



ENF Editor Manual

2020年 8月

株式会社エリジョン

目次

1. このマニュアルについて	1
2. はじめに	2
2.1. ENF Editor とは	2
2.2. ENF ファイルとは	2
3. システム要件	4
3.1. OS サポートバージョン	4
3.2. Ruby スクリプトサポートバージョン	4
4. スクリプティングガイド	5
4.1. ENF Editor 概要	5
4.2. ENF Editor 入門	5
4.3. その他の重要項目	8
4.4. 発展的な項目	11
5. ENF 構造	12
5.1. コンポーネント	12
5.2. Brep 要素	12
5.3. Brep 要素に対応する表示属性	14
5.4. システム属性	16
5.5. ユーザ属性	16
5.6. ENF バージョンの互換性	17
6. CAD および変換仕様	18

1. このマニュアルについて

本マニュアルはENF Editor スクリプトを記述する際の導入として最低限必要な各種情報を掲載しています。本マニュアルは以下の内容から構成されています。

- はじめに- ENF Editor とENF フォーマットの概要について説明します。
- システム要件- ENF Editor のシステム要件・動作バージョンを紹介します。
- スクリプティングガイド- ENF Editor で動作するRuby スクリプトの記述方法を説明します。
- ENF 構造- ENF フォーマットの構造を説明します。
- CAD および変換仕様- CAD およびASFALIS アダプター(CAD to ENF およびENF to CAD) の仕様を紹介します。

2. はじめに

2.1. ENF Editor とは

ENF Editor API とスクリプトを用いることで、3D CAD データに含まれる様々な要素やレイヤ、色、ユーザ属性などの属性を自由に取り出し、編集することができます。ENF Editor API はシンプルさと高い生産性を備えており、自然に読み書きができるオブジェクト指向スクリプト言語であるRuby 上で提供しているため、強力なAPI の機能を簡単に利用することができます。

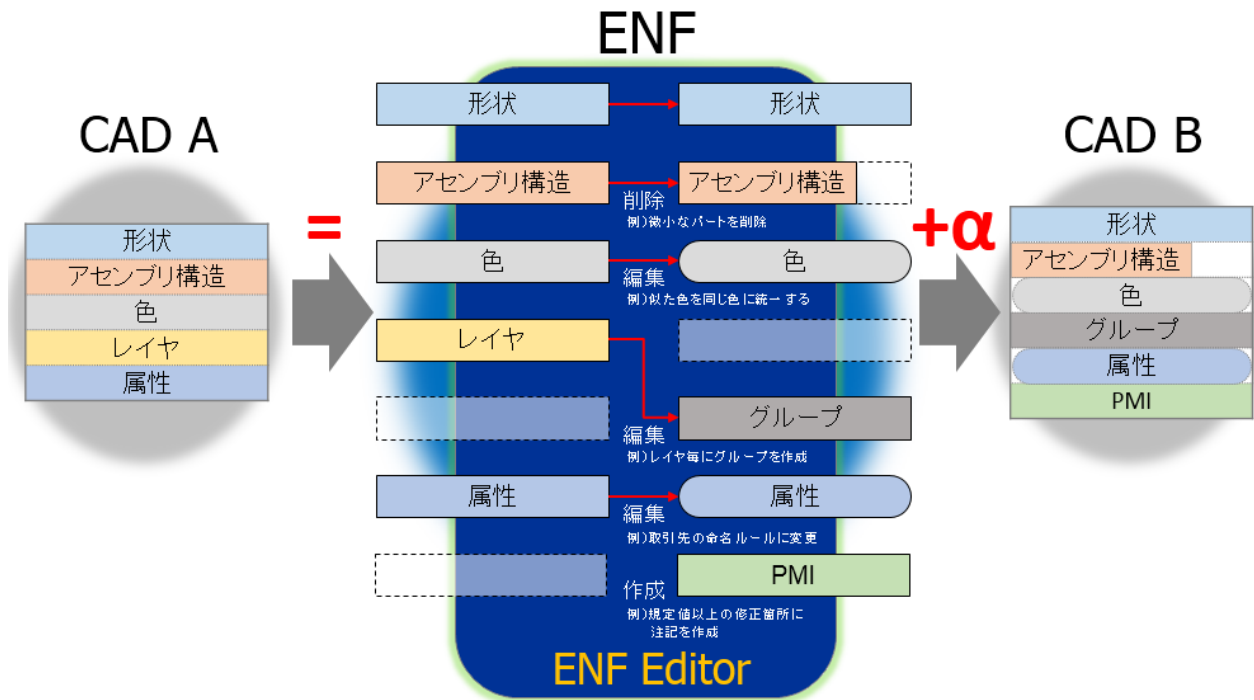


図 1. ENF Editor Manual, リリースEX8.2

2.2. ENF ファイルとは

ENF (Elysium Neutral File) はASFALIS の中核をなすエリジオン独自の間ファイルフォーマットです。ENF は形状要素だけではなく、3D データを製造プロセス全体で活用する際に非常に重要な情報となる3D 注記や属性も保持することができます。

