

PDF3D ReportGen 自動化（複数データの複数ビューへの埋め込み）

ReportGen による複数データの複数ビューへの埋め込みを自動化する方法について説明します。最初に、「PDF3D ReportGen のバッチ処理と自動化」に関するドキュメント（別資料）を参照し、まずは基本的なバッチ処理の方法について確認してください。

なお、以下の説明では、自動化のプログラムに Python3 を利用しています。Python についての詳細は、Web 上の情報などを参照してください。以下のサンプルを実際に実行してみるには、Python の環境が必要です。

<補足>

複数のデータを複数のビューに埋め込むための補助ツールも公開しています。サポート情報にある、以下の項目を参照してください。

「複数のデータを複数のビューに並べるには（ツールの利用）」

「複数のデータを複数のビューに並べるには（ツールの利用）（360 度パノラマ画像用）」

<注意>

このドキュメントでは、3D データの変換を対象に説明しています。パノラマ画像の変換では、ステート・ファイルの作成方法が異なります。別資料「PDF3D ReportGen のバッチ処理と自動化」では、パノラマ画像の場合のステート・ファイルの作り方について説明していますので、そちらを参照してください。ステート・ファイルが作成できれば、それ以降の手順は基本的に同じです。

複数データの複数ビューへの埋め込みについて

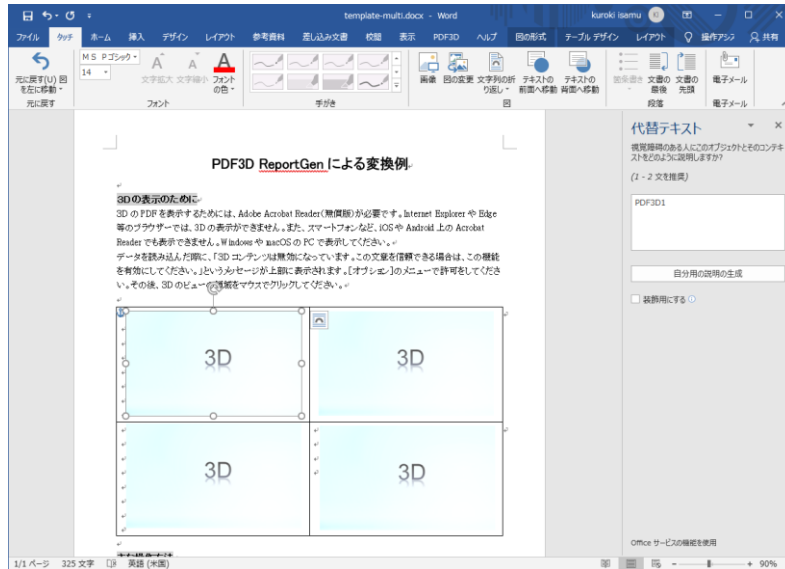
ReportGen で、複数のデータを複数のビューに埋め込むには、データを選択とそのデータを埋め込むアノテーション番号の指定、出力する PDF ファイルの指定を複数回繰り返す必要があります。（詳細はチュートリアル・ガイドを参照してください。）

ここでは、この手順を自動化する方法について説明します。

まず、複数のデータを埋め込みたいテンプレート PDF ファイルを Word や PPT で作成し、その埋め込む位置に仮の画像を貼り付け、PDF3D1 や PDF3D2 といった代替テキストを埋め込んでください。（このテンプレートの作り方についてもチュートリアル・ガイドを参照してください。）

