



## **ENF Editor Manual**

August 2020

Elysium Co. Ltd.

# Contents

1. About This Manual .....	1
2. Introduction .....	2
2.1. What is ENF Editor ? .....	2
2.2. About ENF File .....	2
3. Requirements .....	4
3.1. Supported OS .....	4
3.2. Supported Ruby Script Version .....	4
4. Programming Guide .....	5
4.1. Introduction .....	5
4.2. Getting Started .....	5
4.3. Basic Topics .....	8
4.4. Advanced Topics .....	11
5. ENF Structure .....	12
5.1. Component .....	12
5.2. Brep Elements .....	12
5.3. Display Properties of Brep Elements .....	14
5.4. System Property .....	16
5.5. User Property .....	16
5.6. ENF Version Compatibility .....	17
6. CAD and Translator Specification .....	18

# 1. About This Manual

This document presents basic information on how to write and execute an ENF Editor script. This document consists of the following chapters:

- Introduction - describes what you can do with ENF Editor.
- Requirements - describes the requirements of ENF Editor.
- Programming Guide - describes how to write ENF Editor Ruby scripts.
- ENF Structure - describes the format of ENF file.
- CAD and Translator Specification - describes the specification of CAD and ASFALIS Adapter (CAD to ENF and ENF to CAD).

## 2. Introduction

### 2.1. What is ENF Editor ?

We are now ready to offer you the new ASFALIS component **ENF Editor**. **ENF Editor API** along with its **Script** allow users to edit various types of attributes and elements of 3D CAD Data such as layer, color, user property and so on. The API is based on a simple, powerful and user-friendly scripting language **Ruby**.

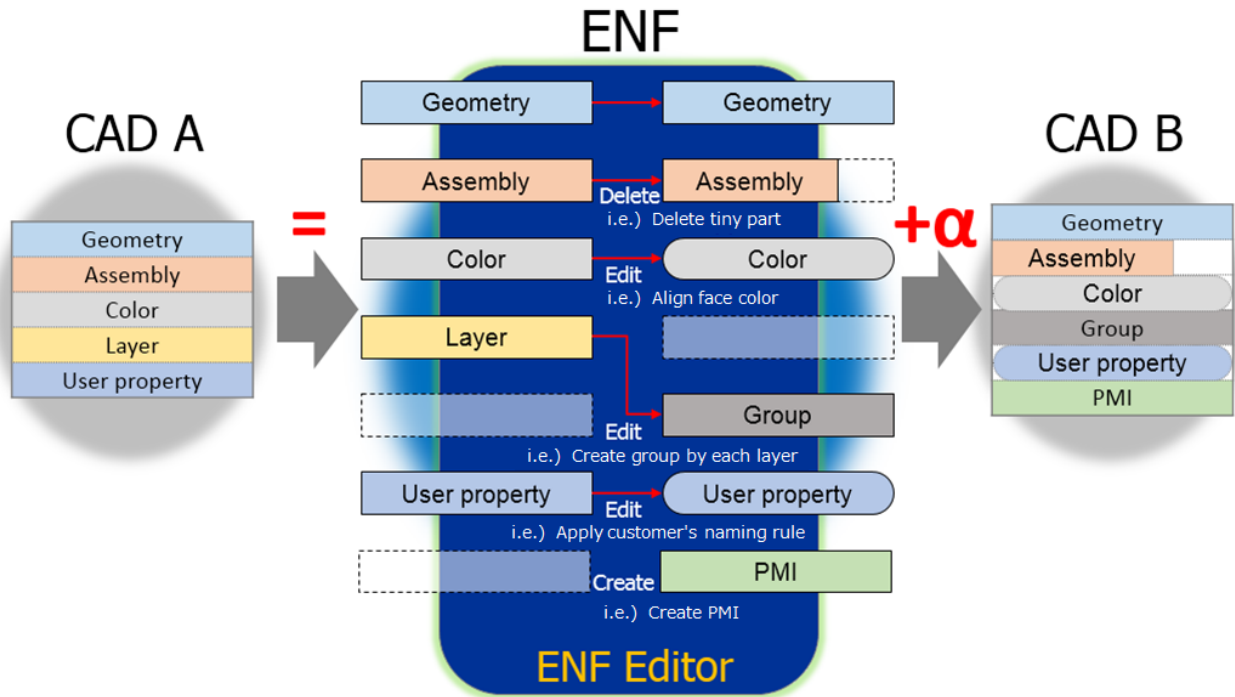


Figure 1. ENF Editor

### 2.2. About ENF File

**ENF**, an acronym for Elysium Neutral File, is a unique intermediate file format that acts as the core of **ASFALIS**. **ENF** holds not only geometry information, but also 3D annotation and attribute information, as they are indispensable for utilization of 3D Data in the manufacturing process.

