```
Standard_Boolean XCAFDoc_LayerTool::IsLayer(const TDF_Label& lab)
const
```

Purpose: Returns True if a label belongs to the Layertable and is a Layer definition.

- XCAFDoc_LayerTool::GetLayer

```
Standard_Boolean XCAFDoc_LayerTool::GetLayer (const TDF_Label&
lab,
TCollection_ExtendedString& aLayer) const
```

Purpose: Returns a Layer defined by label <lab>. Returns False if the label is not in Layertable or does not define <aLayer>.

- XCAFDoc_LayerTool::FindLayer

```
Standard_Boolean XCAFDoc_LayerTool::FindLayer (
const TCollection_ExtendedString& aLayer,
TDF_Label& lab) const
```

Purpose: Finds a Layer definition in theLayertable and returns its label if found. Returns False if the Layer is not found in the Layertable.

```
TDF_Label XCAFDoc_LayerTool::FindLayer(
const TCollection_ExtendedString& aLayer) const
```

Purpose: Finds a Layer definition in the Layertable and returns its label if found (or the Null label otherwise).

- XCAFDoc_LayerTool::AddLayer

```
TDF_Label XCAFDoc_LayerTool::AddLayer(
const TCollection_ExtendedString& aLayer) const
```

Purpose: Adds a Layer definition to the Layertable and returns its label (returns the existing label if the same Layer is already defined).

- XCAFDoc_LayerTool::RemoveLayer

```
void XCAFDoc_LayerTool::RemoveLayer(const TDF_Label& lab) const
```

Purpose: Removes a Layer from the Layertable.

- XCAFDoc_LayerTool::GetLayerLabels

```
void XCAFDoc_LayerTool::GetLayerLabels(TDF_LabelSequence& Labels)
const
```

Purpose: Returns a sequence of Layers currently stored in the Layertable.

- XCAFDoc_LayerTool::SetLayer

```
void XCAFDoc_LayerTool::SetLayer(const TDF_Label& L,
const TDF_Label& LayerL,
const Standard_Boolean shapeInOneLayer) const
```

Purpose: Sets a link from label <L> to a Layer defined by <LayerL>, an optional parameter <shapeInOneLayer> shows whether a shape could be in a number of layers or only in one.

```
void XCAFDoc_LayerTool::SetLayer(const TDF_Label& L,
const TCollection_ExtendedString& aLayer,
const Standard_Boolean shapeInOneLayer) const
```

Purpose: Sets a link from label <L> to Layer <aLayer> in the Layertable. Adds <aLayer> as necessary; an optional parameter <shapeInOneLayer> shows whether a shape could be in a number of layers or only in one.

- XCAFDoc_LayerTool::UnSetLayers

```
void XCAFDoc_LayerTool::UnSetLayers(const TDF_Label& L) const
```

Purpose: Removes a link from label <L> to all layers.

- XCAFDoc_LayerTool::UnSetOneLayer

```
Standard_Boolean XCAFDoc_LayerTool::UnSetOneLayer(
const TDF_Label& L,
const TCollection_ExtendedString& aLayer) const
```

Purpose: Removes a link from label <L> and Layer <aLayer>. Returns FALSE if no such layer exists.

- XCAFDoc_LayerTool::IsSet

```
Standard_Boolean XCAFDoc_LayerTool::IsSet(const TDF_Label& L,
const TCollection_ExtendedString& aLayer) const
```

Purpose: Returns True if label <L> has a Layer associated with the <aLayer>.

- XCAFDoc_LayerTool::GetLayers

```
Standard_Boolean XCAFDoc_LayerTool::GetLayers(const TDF_Label& L,
Handle(TColStd_HSequenceOfExtendedString)& aLayerS)
```

Purpose: Returns a sequence of strings <aLayerS> that are associated with label <L>.

```
Handle(TColStd_HSequenceOfExtendedString)
XCAFDoc_LayerTool::GetLayers(
const TDF_Label& L)
```

Purpose: Returns a sequence of strings that are associated with label <L>.

- XCAFDoc_LayerTool::GetShapesOfLayer

```
void XCAFDoc_LayerTool::GetShapesOfLayer(const TDF_Label& layerL,
TDF_LabelSequence& ShLabels) const
```

Purpose: Returns a sequence of shape labels that are assigned with layers to <ShLabels>.

- XCAFDoc_LayerTool::IsVisible

```
Standard_Boolean XCAFDoc_LayerTool::IsVisible (
const TDF_Label& layerL) const
```

Purpose: Return TRUE if a layer is visible, FALSE if it is invisible.

- XCAFDoc_LayerTool::SetVisibility

```
void XCAFDoc_LayerTool::SetVisibility (const TDF_Label& layerL,
const Standard_Boolean isvisible) const
```

Purpose: Set the visibility of a layer. If a layer is invisible when on its layer will set UAttribute with a corresponding GUID.