例えば、以下に Word 文書のサンプルを示します。

4 つのビューを配置する画像を置き、それぞれ、PDF3D1～PDF3D4 の代替テキストを設定しています。



まずは、この文書をテンプレート PDF ファイル（例えば template-multi.pdf）として保存しておきます。

データの準備とステート・ファイルの作成

4 つのビューに埋め込むデータ・ファイル、例えば、以下の 4 つのファイルを準備します。

- data/00.wrl.gz
- data/10.wrl.gz
- data/20.wrl.gz
- data/30.wrl.gz

連番である必要はありません。4 種類のファイルを準備します。

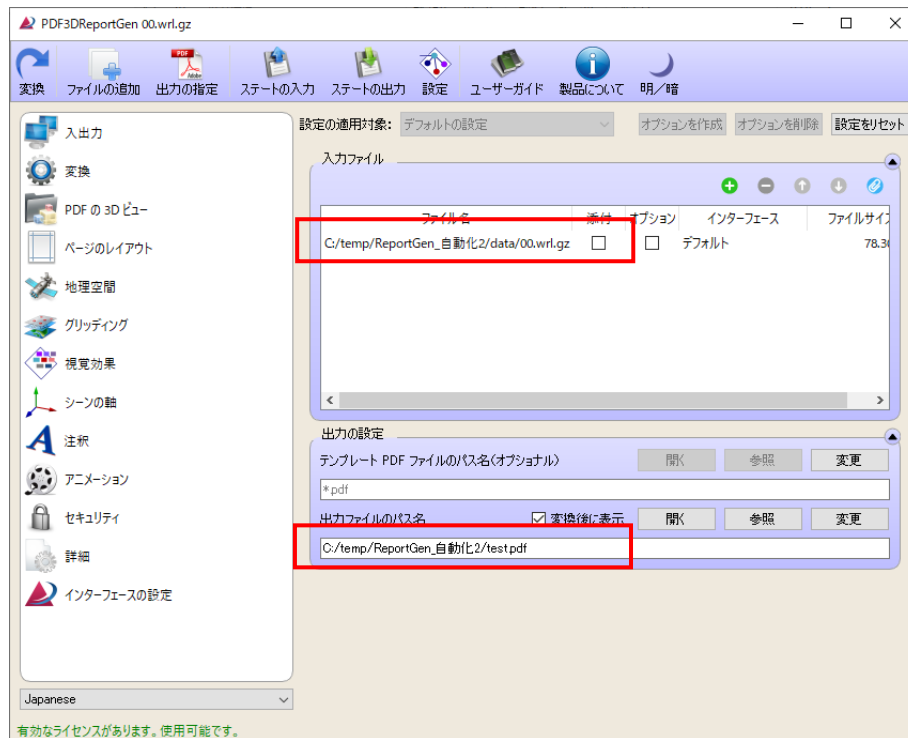
準備できたら、まず、自動化のためのステート・ファイルを作成します。

今回は、テンプレート PDF ファイルを、[テンプレート PDF ファイルのパス名(オプション)] に設定する方法ではなく、直接、[出力ファイル名のパス名] に指定する方法を使います。そのため、まず、上書きしてなくならないように、作成したテンプレート PDF ファイル（template-multi.pdf）をコピーし、test.pdf の名前で保存します。