具体的には、以下の要素を扱うことができます。

- アセンブリ (アセンブリ、パート、インスタンス、コンポーネント)
- Brep (ボリューム、フェース、エッジ... 等)
- ポリゴン (ポリゴンフェース、ポリライン、ポリゴン頂点)
- PMI (注記、寸法、幾何公差... 等)
- 視点情報 (カメラ、カメラフォルダ)
- 属性 (ユーザ属性、システム属性、レイヤフィルタ、フィーチャグループ、座標系... 等)

- 表示要素 (色、レイヤ、表示/非表示... 等)



ENF Editor API は現バージョンにおいて Brep およびコンポーネントをサポートします。その他の要素については、次バージョン以降にてサポートする予定です。サポートされている要素の詳細に関しては別冊の『ENF Editor API Reference Guide』をご参照ください。

3. システム要件

3.1. OS サポートバージョン

- Windows 8.1 64bit
- Windows 10 64bit

3.2. Ruby スクリプトサポートバージョン

- Ruby 2.6

4. スクリプティングガイド

この章では、ENF Editor スクリプトの実行、記述およびデバッグの方法について紹介します。

4.1. ENF Editor 概要

ENF Editor スクリプトはオブジェクト指向スクリプト言語である [Ruby](#) をベースにしています。

enfeditor のライブラリは gem 形式で提供しているため、ENF ファイル内の様々な 3D CAD 情報にアクセスすることのできる **enfeditor** モジュールの機能と、*Math*, *CSV*, *Date* 等の標準ライブラリなど Ruby が持つ様々な機能を自由に組み合わせてスクリプトを記述することができます。

ENF Editor API の詳細については、**ENF Editor API Reference Guide** をご参照ください。なお、本マニュアルでは [Ruby](#) 自体の説明に関しては省略します。詳しい [Ruby](#) の説明については、ruby-lang.org 等をご参照ください。

4.2. ENF Editor 入門

4.2.1. サンプルスクリプトの実行

はじめに、事前に準備されているサンプルスクリプトを実行してみましょう。ASFALIS コンポーネントをインストールすると、下記のパスにサンプルスクリプトとスクリプト実行用のバッチファイルが保存されます。

内容	パス
サンプルスクリプト	<インストールフォルダ>\module\tools\win\elyEnfEditor\scripts
実行用バッチファイル	<インストールフォルダ>\module\tools\win\elyEnfEditor\dev\test.bat
dev フォルダ	<インストールフォルダ>\module\tools\win\elyEnfEditor\dev



<インストールフォルダ>の初期値は "C:\ELYSIUM\ASFALIS_Component\EX8_2" です。

以下がENF Editor スクリプトをバッチファイルから実行する際の流れです。

1. dev フォルダを任意のローカルフォルダにコピーします。
2. test.bat に記述されているインストールフォルダ、出力ファイル名等の情報をテキストエディタで編集します。詳細については dev\readme.txt をご参照ください。
3. config\test.cfg に記述されているエリジオンライセンスサーバーの情報をテキストエディタで編集します。
4. test.bat を実行します。
処理結果のENF ファイルとログファイルがtest.bat で指定したEE_WORKDIR(作業フォルダ) に出力されます。

test.bat では EnfEditor スクリプトと入力 ENF ファイルを以下の箇所指定しています。

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

今回実行した "edit_face_color_by_layer.rb" はレイヤ番号に基づきフェース色を変更するスクリプトです。実行結果の ENF ファイルは以下のようになります。