ENF can handle various elements such as:

- Assembly Structure (Assembly, Part, Instance, Component)
- Brep (Volume, Face, Edge, etc.)
- Polygonrep (MultiTriangle, MultiPolyline, MultiPoint)
- PMI (Note, Dimension, GDT, etc.)
- Camera (Camera, CameraFolder)
- Attribute (UserAttribute, SystemAttribute, LayerFilter, FeatureGroup, Axis, etc.)

- Display Property (Color, Layer, Display, etc.)



In current version, ENF Editor API supports only Brep and Component (i.e., Assembly and Part). Other elements will be supported in the future release. Please refer to **ENF Editor API Reference Guide** for more details on API.

## 3. Requirements

### 3.1. Supported OS

- Windows 8.1 64bit
- Windows 10 64bit

### 3.2. Supported Ruby Script Version

- Ruby 2.6

## 4. Programming Guide

In this chapter, you can learn how to execute, write and debug ENF Editor scripts.

### 4.1. Introduction

ENF Editor script is based on scripting language [Ruby](#). Enf Editor is provided in gem format, and integrated into Ruby as module "**EnfEditor**". Therefore, users can take advantage of both Enf Editor's 3D CAD Data interface and Ruby's standard library such as *Math*, *CSV*, *Date* and so on.

Please refer to **ENF Editor API Reference Guide** for more details on API. In addition, please note that grammar of [Ruby](#) itself is not stated in this document. If you are not familiar with [Ruby](#), visit [ruby-lang.org](http://ruby-lang.org) to get access to informative documents.

### 4.2. Getting Started

#### 4.2.1. Executing sample script from batch file

To get started, procedures to execute pre-installed sample scripts are described in this section. After installation of ASFALIS components, sample scripts and a batch file to run scripts should exist in the following path.

Contents	Path
sample scripts	<InstallFolder>\module\tools\win\elyEnfEditor\scripts\sample
execution batch file	<InstallFolder>\module\tools\win\elyEnfEditor\dev\test.bat
dev folder	<InstallFolder>\module\tools\win\elyEnfEditor\dev



The default ASFALIS installation path <InstallFolder> is "C:\ELYSIUM\ASFALIS\_Component\EX8\_2".

Here are the steps to run ENF Editor scripts.

1. Copy dev folder to your local work folder.
2. Open "test.bat" and specify variables such as install folder and output filenames. See "dev\readme.txt" for details.
3. Open config\test.cfg and specify the hostname and the port number of Elysium license server.
4. Run "test.bat".  
Result ENF file and log files should be created in the specified working folder.

In order to change executing script and input ENF file, edit "test.bat" as follows.

```
set EE_SCRIPT=%~dp0..\scripts\sample\edit_face_color_by_layer.rb
set EE_INPUTENF=%~dp0model\car.enf
```

The script "edit\_face\_color\_by\_layer.rb" modifies face color based on layer number. The result ENF file should look as follows.

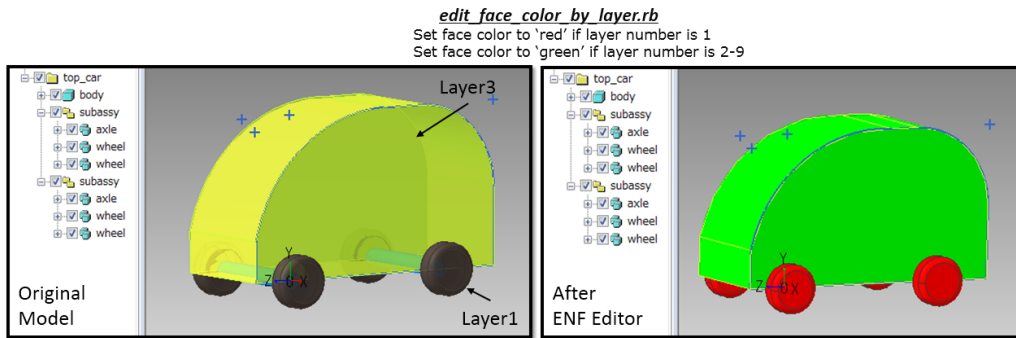


Figure 2. Result of executing edit\_face\_color\_by\_layer.rb

Face color is set to red when layer number is 1, and set to green when layer number is between 2 to 9; otherwise, face color is set to blue.