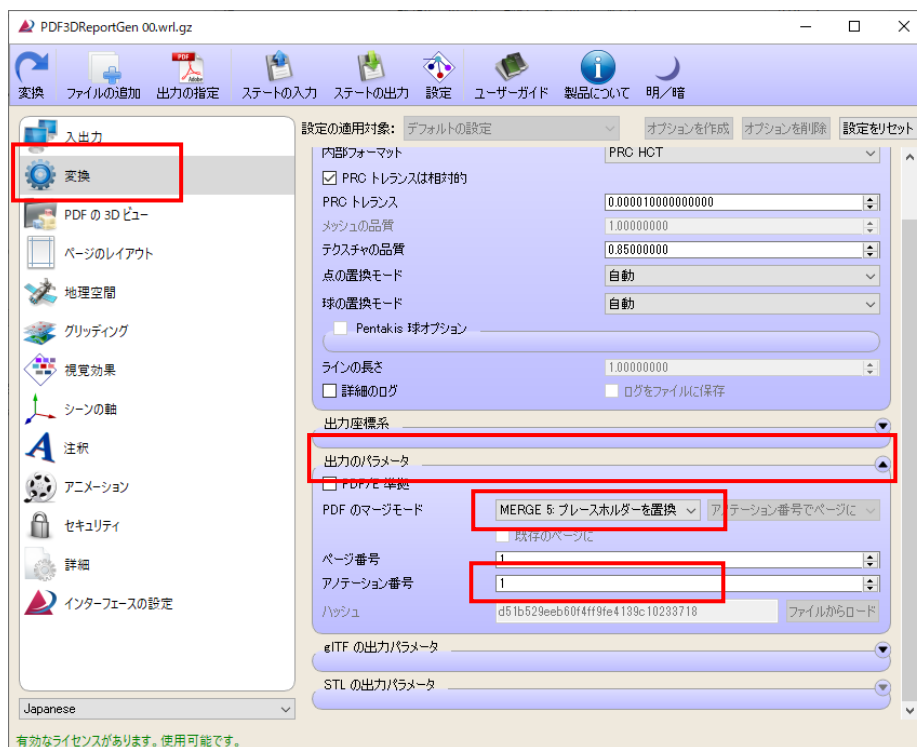
次に ReportGen を起動します。

入力ファイルに 00.wrl.gz を設定します。

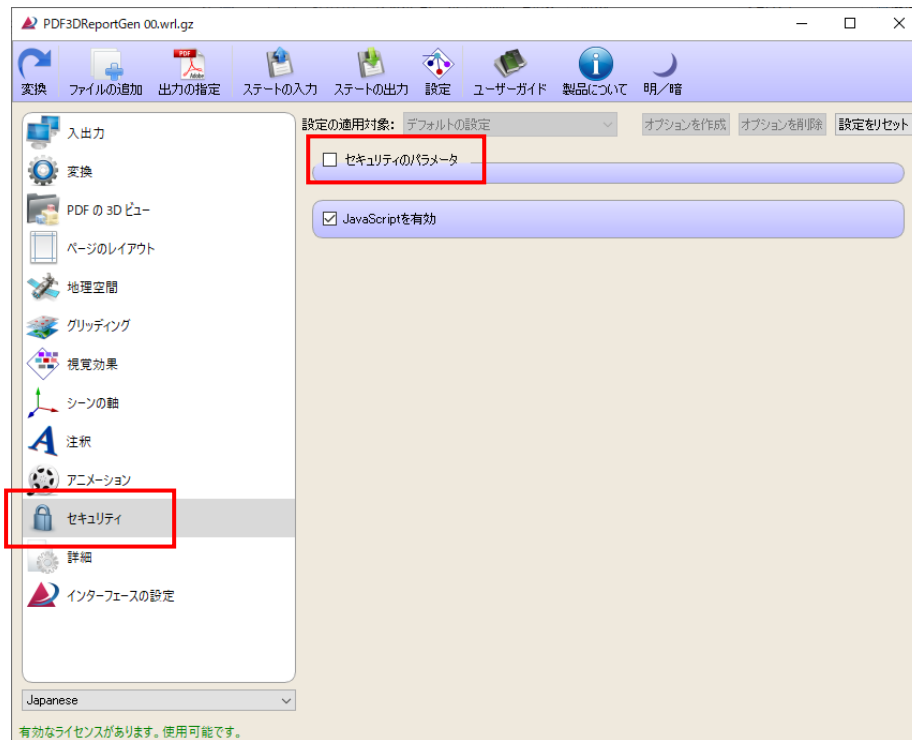
次に、出力ファイルに、今回は直接テンプレート PDF ファイル、test.pdf を指定します。上書きメッセージが表示されますので、[はい] を選びます。



次に変換タブを選択し、[出力のパラメータ]を開きます（右側の▼アイコンをクリック）。
変換のマージモードを [MERGE:5 プレースホルダーを置換] に設定し、アノテーション番号を 1 番（デフォルト）に設定します。