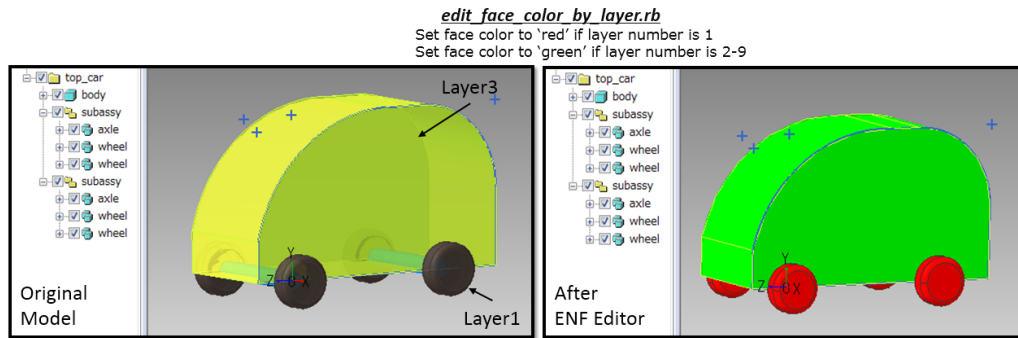


図 2. edit_face_color_by_layer.rb の実行結果

このスクリプトではレイヤ番号に基づきフェース色を変更します。レイヤ番号1 番はフェースを赤色に、2～9 番は緑色に、それ以外のレイヤは青色に変更します。

4.2.2. テンプレートスクリプトの内容確認

独自の ENF Editor スクリプトを書く際には、サンプルスクリプトフォルダの template.rb を利用してください。まず、template.rb を任意の場所にコピーし名前を変更します。次に、test.bat に記述されているスクリプトのパスを修正してください。この時点で test.bat を実行すると、スクリプトは ENF ファイルを読み込み、そのまま出力します。セッションの開始や ENF ファイルの入出力など、ENF Editor スクリプトを記述する上で必要な共通処理は template.rb 上に既に書かれていますので、ENF ファイルを編集する部分の記述に集中することができます。

下記は template.rb の最も重要な部分である "execute" メソッドです。スクリプトの説明として "#" から始まる行にコメントを記載していますので参考にしてください。


```

require 'enfeditor'
def execute(argv)
  # 実行中の旨メッセージをログに出力する
  @logger.info "Processing ENF Editor.."
  # セッションを開始する
  EnfEditor::EESession.open(argv) do |session|
    # ENF ファイルの入出力パスを取得する
    input_enf = session.parameter['inputfile']
    output_enf = session.parameter['outputfile']
    # ENF ファイルを読み込む
    session.read_enf(input_enf)
    # ENF Editor によるENF 編集を実行する(処理はexecute_impl メソッドに記述してく
    ださい)
    execute_impl(session)
    # ENF ファイルを書き出す
    session.write_enf(output_enf)
  end
rescue => ex
  # 例外が発生した場合、エラーをログに出力する
  @logger.error "Exception has been raised."
  @logger.error ex
  # エラーコード(この場合99) を出力し、終了する
  raise SystemExit.new(99)
else
  # 正常終了した場合、完了メッセージをログに出力する
  @logger.info "Complete."
end

```

ENF Editor スクリプトの標準的なフローは以下となります。

1. "enfeditor" ライブラリを読み込む
2. "EESession" セッションを開始する
3. ENF ファイルを読み込む
4. ENF 編集を実行する
5. ENF ファイルを書き出す

ENF Editor のAPI 群を利用するためには、最初に require メソッドを用いて **"enfeditor"** ライブラリを読み込む必要があります。*EESession::open* 部分でセッションを開始しますが、ここで API の初期化やログの準備、引数のパースやライセンスチェック等様々な処理を行います。セッションを開始しなければ、enfeditor のAPI を利用することはできません。

EESession::open 以降、ENF Editor が持つすべてのAPI を利用できる状態になります。

EESession::read_enf メソッドにてENF ファイルをメモリに読み込みます。この後 ENF 編集の処理を行い、最後に *EESession::write_enf* メソッドにて ENF ファイルを書き出します。書き出されたENF ファイルは ENF Editor コンポーネントの出力として、ENF to CAD のような他のASFALIS コンポーネントに渡すことができます。

