DICOM 画像変換ツール
(DICOM Image Converter)

(株)アイプランツ・システムズ

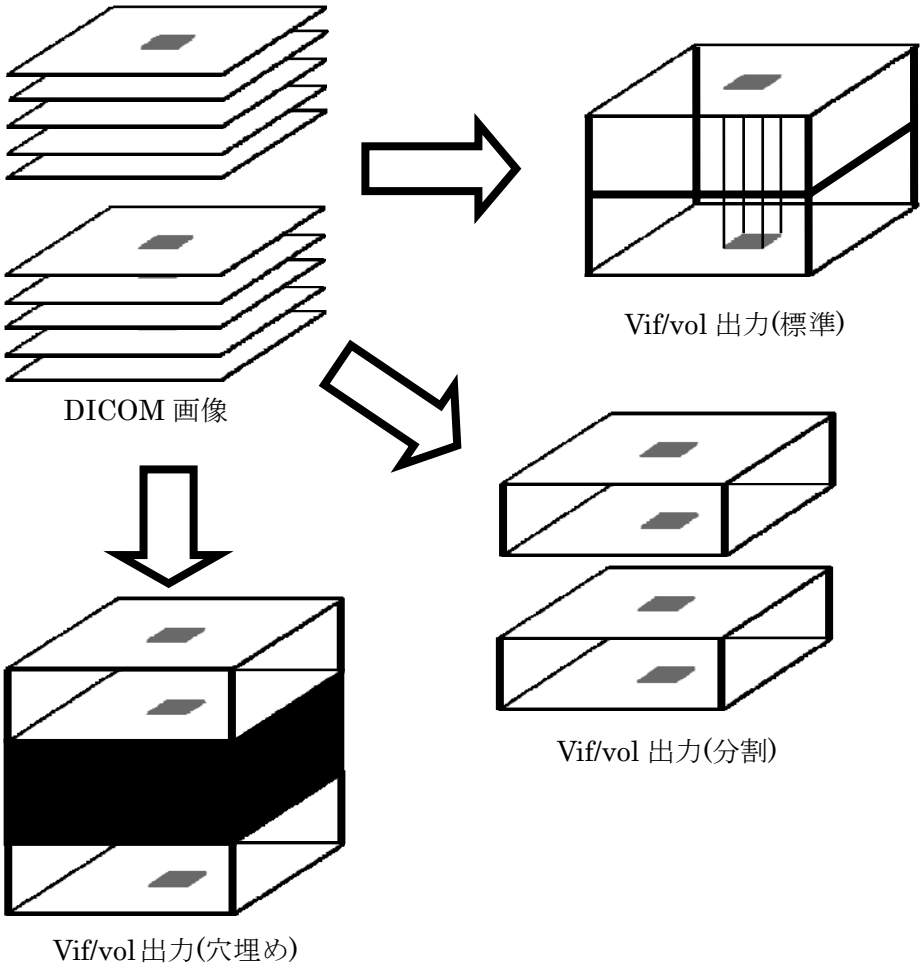
Dicom Image Converter

本ツールの目的

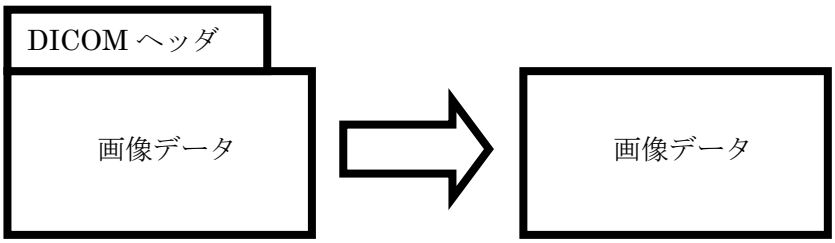
○複数の DICOM スライス画像ファイルから DICOM ヘッダを取り除き、画像データのみの Raw ファイルを抽出します。