#### 4.2.2. Writing script from scratch

To support writing a script from scratch, a template script ("template.rb" under "sample scripts" folder) is provided. Common operations such as opening a session, reading ENF file and writing ENF file are already written in the template script. Please copy and rename this template script to begin with. A completed script file can be executed using "test.bat".

Below is an extraction of the main function "execute" from "template.rb". Follow the comments starting with "#" to understand what is being done each line.



```

require 'enfeditor'
def execute(argv)
  # Write message to log
  @logger.info "Processing ENF Editor.."
  # Open session
  EnfEditor::EESession.open(argv) do |session|
    # Get input and output ENF file path
    input_enf = session.parameter['inputfile']
    output_enf = session.parameter['outputfile']
    # Read ENF file
    session.read_enf(input_enf)
    # Execute something (write your code in execute_impl method)
    execute_impl(session)
    # Write ENF file
    session.write_enf(output_enf)
  end
rescue => ex
  # Log error if exception has been raised
  @logger.error "Exception has been raised."
  @logger.error ex
  # Exit with error code
  raise SystemExit.new(99)
else
  # Log for successful case
  @logger.info "Complete."
end

```

The basic flow of an ENF Editor script is as follows.

1. Load "enfeditor" library.
2. Open EESession.
3. Read ENF file.
4. Execute something.
5. Write ENF file.

First, you must require module **"EnfEditor"** to use ENF Editor APIs. *EESession::open* performs various operations such as initializing API and log files, parsing arguments, checking license and so on.

After *EESession::open* is called, all ENF Editor APIs are ready to be called. Then call *EESession::read\_enf* to load ENF file into memory. At the end, call *EESession::write\_enf* to write an ENF file. Exported ENF file will become an output of the Enf Editor component, and will be passed on to the next ASFALIS component such as ENF to CAD Adapters.

### 4.2.3. Edit ENF data using ENF Editor API

The template script "template.rb" virtually performs nothing: reads the input ENF file and writes it to a different file. This is because content of function "execute\_impl" is commented out. Uncomment the lines in execute\_impl as follows and then all component names should be logged in the ASFALIS log file.

```
def execute_impl(session)
  # Write your code here
  session.model.components.each do |compo|
    unless compo.name.nil?
      @logger.info "Component name is .. #{compo.name}"
    end
  end
end
```

Here is the detail flow of how to traverse all components and log its name by using ENF Editor API. For more detailed information of classes such as EESession, and methods such as EESession::model, please refer to **ENF Editor API Reference Guide**.

1. Get EEModelSession instance from EESession class. Use EESession::model method (session.model).
2. Get the EEComponent instance array which contains all components. Use EEModelSession::components method (session.model.components).
3. Traverse each EEComponent instance. Use Array::each method, which is a standard method in Ruby (session.model.components.each).
4. Get component name from each EEComponent instance. Use EEComponent::name method (compo.name).

The sample presented in this section is only one example. Please refer to other samples under the "sample scripts" folder to get an idea of what ENF Editor is capable of.

## 4.3. Basic Topics

This section describes some basic topics, such as logging and handling errors. It is recommended to understand this section before writing scripts.

### 4.3.1. Module and Classes

All ENF Editor classes belong to module "**EnfEditor**". All classes related to ENF elements begin with a prefix "EE", e.g., EEComponent and EEBrepFace. These classes are all subclasses of class EEBase.

### 4.3.2. Arguments and Parameters

Arguments given to the Ruby script are for internal use and must be passed to `EESession::open`.

```
EESession::open(argv)
```

Input and output ENF file paths can be accessed using `EESession::parameter(Hash)`. By default, input and output ENF files are the only parameters available.

```
puts session.parameter['inputfile']    # => 'c:/work/input.enf'
puts session.parameter['outputfile']   # => 'c:/work/output.enf'
```

### 4.3.3. Logging

All messages written to STDOUT (standard output) is logged to the ASFALIS log file. Class **"Logger"** is useful for writing logs.

```
require 'logger'
def initialize
  @logger = Logger.new(STDOUT)
end

def execute(argv)
  @logger.info 'Processing ENF Editor'
  EESession::open(argv) do |session|
    session.read_enf(input_enf)
    num_faces = session.model.faces.size
    num_edges = session.model.faces.size
    @logger.info "Number of faces : #{num_faces}"
    @logger.info "Number of edges : #{num_edges}"
  end
end
```