4.2.3. ENF 編集用処理の記述

前の節では、入力した ENF ファイルをそのまま出力するだけの `template.rb` を紹介しました。本節では、このテンプレートに様々な ENF 編集の処理を追記してみましょう。以下のサンプルは ENF ファイルに含まれるすべてのコンポーネント(アセンブリおよびパート)を取り出し、その名前をログに記述するものです。`execute_impl` メソッド内、コメントアウトされた行冒頭の `#` を消し、保存した上で `test.bat` を実行してみましょう。ENF ファイルに含まれるコンポーネントの名前が作業フォルダ内のログに出力されるはずです。

```
def execute_impl(session)
  # ここに ENF 編集スクリプトを記述してください
  session.model.components.each do |compo|
    unless compo.name.nil?
      @logger.info "Component name is .. #{compo.name}"
    end
  end
end
```

以下はモデルに含まれるすべてのコンポーネント名を取得する ENF Editor スクリプトの処理フローです。EESession クラスや EESession::model メソッドなど、ENF Editor API の詳細な説明は、ENF Editor API Reference Guide を参照してください。

1. EESession クラスから EEModelSession インスタンスを生成します。EEModelSession インスタンスを生成する際には、EESession::model メソッドを使います。(session.model の部分)
2. EEComponent インスタンスの配列を生成します。EEComponent インスタンスを配列で生成する際は、EEModelSession::components メソッドを使います。(session.model.components の部分)
3. EEComponent インスタンスの各要素を取得します。すべての component に対して同じ処理を行う際には、Ruby の標準メソッドである Array::each を使います。(session.model.components.each の部分)
4. 各 EEComponent インスタンスのコンポーネント名を取得します。component 名を取得する際には、EEComponent::name method を使います。(compo.name の部分)

4.3. その他の重要項目

この節では、スクリプトを書き始める前に必ず目を通して頂きたい、ログ出力やエラー処理などスクリプトを記述する上での重要項目を紹介します。

4.3.1. モジュールとクラス

ENF Editor のすべてのクラスは "**EnfEditor**" モジュールに属しています。ENF 要素を表すクラスは EEComponent や EEBrepFace のようにすべて "EE" から始まります。これらは EEBase クラスのサブクラスです。

4.3.2. 引数とパラメータ

スクリプトにはいくつか引数が与えられますが、これらの値は必ず `EESession::open` に渡す必要があります。

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

`EESession::parameter (Hash)` にて入出力ファイルパスを取得することができます。ハッシュのキーは `inputfile` と `outputfile` となります。

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

4.3.3. ログ出力

STDOUT(標準出力) に書き出されたメッセージはすべて ASFALIS のログファイルとして記録されます。ログ出力の際には "logger" ライブラリを用います。

```
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. 例外処理

何らかの例外が発生した場合、API は `EEError` として例外を出力します。例外発生時には、例外をログに記述することで何が起きたのか記録することができます。例外を無視して処理を続行させる際には、例外を `rescue` で捕捉する必要があります。捕捉しない場合、ENF Editor はエラーコードを出力し終了します。そこから ENF Editor を例外終了させる場合、`SystemExit` クラス経由で例外を発生させ任意のエラーコードを渡してください。SystemExit に渡されたエラーコードはログファイルに出力されます。

```
begin
  # read_enf メソッドは入力ファイルが存在しない場合 EErrorとして例外を出力する
  session.read_enf('no_exist_file')
rescue => ex
  # エラー情報をログに記録
  @logger.error ex
  # エラーコードとともに終了させるために例外を発生させる
  raise SystemExit.new(99)
end
```

4.3.5. エラーコード

SystemExit 例外で発生させたスクリプトのエラーコードと ENF Editor コンポーネントのエラーコードの関係は以下のようになります。

スクリプトのエラーコード	ENF Editor のエラーコード	内容
0	0	正常終了
6	6	入力ファイルが見つからない
上記以外	975	ENF Editor のエラー終了

4.3.6. デバッグ

スクリプトがエラーコードを出力して終了した場合、まずはログファイルを確認してください。エラーメッセージやコールスタックがログファイル内に記録されていますので、問題確認の一助となりますはずです。

4.3.7. エンコーディング

ENF Editor API が String 値を必要とした場合、すべて UTF-8 で渡す必要があります。スクリプトについても UTF-8 を推奨しています。UTF-8 にする場合、先頭行に下記のマジックコメントでスクリプトのエンコーディングを指定してください。

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

UTF-8 以外でエンコーディングされたファイルを読み込む場合、文字列を UTF-8 に変換する必要があります。デフォルトの内部エンコーディングおよび外部エンコーディングは UTF-8 です。