○複数の DICOM スライス画像ファイルを読み込み、一つの VOL（付録参照）ファイルへ変換し、対応する VIF ファイルを作成します。

ケース 1）vif/vol ファイル出力

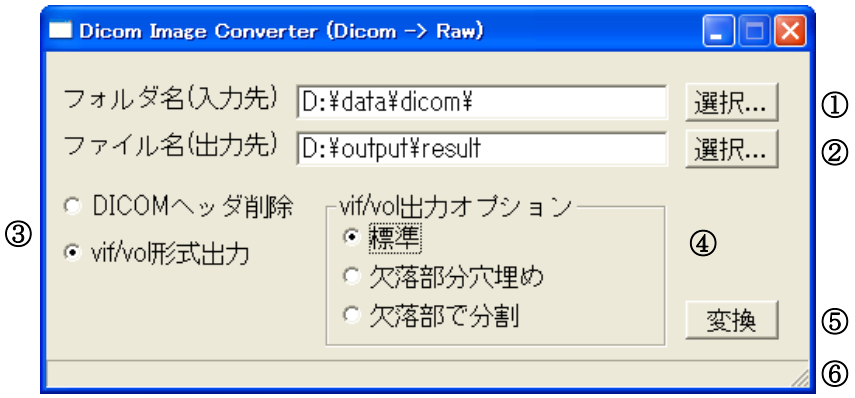


ケース 2）raw ファイル出力



DICOM 画像(一枚)　　Raw データ(一枚)

使用方法



		機能
①	DICOM フォルダ選択	選択ボタンをクリックし、DICOM ファイルが保存されているフォルダを指定します。
②	出力パス選択	選択ボタンをクリックし、変換後のファイルパスとファイル名を入力します。ファイル名に拡張子を付加する必要はありません。
③	出力形式選択	DICOM ヘッダ削除：DICOM ファイルからヘッダを取り除いた、画像データ部のみを抽出します。出力形式は.raw ファイルとなります。 vif/vol 形式出力：複数の DICOM ファイルを一つのファイルにまとめ、Volume Extractor 用フォーマットで出力します。
④	vif / vol 出力オプション	③において vif/vol 形式出力を選択したときのみ選択可能です。 標準：フォルダ内の DICOM 画像を単純に結合します。 欠落部分穴埋め：DICOM ファイルのスライス間隔を調べ、スライス間隔の大きい箇所を全 DICOM ファイルの最小値で埋めた vol ファイルを作成します。 欠落部で分割：DICOM ファイルのスライス間隔を調べ、スライス間隔の等しい箇所を一つのファイルとしてまとめ、複数の vol ファイルを作成します。
⑤	変換ボタン	変換を実行します。
⑥	ステータスバー	作業の進行状態などが表示されます。

出力ファイル名規則

出力されるファイル名には以下の規則が存在します。

・.raw ファイル出力の場合

元の DICOM ファイルと同数の.raw ファイルが作成されます。この時、ファイル名は

ファイル名 ＋4 桁の番号 ＋.raw

の形式をとります。この例では、result0001.raw ～ result0100.raw ～ が出力されます。

・vif / vol 出力(分割)の場合

DICOM 画像のスライス間隔が大きい箇所が 1 カ所の場合、vif/vol ファイルは各 2 つずつ作成されます。この時、ファイル名は

ファイル名 ＋_番号 ＋.vif

の形式をとります。この例では、result_1.vif , result_2.vif が出力されます。

・vif / vol 出力(標準)、及び穴埋めモードでは、ファイル名は以下の規則で決定します。

ファイル名 ＋.vif

付録

Volume　Extractor で使用している３次元画像のファイルフォーマットについて、説明します。

１．ファイルフォーマット

1.1. VDF フォーマット

Volume Extractor 用のファイルフォーマットです。

ファイル情報　［256byte］
画像データ　（Raw 形式） （縦 x 横 x 高 x 単位データサイズ）

ファイル情報

先頭の 256byte にはファイル情報として下記の情報が ASCII で格納されます。
各項目は、スペース（0x20）で区切られ格納されます。

	項目	内容
ファイル情報 ［256byte］	ファイルヘッダ	“VDF_1.0_VE12.8”
	StartPoint（描画開始位置）タグ	“sp”
	開始位置 X 座標	数値（実数）
	開始位置 Y 座標	数値（実数）
	開始位置 Z 座標	数値（実数）
	グリッドサイズタグ	“n”
	X 軸サイズ	数値（整数）
	Y 軸サイズ	数値（整数）
	Z 軸サイズ	数値（整数）
	ピッチサイズタグ	“pitch”
	X 方向 Pitch	数値（実数）
	Y 方向 Pitch	数値（実数）
	Z 方向 Pitch	数値（実数）
	データタイプタグ	“dt”
	データタイプ(1～4)	数値（1～4）
	情報終了識別子	¥n(0x0A)
	（予備）	0x00