### 4.3.4. Exceptions

API throws *EEError* exception when an error occurs. Exception must be caught with "rescue", otherwise the ENF Editor process will exit with an error code. When catching an exception, it's also recommended to log error messages to understand what is happening. In order to force ENF Editor to exit with a certain error code, raise *SystemExit* with the error code. Error code passed to *SystemExit* will be written in the ASFALIS log file.

```
begin
  # method read_enf throw EError when input file does not exist.
  session.read_enf('no_exist_file')
rescue => ex
  # log error info
  @logger.error ex
  # raise error to exit process with error code.
  raise SystemExit.new(99)
end
```

### 4.3.5. Error codes

The following table shows the relation between Error code generated by SystemExit exception and that of ENF Editor component.

Script Error Code	ENF Editor Error Code	Description
0	0	Normal End
6	6	The specified input file is not found
Others	975	Failed to process ENF Editor

### 4.3.6. Debugging

If a script ends with an error code other than 0, take a look at the ASFALIS log file first. Error messages or call stack information in the log file should help you understand what is happening.

### 4.3.7. Encoding

All ENF Editor APIs require UTF-8 encoding, i.e., String parameters must be passed in UTF-8. Recommended script encoding is UTF-8. In order to encode the script as UTF-8, save the file in UTF-8 format and write a magic comment in the first line as follows:

```
# -*- encoding: UTF-8 -*-
```

In order to read non-UTF-8 encoded files, string should be converted into UTF-8. External and internal encoding is set to UTF-8 by default.

```
# open shift-jis file as UTF-8
File.open(sjis_file, 'r:windows-31j:utf-8') do |f|
  # ...
end
```

## 4.4. Advanced Topics

This section describes some tips for advanced developers.

### 4.4.1. ENF object validity

Use "null?" method to check object validity. For example, after a user property is deleted, it cannot be accessed anymore. In such a case, "null?" returns true.

```
component.user_properties.each do |user_prop|
  user_prop.null?      # => false
  user_prop.delete!    # delete user property
  user_prop.null?      # => true
end
```

### 4.4.2. Equality of ENF objects

Use "eql?" or "==" to check equality of ENF objects. In the following example, "edge\_org" and "edge\_from\_curve" point to the same EEBrepEdge object in ENF. By calling "eql?" or "==", internal ENF object IDs are compared and returns true if IDs are identical. On the other hand, "equal?" method returns false because compared objects are different objects in the Ruby world.

```
edge_from_curve = edge_org.curve.edge
edge_org.equal?(edge_from_curve) # => false
edge_org.eql?(edge_from_curve)   # => true
edge_org == edge_from_curve      # => true
```

### 4.4.3. Update progress

In order to update Enf Editor component's progress (when used from ASFALIS Desktop or ASFALIS TransServer), use method "**EESession::process\_meter\_main**". Valid value ranges from 0 to 100.

```
session.process_meter_main = 0      # 0 percent
session.read_enf(input_enf)
session.process_meter_main = 30     # 30 percent
# write your code here...
session.write_enf(output_enf)
session.process_meter_main = 100    # 100 percent
```

## 5. ENF Structure

This chapter describes the data structure of ENF.

### 5.1. Component

**Component** is either an assembly or part. Part component is a terminal component of the assembly structure and contains brep and/or polygonrep. Assembly is the parent of other components and acts as a collection of multiple components.

Looking at the example below, the car consists of two assembly components and three part components. Although there are four instances of the "wheel" part, they are shared and only **ONE** "wheel" component exists.