```
# Shift-JIS のファイルを UTF-8に変換して読み込む
File.open(sjis_file, 'r:windows-31j:utf-8') do |f|
  # ...
end
```

4.4. 発展的な項目

本節の項目はより進んだ開発者向けの情報です。

4.4.1. ENF オブジェクトの有効性確認

オブジェクトが有効であるかどうかを確認する必要がある場合、`null?`メソッドを使用します。たとえば、いったんユーザ属性をクラスごと削除すると `UserProperty` クラスにアクセスできなくなりますが、この状況で `user_prop.null?` は `true` を返します。

```
component.user_properties.each do |user_prop|
  user_prop.null?      # => false
  user_prop.delete!    # ユーザ属性を削除
  user_prop.null?      # => true
end
```

4.4.2. ENF オブジェクトの等価性確認

2つの ENF オブジェクトが同じであることを確認する際には、`eql?` もしくは `==` を用います。下記の例を確認してください。`edge_org` と `edge_from_curve` は同じ `EEBrepEdge` オブジェクトです。ここで `eql?` もしくは `==` を使うと、内部の ENF オブジェクト ID を確認し、その ID が同じ場合 `true` を返します。Ruby には類似のメソッドとして `equal?` もありますが、これはオブジェクトそのものが同じかどうか調べるメソッドですので下記の例では `false` を返します。

```
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. 進捗度のアップデート

ASFALIS デスクトップや ASFALIS TransServer 等で ENF Editor コンポーネントの進捗度を表示したい場合、`EESession::process_meter_main=` を利用してください。0~100 までの値が有効です。

```
session.process_meter_main = 0      # 0 パーセント
session.read_enf(input_enf)
session.process_meter_main = 30     # 30 パーセント
# ここにENF編集スクリプトを記述してください
session.write_enf(output_enf)
session.process_meter_main = 100    # 100 パーセント
```

5. ENF 構造

本章では ENF ファイルの構造を説明します。

5.1. コンポーネント

コンポーネント(Component)とはアセンブリ要素およびパート要素のことです。パートコンポーネントはアセンブリツリーの終端に位置し、Brep またはポリゴンのどちらか一方もしくはその両方にて表現されます。アセンブリコンポーネントはパートもしくはアセンブリの親となる要素で、複数のコンポーネントをまとめる要素です。

以下の図をご参照ください。このモデルは二つのアセンブリコンポーネントと三つのパートコンポーネントで構成されています。"wheel" というパートは四つのインスタンスを持ちますが、これは共有コンポーネントとなるため "wheel" コンポーネント自体は一つと見なします。