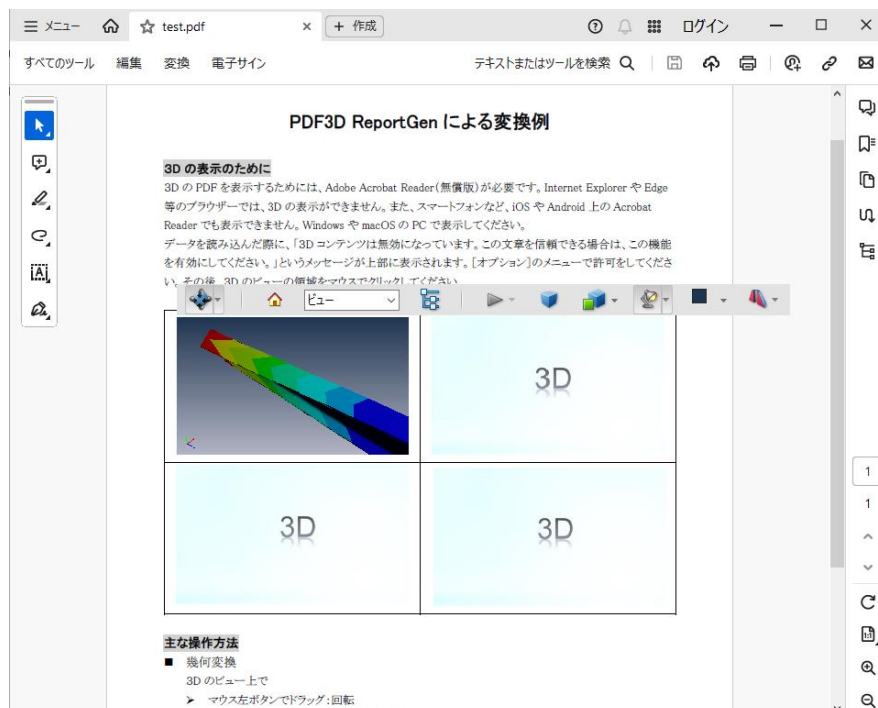


次に [セキュリティ] タブに移動し、[セキュリティのパラメータ] のチェックをオフにします。



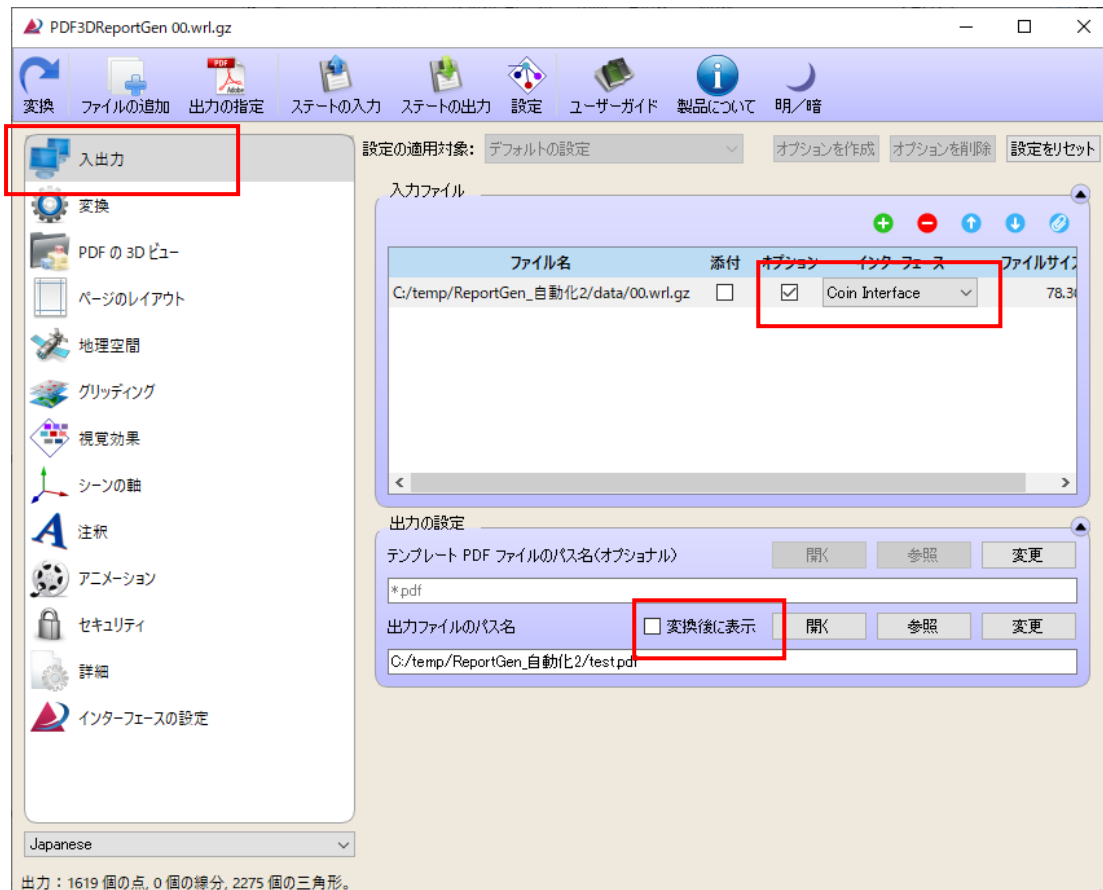
その他、背景色など、適宜、必要なパラメータを設定し、変換を実行してみてください。