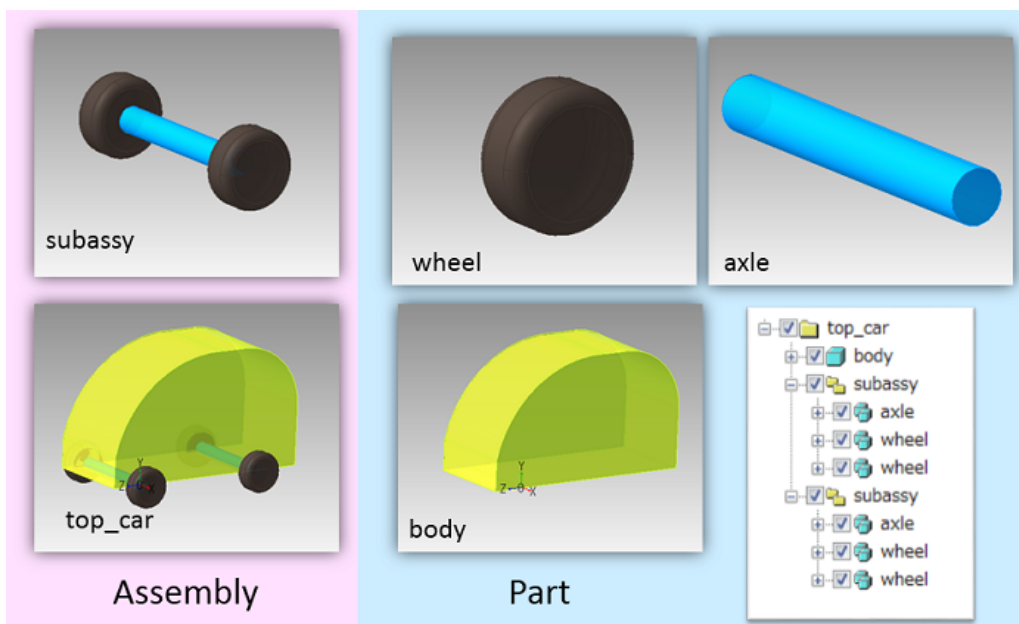


Figure 3. Example of assembly and part component

### 5.2. Brep Elements

Brep is a child element of part. Brep consists of topological and geometrical entities as shown below.

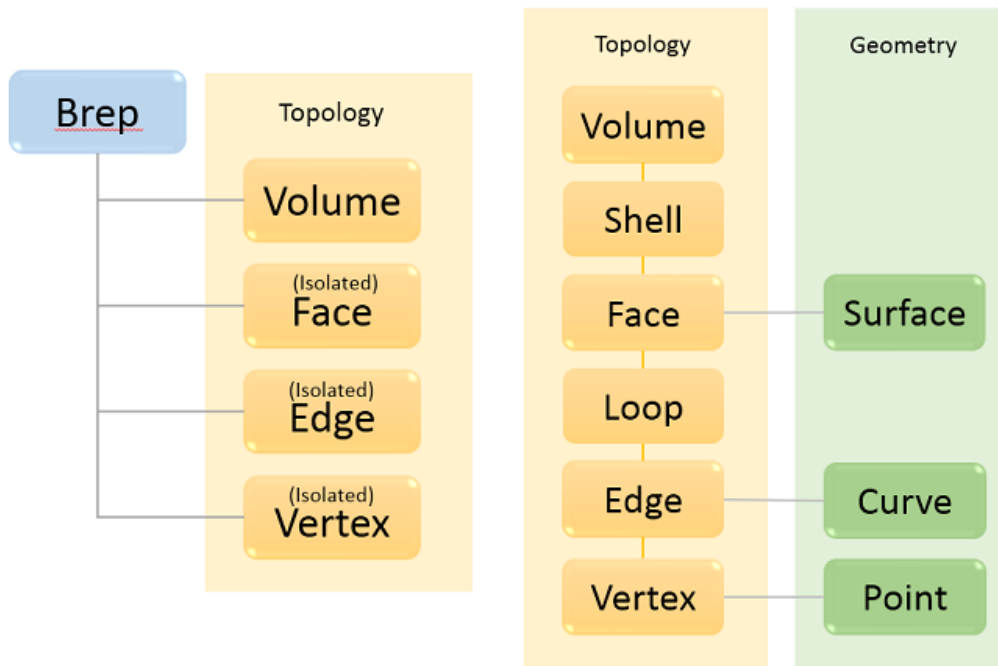


Figure 4. Brep Tree Structure

#### 1. Volume

Volume consists of shells. It can be a closed volume (Solid) or an open volume (Sheet).

- Class : EnfEditor::EEBrepVolume

#### 2. Shell

Shell consists of faces.

- Class : EnfEditor::EEBrepShell

#### 3. Face

Face is a topological entity for surface and it consists of loops and surface geometry data. It can be either a face under a volume or an isolated face.

- Class : EnfEditor::EBrepFace

#### 4. Loop

Loop consists of edges.