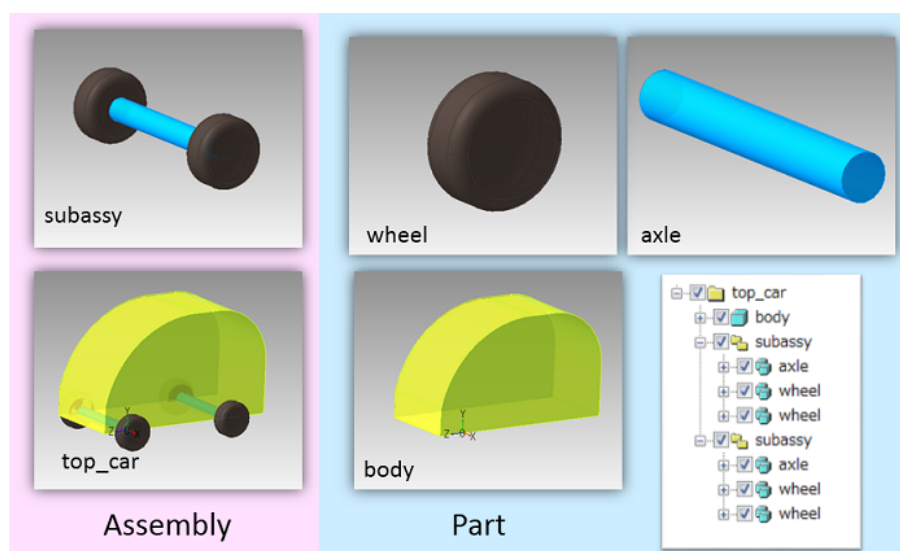


図 3. アセンブリコンポーネント、パートコンポーネントの例

5.2. Brep 要素

Brep はパートコンポーネントの子要素です。Brep は以下の図で示すように位相要素と形状要素で構成されます。

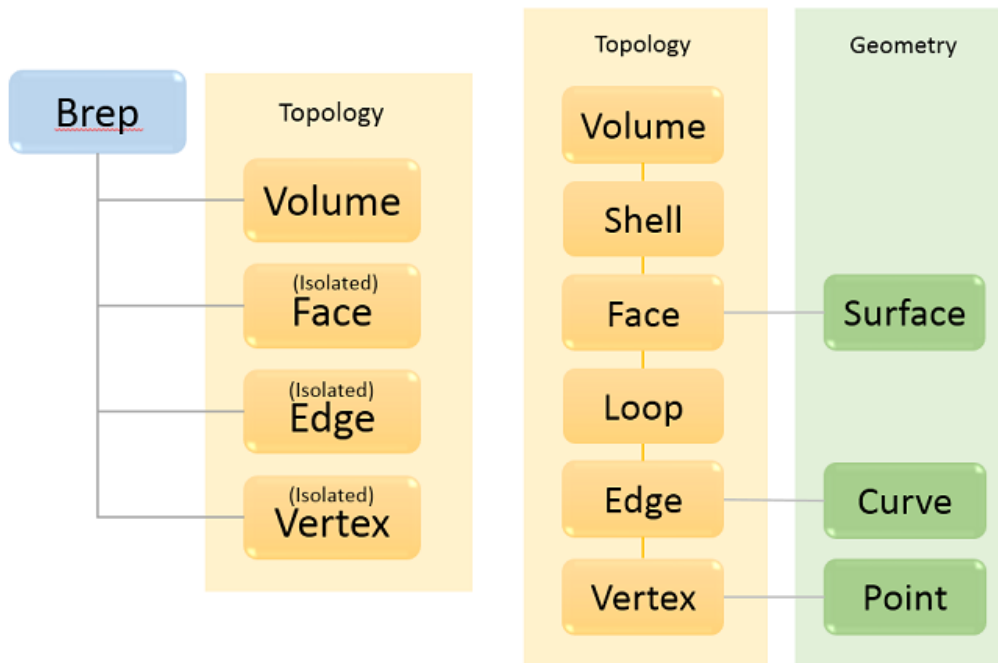


図 4. Brep のツリー構造

1. ボリューム

ボリューム (Volume) は複数のシェルで構成されます。閉じたボリュームをソリッド、開いたボリュームをシートと呼びます。

- クラス: EnfEditor::EEBrepVolume

2. シェル

シェル (Shell) は複数のフェースで構成されます。

- クラス: EnfEditor::EEBrepShell

3. フェース

フェース (Face) は複数のループと形状要素である曲面から構成されます。フェースはボリュームに属していることもありますが、ボリュームに属さない単独フェースの可能性もあります。

- クラス: EnfEditor::EBrepFace

4. ループ

ループ (Loop) は複数のエッジから構成されます。

- クラス: EnfEditor::EEBrepLoop

5. エッジ

エッジ (Edge) は曲線に対応する位相要素です。エッジはボリュームに属していることもありますが、ボリュームに属さない単独エッジの可能性もあります。

- クラス: EnfEditor::EEBrepEdge

6. 頂点

頂点 (Vertex) は点に対応する位相要素です。頂点はボリュームに属していることもありますが、ボリュームに属さない可能性もあります。

- クラス: EnfEditor::EEBrepVertex

7. 曲面

曲面 (Surface) はフェースに対応する形状要素です。

- クラス: EnfEditor::EEBrepSurface

8. 曲線

曲線 (Curve) はエッジに対応する形状要素です。

- クラス: EnfEditor::EEBrepCurve

9. 点

点 (Point) は頂点に対応する形状要素です。

- クラス: EnfEditor::EEBrepPoint

5.3. Brep 要素に対応する表示属性

5.3.1. レイヤ

各要素にはレイヤ番号として整数値の Layer 属性が付与されています。値の範囲は 0~2147483647 です。レイヤをサポートしている要素は以下の通りです。

- ボリューム
- フェース
- 曲面
- エッジ
- 曲線
- 頂点
- 点



エッジおよび頂点のレイヤは対応する曲線および点を相互参照します。たとえば、エッジのレイヤを変更した場合、対応する曲線のレイヤも変更されます。



ENF2 フォーマットの場合、値の範囲は 0~255 となります。ENF2 フォーマットを読み書きする際、レイヤ番号は強制的に 0~255 の範囲に補正されます。