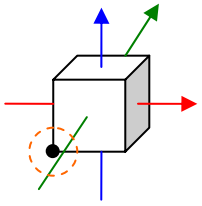
・　ファイルヘッダ

文字列“VDF_1.0_VE12.8”固定

・　StartPoint

3D 空間上での配置位置の指定

右図のようにモデルの 3D 空間上での開始点を指定します。



・　グリッドサイズ

X 軸、Y 軸、Z 軸、各方向のピクセル（ボクセル）数のサイズ

・　データタイプ

１ピクセル（ボクセル）のデータで使用されるデータの型

1 ： Byte 型（1byte）

2 ： Unsigned Short 型（2byte）

3 ： Short 型（2byte）

4 ： int 型（4byte）

・　情報終了識別子

ファイル情報の最後に“¥n”(0x0A)を格納

・　（予備）

この部分は、今後の拡張等のための箇所として用意されています

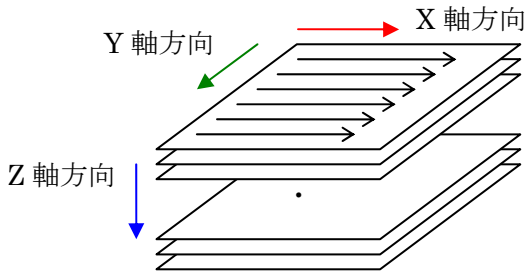
0x00 で埋めます。

画像データ

画像データが RAW 形式で格納されます。
各データ要素は「データタイプ」で指定されたサイズになり、データ全体としては下記のサイズになります。

（グリッド X 軸サイズ）×（グリッド Y 軸サイズ）×（グリッド Z 軸サイズ）×（データタイプの指定 byte 数）

格納順序は、X 軸方向の要素から格納開始され、Y 軸方向、Z 軸方向へと格納されています。
（右図イメージ参照）



※VE での表示は、Z 軸方向を上向きにしていますのでご注意下さい。

１．２．VOL フォーマット

拡張子、“vol” と “vif” の 2 つのファイルにより構成されたボリュームデータファイルです。vif がファイル情報部分、vol が画像データ部分となります。

vif

vif ファイルにはファイル情報として下記の内容が ASCII として 5 行構成で格納されます。（改行コードは ¥r¥n（0x0D0A））

行数	項目	記述内容
1	ファイルヘッダ文字列	“VIF 1.0 VE12.8”
2	StartPoint	“start_pt [X 座標] [Y 座標] [Z 座標]”
3	画像（グリッド）サイズ	“size [X 方向] [Y 方向] [Z 方向]”
4	各方向のピッチ	“pitch [X 方向] [Y 方向] [Z 方向]”
5	データタイプ	“data_type [1～4]”

2 行目以降の各項目は、内容を示す文字列（”start_pt”等）が記述され、続けて値（数値）が格納されます。

文字列と値の間は半角スペース（0x20）2 文字分で区切り、各値（数値）の間は半角スペース 1 文字分で区切ります。

【例】

```
VIF 1.0 VE12.8
start_pt  -0.5 -0.5 -0.5
size  512 512 469
pitch  0.1693333 0.1693333 0.64
data_type  2
```