- Class : EnfEditor::EEBrepLoop

#### 5. Edge

Edge is a topological entity for curve. It can be either an edge under a volume or an isolated edge.

- Class : EnfEditor::EEBrepEdge

#### 6. Vertex

Vertex is a topological entity for point. It can be either a vertex under a volume or an isolated vertex.

- Class : EnfEditor::EEBrepVertex

#### 7. Surface

Surface is a geometrical entity for face.

- Class : EnfEditor::EEBrepSurface

#### 8. Curve

Curve is a geometrical entity for edge.

- Class : EnfEditor::EEBrepCurve

#### 9. Point

Point is a geometrical entity for vertex.

- Class : EnfEditor::EEBrepPoint

## 5.3. Display Properties of Brep Elements

### 5.3.1. Layer

Layer number property is an integer property attached to an element. Valid range is from 0 to 2147483647. Supported elements are ...

- Volume
- Face
- Surface
- Edge
- Curve
- Vertex
- Point



Layer number of edge and vertex refers to that of corresponding curve and point, i.e., when layer number is set to an edge, layer number of corresponding curve is modified.



In ENF2 format, supported layer number range is from 0 to 255. When reading or writing an ENF2 file, layer number is cutoff to be between 0-255.

### 5.3.2. Color, Transparency

MaterialColor property represents color and transparency (alpha). It should be expressed by one of the following color types:

- RGB type - RGB only. Transparency is not defined.
- Alpha type - Transparency only. RGB is not defined.
- RGBA type - Both RGB and transparency.
- Material Strength type - RGBA and material properties as scalar.
- Material RGBA type - Material properties as RGBA.



Valid range of each r,g,b,a value is from 0.0 to 1.0 with double precision. For example, (r,g,b) = (1.0, 0.0, 0.0) represents color "**Red**". Alpha is an opacity value, i.e., 0.0 means completely transparent while 1.0 means completely opaque. Supported elements are ...

- Volume
- Face
- Surface
- Edge
- Curve
- Vertex
- Point



Material color of edge and vertex refers to that of corresponding curve and point, i.e., when color is set to an edge, color of corresponding curve is modified.



In ENF2 format, the precision of color is 1/100 for each r,g,b value. When reading or writing an ENF2 file, each r,g,b value is rounded off to the nearest N/100 value. Additionally, "Material Strength" and "Material RGBA" type is not supported in ENF2 format.

### 5.3.3. Display

Display property is a boolean property. It represents visible or invisible status of an element. Supported elements are:

- Volume
- Face
- Surface
- Edge
- Curve
- Vertex
- Point



Display property of edge and vertex refers to that of corresponding curve and point, i.e., when display property is set to an edge, display property of corresponding curve is modified.

### 5.3.4. Name

Name property is a String property. Supported elements are:

- Volume



In ENF2 format, the maximum length for volume name is 80.

## 5.4. System Property

System property is a key-value property specifically for components. It has reserved keys such as ...

- BinName
- ChangeHistory
- ConfigName
- Description
- PartComment
- PartDefinition
- PartName
- PartNumber
- PartVersion
- PartRevision
- PartSource
- PartLayer
- Name
- NFName - Native file name
- TFName - Target file name
- Material



Not all CAD systems support all the system properties listed above. For example, BinName and ConfigName are properties only for IDEAS.

## 5.5. User Property

User Property is a key-value property attached to an element. It consists of ...

- Key - user key string
- Value - user value string
- Type - user value type
- Subtype - subtype string (optional)

Available Types are ...

- BoolChar - one character value such as "c".
- Bool - boolean value such as "1" or "0".
- Integer - integer value such as "123".
- Real - double value such as "4.56".
- Section - for internal use only. Do-not-use.
- Text - text value.

Supported owner elements are ...

- Component
- Volume
- Face
- Edge
- Vertex

## 5.6. ENF Version Compatibility

The latest version of ENF file format is Ver3.3 which is used for exporting by default. All major and minor versions of ENF is supported in Enf Editor, but it is recommended to use the latest version. This is because some attributes and elements could be lost using older versions.

## 6. CAD and Translator Specification

When using ENF Editor, it is very important to understand the specification of CAD and translators (both CAD to ENF and ENF to CAD direction).

Please refer to 'cad\_spec/\*\_spec.pdf' for details.

All rights reserved by Elysium or the original author of this material. The content may not be edited, reproduced, distributed, transmitted, displayed, published, broadcast, sold or lent without the prior permission of the author.