1 つ目の画像の位置に 1 つ目のデータが埋め込まれたことを確認してください。



正しく変換できたら、自動化のためのステート・ファイルに保存します。

まず、[入出力] タブを開き、[入力ファイル] のリストにある [オプション] にチェックします。このデータ (VRML) の変換インターフェースは 1 つのみで、[Coin Interface] が選択された状態となります。

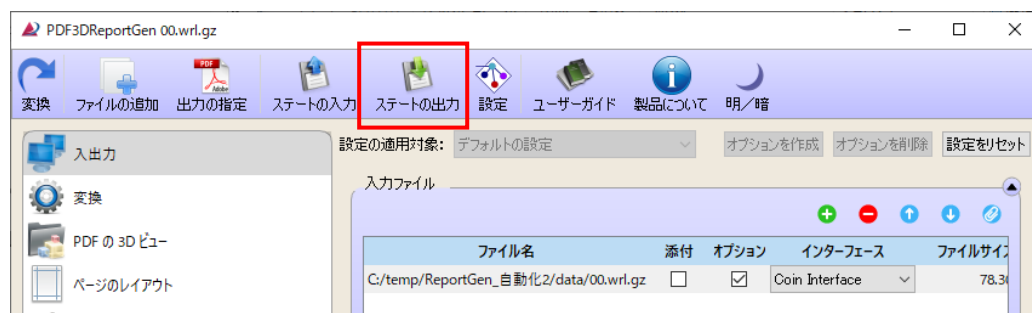


<注意>

バッチ処理を行うには、そのデータに対応したインターフェイスが 1 つの場合でも、必ず、入力ファイル名の横の「オプション」がオンに設定され、そのデータを変換するためのインターフェイスが指定されている必要があります。

次に、[出力ファイルのパス名]にある[変換後に表示]のチェックをオフにします。変換終了時に自動的に PDF ファイルを開くかどうかの設定で、自動化の方法によっては、途中で PDF ファイルが開くと上書きできないなど、処理が正しく行われないことがありますので、オフにします。

最後に、[ステートの出力]アイコンをクリックし、ステート・ファイルに保存します。拡張子が.pdf3dsettings の XML 形式のアスキー・ファイルができます。(ここでは、multi.pdf3dsettings ファイルとします。)





メモ帳で開き、入力ファイル、出力ファイル、アノテーション番号をキーワード化します。Python プログラムでは、これらのキーワードをプログラムで置換しながら、ReportGen をバッチで実行します。

以下のタグの値をキーワード化します。

```
<OutputFileName value="MY_OUTPUT_PDF"/>    ← 出力ファイル

<MergeMode value="Placeholder">
  <ReplaceMode value="Number"/>
  <AnnotationNumber value="MY_ANNOT_NO"/>    ← アノテーション番号
  <AnnotationHash value="d51b529eeb60f4ff9fe4139c10233718"/>
</MergeMode>

<Assembly>
  <InputFileName value="MY_INPUT_DATA"/>    ← 入力ファイル
  ...
```

なお、ステート・ファイルの保存の際に述べた、PDF ファイルを変換終了後に自動で開くかどうかは、以下のタグで設定されています。（false で開かない設定になっています。）

```
<ShowPDFAfterClose value="false"/>
```