Methods for assignment of Layers to shapes in the Shapes section

- XCAFDoc_LayerTool::SetLayer

```
Standard_Boolean XCAFDoc_LayerTool::SetLayer(const TopoDS_Shape&
Sh,
const TDF_Label& LayerL,
const Standard_Boolean shapeInOneLayer)
```

Purpose: Sets a link from a label containing shape <Sh> with a layer situated at label <LayerL>. An optional parameter <shapeInOneLayer> shows whether a shape could be in a number of layers or only in one. Returns FALSE if no such shape <Sh> or label <LayerL> exists.

```
Standard_Boolean XCAFDoc_LayerTool::SetLayer(const TopoDS_Shape&
Sh,
const TCollection_ExtendedString& aLayer,
const Standard_Boolean shapeInOneLayer)
```

Purpose: Sets a link from a label containing shape <Sh> with layer <aLayer>. Adds <aLayer> to LayerTable if necessary. An optional parameter <shapeInOneLayer> shows whether a shape could be in a number of layers or only in one. Returns FALSE if no such shape <Sh> exists.

- XCAFDoc_LayerTool::UnSetLayers

```
Standard_Boolean XCAFDoc_LayerTool::UnSetLayers(const
TopoDS_Shape& Sh)
```

Purpose: Removes a link between shape <Sh> and all Layers at the LayerTable. Returns FALSE if no such shape <Sh> exists in XCAF Document.

- XCAFDoc_LayerTool::UnSetOneLayer

```
Standard_Boolean XCAFDoc_LayerTool::UnSetOneLayer(
const TopoDS_Shape& Sh,
const TCollection_ExtendedString& aLayer)
```

Purpose: Removes a link between shape <Sh> and layer <aLayer>. Returns FALSE if no such layer <aLayer> or shape <Sh> exists.

- XCAFDoc_LayerTool::IsSet

```
Standard_Boolean XCAFDoc_LayerTool::IsSet(const TopoDS_Shape& Sh,
```

Purpose: Returns True if shape <Sh> has a Layer associated with the <aLayer>.

- XCAFDoc_LayerTool::GetLayers

```
Standard_Boolean XCAFDoc_LayerTool::GetLayers(const TopoDS_Shape&
Sh,
Handle(TColStd_HSequenceOfExtendedString)& aLayerS)
```

Purpose: Returns a sequence of strings <aLayerS> associated with shape <Sh>.

```
Handle(TColStd_HSequenceOfExtendedString)
XCAFDoc_LayerTool::GetLayers(
const TopoDS_Shape& Sh)
```

Purpose: Returns a sequence of strings associated with shape <Sh>.

3.12. Class XCAFDoc_GraphNode;

3.12.1. General description

Purpose: Attribute containing a sequence of father and child labels.

Creates and allows to work with Graph in XCAFDocument.

3.12.2. Methods

The methods are the following:

Constructors

```
XCAFDoc_GraphNode()
```

Purpose: Empty constructor.

Class methods working on the node

- XCAFDoc_GraphNode::Find

```
Standard_Boolean XCAFDoc_GraphNode::Find(const TDF_Label& L,
Handle(XCAFDoc_GraphNode)& G)
```

Purpose: Shortcut to search a Graph node attribute with a default GraphID. Returns true if found.

- XCAFDoc_GraphNode::GetDefaultGraphID

```
const Standard_GUID&  XCAFDoc_GraphNode::GetDefaultGraphID()
```

Purpose: Returns a default Graph ID. This ID is used by the <Set> method without an explicit tree ID.

- XCAFDoc_GraphNode::Set

```
Handle(XCAFDoc_GraphNode) XCAFDoc_GraphNode::Set(const TDF_Label&
L)
```

Purpose: Finds or Creates a GraphNode attribute on the label <L> with a default Graph ID, returned by the method GetDefaultGraphID(). Returns the created/found GraphNode attribute.

```
Handle(XCAFDoc_GraphNode) XCAFDoc_GraphNode::Set (const TDF_Label&
L,
const Standard_GUID& explicitID)
```

Purpose: Finds or Creates a GraphNode attribute on the label <L>, with an explicit tree ID. <ExplicitGraphID> is the ID returned by the TDF_Attribute::ID() method. Returns the found/created GraphNode attribute.

Instance methods

- XCAFDoc_GraphNode::SetFather

```
Standard_Integer XCAFDoc_GraphNode::SetFather(
const Handle(XCAFDoc_GraphNode)& F)
```

Purpose: Sets GraphNode <F> as the father of this attribute and returns an index of <F> in a sequence containing Father GraphNodes. Returns an index of <F> from GraphNodeSequnece.