5.3.2. 色・透明度

MaterialColor 属性は色や透明度 (アルファ値) を表現するためのものです。各要素の色や透明度は以下のリストのうち一つの形式で表現されます。

- RGB 型- RGB のみ。透明度は定義されません。
- Alpha 型- 透明度のみ。RGB は定義されません。
- RGBA 型- RGB と透明度の両方を定義します。
- Material Strength 型- RGBA に加えスカラー値のマテリアル属性を定義します。
- Material RGBA 型- RGBA 形式でマテリアル属性を定義します。

r,g,b,a 各値の有効範囲は 0.0~1.0 で、精度は double です。たとえば、(r,g,b) = (1.0, 0.0, 0.0) とした場合、‘赤色’を表現できます。alpha は透明度を示す値で、0.0 に近づくほど透明、1.0 に近づくほど不透明になります。サポートしている要素は以下の通りです。

- ボリューム
- フェース
- 曲面
- エッジ
- 曲線
- 頂点
- 点



エッジおよび頂点の色・透明度は対応する曲線および点の色・透明度を相互参照します。たとえば、エッジに色を設定した場合、対応する曲線の色も変更されます。



ENF2 フォーマットの場合、r,g,b 値の精度は 1/100 となります。この場合、r,g,b 値は N/100 の値まで丸められます。また、ENF2 フォーマットにおいて Material Strength 型および Material RGBA 型はサポートしていません。

5.3.3. 表示・非表示

Display 属性は boolean 型で表現されます。これは各要素の表示非表示を表現するためのものです。サポートしている要素は以下の通りです。

- ボリューム
- フェース
- 曲面
- エッジ
- 曲線
- 頂点
- 点



エッジおよび頂点の表示・非表示は対応する曲線および点の表示・非表示を相互参照します。たとえば、あるエッジの Display 属性を変更した場合、対応する曲線の Display 属性も変更されます。

5.3.4. 名前

Name 属性は各要素の名前を表すためのもので、String 値で表現されます。サポートしている要素は以下の通りです。

- ボリューム



ENF2 フォーマットでは、ボリューム名の最大長は 80 となります。

5.4. システム属性

System property は各コンポーネントのシステム属性を表し、予約語となります。システム属性は以下の通りです。

- BinName
- ChangeHistory
- ConfigName
- Description
- PartComment
- PartDefinition
- PartName
- PartNumber
- PartVersion
- PartRevision
- PartSource
- PartLayer
- Name
- NFName - 変換元ファイル名
- TFName - 変換先ファイル名
- Material



サポートしているシステム属性は CAD ごとに異なります。たとえば、BinName と ConfigName は IDEAS のみサポートしています。各CAD がどのシステム属性に対応しているかは ASFALIS_manual.pdf 内の「属性変換」の節をご確認ください。

5.5. ユーザ属性

User Property は各要素に付与できるユーザ属性です。以下の要素で構成されます。

- Key - ユーザ属性キー (string 値)
- Value - ユーザ属性値 (string 値)
- Type - ユーザ属性値のデータ型
- Subtype - サブタイプ (string 値・任意)

Type として指定できるユーザ属性値のデータ型は以下の通りです。

- BoolChar - 1 文字の値 (例: "c")
- Bool - 真偽を表すbool 値 ("1" もしくは "0")
- Integer - 整数値 (例: "123")
- Real - 倍精度浮動小数点値 (例: "4.56")
- Section - 内部処理用の型ですので使用しないでください
- Text - 文字列

サポートしている要素は以下の通りです。

- コンポーネント
- ボリューム
- フェース
- エッジ
- 頂点

5.6. ENF バージョンの互換性

ENF フォーマットの最新バージョンは Ver3.3 で、出力時にはデフォルトでこのバージョンの ENF を書き出します。また、ENF Editor ではすべてのバージョンの ENF ファイルを読み込むことができます。ただし、最新バージョンの ENF を使用することを推奨します。旧バージョンの ENF を用いた場合、サポートされていない属性や要素がスキップされる可能性があります。

6. CAD および変換仕様

ENF Editor API を活用するためには、CAD 自体の仕様および CAD-ENF 間の変換仕様を理解することが不可欠です。

各CAD の仕様詳細に関しては別冊の `cad_spec/**_spec.pdf` をご参照ください。

本コンテンツに関わる著作権は株式会社エリジオンもしくは原権利者に帰属しています。
著作権者の承諾なしに無断で改変、複製、転載、再配布、転送、公衆送信、販売、貸与などの
行為をすることは禁じられています。