Python を利用した自動化サンプル

ステート・ファイルと Python を利用した自動化のサンプルを紹介します。

PythonMulti.py : Python プログラム

<補足>

Python の詳細については、Web 上の Python に関する情報などを参照してください。

1) バッチ実行のための変数定義

まず、このバッチファイルでは、プログラム名（ReportGen の実行バイナリ）とステート・ファイルの変数を定義しています。

```
program_filename = r'C:\Program Files\PDF3DReportGen\PDF3DReportGen.exe'  
org_state_filename = r'multi.pdf3dsettings'
```

また、ループの回数（4つのビューを作成するため、4回実行）やテンプレート PDF ファイル名、出力ファイル名、入力データの配列（4つのビューに埋め込むための4つのデータ）を定義しています。

```
loop_num = 4  
  
template_pdfname = r'template-multi.pdf'  
output_pdfname = r'multi-3D.pdf'  
  
input_data = [  
    r'./data/00.wrl.gz',  
    r'./data/10.wrl.gz',  
    r'./data/20.wrl.gz',  
    r'./data/30.wrl.gz'  
]
```

2) ファイルのコピー

このプログラムの main 関数では、run_batch 関数を呼び出しています。この関数では、最初にテンプレートの PDF ファイルを出力用の PDF ファイルにコピーしてから実行しています。

```
def run_batch():  
  
    # copy template to target 3D pdf  
    shutil.copy(template_pdfname, output_pdfname)
```

次にループ（4回の処理）を作成しています。

```
for i in range(loop_num):
```

ループの中では、まず、ステート・ファイルをテンポラリの名前でコピーします。

```
# set file
state_filename = org_state_filename + ".tmp"
shutil.copy(org_state_filename, state_filename)
```

ステート・ファイルの中の変数化したキーワードの置換を行っています。まず、MY_OUTPUT_PDF を置換しています。

```
# change keyword (Output filename)
old_name = "MY_OUTPUT_PDF"
out_name = output_pdfname
change_target(state_filename, old_name, out_name)
```

出力ファイル名は、常に同じファイルを指定しています。先の手動による 1 つ目のデータの変換と同じように、テンプレート PDF ファイルを出力ファイル名に指定し、アノテーション番号のビューを置換します。

change_target 関数の中では、ファイルを開き、replace 関数で置換処理を行っています。

以降、MY_INPUT_DATA、MY_ANNOT_NO のキーワードの置換を行っています。MY_INPUT_DATA は、最初に定義した入力ファイルの配列、input_data [i] に置換しています。また、MY_ANNOT_NO の置換では、ループの回数 str(i+1) を変換後の文字列としています。

3) ReportGen の実行コマンドの作成

文字列の置換が終わったステート・ファイルを利用して、以下のように ReportGen をバッチ処理します。

```
'C:¥Program Files¥PDF3DReportGen¥PDF3DReportGen.exe'
    -state multi.pdf3dsettings.tmp （置換後のステート・ファイル）
    -silent （紙面の都合上、改行）
```

command_string を作成し、subprocess モジュールを使って、実行しています。

```
# run command
command_string = [program_filename, "-state", state_filename, "-silent"]

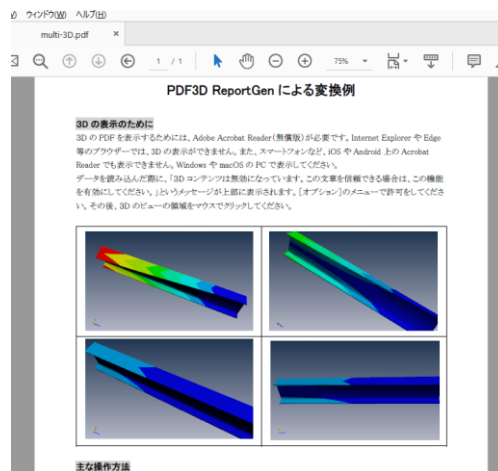
# execute
try:
    print('running [%d] : %s ' % (i, command_string))
    success = subprocess.call(command_string)
```


4) 実行

Python プログラムを実行します。

```
python PythonMulti.py
```

正常に処理が終わると、4 つのデータがそれぞれ 4 つのビューに埋め込まれます (multi-3D.pdf)。



注) 本資料の説明用フォルダに含まれているデータを PDF3D ReportGen の動作確認以外の目的で利用することを禁じます。