vdf

画像データ要素が RAW 形式で格納されます。

格納されるデータのサイズ、順序等は VDF のデータ部分と同じです。（VDF フォーマット参照）

２．出力サンプルコード

VDF、VOL、VIF の出力サンプルコードを下記に記します。

※　エラー処理等は考慮されていませんので、ご注意下さい

```
    ///
    /// Raw データの書き出し
    /// 下記のメソッド内で使用します。
    ///
    private: bool SaveRAW(BinaryWriter^ bw, array<unsigned char>^ data){
        // Raw データ書き出し
        int count = 0;
        while(count < data->Length){
            bw->Write(data[count]);
            count++;
        }
        return true;
    }


    ///
    /// VOL ファイルの書き出し
    ///
    private: bool SaveVOL(
      String^ path,                         // 出力先のファイルパス
      array<unsigned char>^ data            // 出力元データ（1 次元配列のボリュームデータ）
    ){
        // VOL 書き出し
        FileStream^ fs = gcnew FileStream(
                         path, System::IO::FileMode::Create,
                         System::IO::FileAccess::Write, System::IO::FileShare::None);
        BinaryWriter^ bw = gcnew BinaryWriter(fs);

        // VOL ファイルは、実際のところ Raw データそのもの
        SaveRAW(bw, data);

        bw->Close();
        fs->Close();

        return true;
    }
```

```
///
/// VIF ファイルの書き出し
///
private: bool SaveVIF(
  String^ path,                          // 出力先のファイルパス
  int x, int y, int z,                   // データグリッドサイズ
  double spx, double spy, double spz, // データ開始位置
  double ptx, double pty, double ptz  // データピッチ
){
    // VIF 書き出し
    FileStream^ fs = gcnew FileStream(
                        path, System::IO::FileMode::Create,
                        System::IO::FileAccess::Write,
                    System::IO::FileShare::None);
    StreamWriter^ sw = gcnew StreamWriter(fs);
    String^ tmpstr = L"";
    sw->NewLine = L"¥r¥n";    // 改行コード
    //全部で 5 行
    // 1 行目
    sw->WriteLine(L"VIF 1.0 VE12.8");
    // 2 行目
    tmpstr = L"start_pt  "
           + spx.ToString() + L" " + spy.ToString() + L" " + spz.ToString();
    sw->WriteLine(tmpstr);
    // 3 行目
    tmpstr = L"size  "
           + x.ToString() + L" " + y.ToString() + L" " + z.ToString();
    sw->WriteLine(tmpstr);
    // 4 行目
    tmpstr = L"pitch  "
           + ptx.ToString() + L" " + pty.ToString() + L" " + ptz.ToString();
    sw->WriteLine(tmpstr);
    // 5 行目
    tmpstr = L"data_type  " + L"1";   // サンプルでは unsigned char 型に固定
    sw->WriteLine(tmpstr);

    sw->Close();
    fs->Close();

    return true;
}
```

```
    ///
    /// VDF ファイルの書き出し
    ///
    private: bool SaveVDF(
      String^ path,                          // 出力先のファイルパス
      int x, int y, int z,                   // データグリッドサイズ
      double spx, double spy, double spz,    // データ開始位置
      double ptx, double pty,double ptz,     // データピッチ
      array<unsigned char>^ data             // 出力元データ（1 次元配列のボリュームデータ）
    ){
        // VDF 書き出し
        FileStream^ fs = gcnew FileStream(
                    path, System::IO::FileMode::Create,
                    System::IO::FileAccess::Write, System::IO::FileShare::None);
        BinaryWriter^ bw = gcnew BinaryWriter(fs);

        // ヘッダ文字列     unsigned char 型の例（データタイプ=1。ヘッダ長さは 256 固定）
        String^ str_header = L"VDF_1.0_VE12.8"
                + " sp "    + spx.ToString() + " " + spy.ToString() + " " + spz.ToString()
                + " n "     + x.ToString()   + " " + y.ToString()   + " " + z.ToString()
                + " pitch " + ptx.ToString() + " " + pty.ToString() + " " + ptz.ToString()
                + " dt "    + "1"
                + "¥n";

        // String を Char クラス配列に変換
        array<Char>^ transfer_header = str_header->ToCharArray();
        // 実際に書き出す Char クラス配列
        array<Char>^ put_header_array = gcnew array<Char>(256);
        // 書き出しに使用する配列に、データを転送
        Array::Copy(transfer_header, put_header_array, transfer_header->Length);

        // 未使用列は、0 で埋める
        int count = transfer_header->Length;
        while(count < 256){
            put_header_array[count] = 0;
            count++;
        }

        // ヘッダを書き出す
        count = 0;
        while(count < 256){
            bw->Write(put_header_array[count]);
            count++;
        }

        // データ部分を書き出す。データ部は、Raw データと同じ
        SaveRAW(bw, data);

        bw->Close();
        fs->Close();
```

```
        return true;
    }
```

DICOM 画像変換ツールマニュアル

2010 年 6 月 10 日 第 1.0 版発行

製作・著作　株式会社 i-Plants Systems

info@i-plants.jp

ve_support@i-plants.jp（VE サポート専用窓口）

019 – 694 – 3103（代表）

http://www.i-plants.jp/hp