- XCAFDoc_GraphNode::SetChild

```
Standard_Integer XCAFDoc_GraphNode::SetChild(
const Handle(XCAFDoc_GraphNode)& Ch)
```

Purpose: Sets GraphNode <Ch> as a child of this attribute and returns an index of <Ch> in a sequence containing Children GraphNodes. Returns an index of <Ch> from GraphNodeSequnece.

- XCAFDoc_GraphNode::UnSetFather

```
void XCAFDoc_GraphNode::UnSetFather(const
Handle(XCAFDoc_GraphNode)& F)
```

Purpose: Removes <F> from Fathers GraphNodeSequence and removes the link between the father and the child.

```
void XCAFDoc_GraphNode::UnSetFather(const Standard_Integer Findex)
```

Purpose: Removes Father GraphNode by index from Fathers GraphNodeSequence and removes the link between the father and the child.

- XCAFDoc_GraphNode::UnSetFatherlink

```
void XCAFDoc_GraphNode::UnSetFatherlink(
const Handle(XCAFDoc_GraphNode)& F)
```

Purpose: Removes the link between the father and the child.

- XCAFDoc_GraphNode::UnSetChild

```
void XCAFDoc_GraphNode::UnSetChild(const
Handle(XCAFDoc_GraphNode)& Ch)
```

Purpose: Removes <Ch> from GraphNodeSequence and removes the link between the father and the child.

```
void XCAFDoc_GraphNode::UnSetChild(const Standard_Integer Chindex)
```

Purpose: Removes Child GraphNode by index from Children GraphNodeSequence and removes the link between the father and the child.

- XCAFDoc_GraphNode::UnSetChildlink

```
void XCAFDoc_GraphNode::UnSetChildlink(
```

```
const Handle(XCAFDoc_GraphNode)& Ch)
```

Purpose: Removes the link between the father and the child.

- XCAFDoc_GraphNode::GetFather

```
Handle(XCAFDoc_GraphNode) XCAFDoc_GraphNode::GetFather(
const Standard_Integer Findex) const
```

Purpose: Returns GraphNode by index from GraphNodeSequence.

- XCAFDoc_GraphNode::GetChild

```
Handle(XCAFDoc_GraphNode) XCAFDoc_GraphNode::GetChild(
const Standard_Integer Chindex) const
```

Purpose: Returns GraphNode by index from GraphNodeSequence.

- XCAFDoc_GraphNode::FatherIndex

```
Standard_Integer XCAFDoc_GraphNode::FatherIndex(
const Handle(XCAFDoc_GraphNode)& F) const
```

Purpose: Returns the index of <F>, or zero if there is no such Graphnode.

- XCAFDoc_GraphNode::ChildIndex

```
Standard_Integer XCAFDoc_GraphNode::ChildIndex(
const Handle(XCAFDoc_GraphNode)& Ch) const
```

Purpose: Returns the index of <Ch>, or zero if there is no such Graphnode.

- XCAFDoc_GraphNode::IsFather

```
Standard_Boolean XCAFDoc_GraphNode::IsFather(
const Handle(XCAFDoc_GraphNode)& Ch) const
```

Purpose: Returns True if this attribute is the father of <Ch>.

- XCAFDoc_GraphNode::IsChild

```
Standard_Boolean XCAFDoc_GraphNode::IsChild(
const Handle(XCAFDoc_GraphNode)& F) const
```

Purpose: Returns True if this attribute is a child of <F>.

- XCAFDoc_GraphNode::NbFathers

```
Standard_Integer XCAFDoc_GraphNode::NbFathers() const
```

Purpose: Returns the Number of Father GraphNodes.

- XCAFDoc_GraphNode::NbChildren

```
Standard_Integer XCAFDoc_GraphNode::NbChildren() const
```

Purpose: Returns the Number of Children GraphNodes.

3.13. Package methods

Purpose: Definition of GUIDs

Methods:

- XCAFDoc::AssemblyGUID

```
Standard_GUID XCAFDoc::AssemblyGUID ()
```

Purpose: Returns a GUID for UAttribute identifying the assembly

- XCAFDoc::ShapeRefGUID

```
Standard_GUID XCAFDoc::ShapeRefGUID ()
```

Purpose: Returns a GUID for TreeNode representing an assembly link.

- XCAFDoc::ColorRefGUID

```
Standard_GUID XCAFDoc::ColorRefGUID (const XCAFDoc_ColorType type)
```

Purpose: Returns a GUID for TreeNode representing specified types of colors.

- XCAFDoc::LayerRefGUID

```
Standard_GUID XCAFDoc::LayerRefGUID ()
```

Purpose: Returns a GUID from Standard.

- XCAFDoc::InvisibleGUID

```
Standard_GUID XCAFDoc::InvisibleGUID ()
```

Purpose: Returns a GUID from Standard;

- XCAFDoc::ExternRefGUID

```
Standard_GUID XCAFDoc::ExternRefGUID ()
```

Purpose: Returns a GUID for UAttribute identifying an external reference.