Particle Data Manegement Library

User's Guide

Ver. 1.1.0

Advanced Visualization Research Team
Advanced Institute for Computational Science
RIKEN

7-1-26, Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo, 650-0047, Japan

`http://aics.riken.jp/`

March 2014

Release

初版	20 March	2014
第二版	3 October	2014
第三版	8 October	2014
第三版	28 September	2015

目次

第 1 章

パッケージのビルド

本章では、PDMlib のビルド方法について記述します。

1.1 PDMlib が依存するライブラリのビルド

PDMlib は以下の 5 つのライブラリに依存しているため、これらのライブラリがシステムにインストールされていない場合は PDMlib 自身のビルドの前に、これらのライブラリをビルドする必要があります。

- zlib
- fpzip
- Zoltan
- TextParser
- HDF5(optional)

HDF5 ライブラリは、H5PartConverter のみが依存しています。したがって、HDF5 ライブラリが無い場合でも PDMlib 本体や FV14Converter は利用することができます。

1.1.1 zlib のビルド

zlib は一般的な Linux ディストリビューション向けにバイナリパッケージが用意されていますので、そちらを使用してください。

1.1.2 fpzip のビルド

fpzip を配布元の URL(http://computation.llnl.gov/casc/fpzip/fpzip.html) よりダウンロード、展開した後に以下のコマンドを順に実行してください。

```
$cd src
$make
```

makefile 内では C++ コンパイラとして g++ が指定されているので、これ以外のコンパイラを使用する場合は CXX および CXXFLAGS 変数を変更してから make コマンドを実行してください。

正常にビルドできた場合は、” src ” と同じレベルにある”lib”ディレクトリの下に libfpzip.a が生成されています。

install スクリプトは用意されていませんので、以下の各ファイルを適切なディレクトリにコピーしてください。

- inc/fpzip.h
- lib/libfpzip.a

1.1.3　Zoltan のビルド

Zoltan を配布元の URL(http://www.cs.sandia.gov/ web1400/1400_download.html) からダウンロードし以下のコマンドを順に実行してください。

```
$tar xfz zoltan_distrib_v3.8.tar.gz
$cd Zoltan_v3.8
$mkdir BUILD
$cd BUILD/
$../configure --with-id-type=ulong --with-mpi-compilers=yes
$make everything
#make install
```

1.1.4　TextParser のビルド

TextParser のビルド方法は、TextParser に同梱されているドキュメントをご参照ください。

1.1.5　HDF5 のビルド

HDF5 ライブラリは一般的な Linux ディストリビューション向けにバイナリパッケージが用意されていますので、そちらを使用してください。

1.2　PDMlib のビルド

PDMlib 本体のビルドには Cmake を利用します。ビルドの実行

以下のコマンドを順に実行することで、ヘッダファイル、ライブラリ本体、付属ツール類がインストールされます。実行例中の PATH_TO_PDMLIB_DIR には CMakeLists.txt ファイルがあるディレクトリを指定してください。

```
$mkdir BUILD
$cd BUILD
$cmake PATH_TO_PDMLIB_DIR
#make install
```

cmake コマンドの実行時に以下のように-D オプションを使用して、PDMlib のコンパル設定を変更することができます。

```
$cmake -DCMAKE_CXX_COMPILER=mpiicpc -Dbuild_tests=on
```

設定可能なオプションは、表 1.1 に示すとおりです。なお、FPZIP_ROOT, ZOLTAN_ROOT, TP_ROOT については、/usr/local, /opt 等の一般的なパスにインストールしている場合は明示的に設定しなくても、cmake により自動的に検出されます。標準の検索対象のパスは/usr/shar/cmake-*/Modules/Platform/UnixPaths.cmake というファイル内に記述されています。

1.3　インストールスクリプト

fpzip、Zoltan、TextParser および PDMlib 自身のビルドをまとめて行うシェルスクリプト”install.sh”が、ソースコードに同梱されています。install.sh と同じディレクトリに各ライブラリのアーカイブまたはアーカイブを展開したソースコード一式を置いた状態で install.sh を実行すると、これらのライブラリを一括してインストールします。また、依

表 1.1　PDMlib のビルドオプション

オプション	内容	デフォルト値
CMAKE_CXX_COMPILER	使用する C++ コンパイラ	mpicxx
CMAKE_CXX_FLAGS	C++ コンパイラに渡すオプション	-g
CMAKE_INSTALL_PREFIX	PDMlib のインストール先	/usr/local
build_samples	サンプルコードをコンパイルするかどうか指定するフラグ (on or off)	off
build_tests	テストプログラムをコンパイルするかどうか指定するフラグ (on or off)	off
FPZIP_ROOT	fpzip がインストールされているパス	
ZOLTAN_ROOT	Zoltan がインストールされているパス	
TP_ROOT	TextParser がインストールされているパス	

存するライブラリがインストール済の場合は、インストール先ディレクトリを指定することでビルドせずにそのライブラリを使うこともできます。実行時オプションなどは、以下のコマンドを実行すると表示されるヘルプメッセージをご参照ください。

```
>./install --help
```

第 2 章

PDMlib の機能説明

本章では PDMlib の機能を解説します。

2.1 データ形式

PDMlib が対象とする粒子系のシミュレーションプログラムでは、データサイズが計算対象の粒子数に比例するため、一般的にはテキストファイルとして管理することが困難なサイズになります。一方、バイナリファイルは入出力の効率やサイズの面でテキストファイルより有利ですが、データの内容を確認するために専用のアプリケーションが必要になり一般的なテキストエディタやページャーなどのツールで手軽に見ることができないという難点があります。

そこで、PDMlib ではソルバが使用するデータを次の 2 種類に分けて取り扱います。

フィールドデータ
: 粒子の座標や物理量のような、いわゆるデータ

DFI
: 各データの名前や型、実行時のプロセス数や解析領域の Bounding Box などの付加的な情報

フィールドデータはデータの種類毎にコンテナとして管理します。コンテナには、粒子数分のスカラ量またはベクトル量を、表 2.1 に示すいずれかの型の値として登録することができます。

表 2.1　PDMlib がサポートするデータ型一覧

int	32bit 整数
long	64bit 整数
uint	32bit 符号無し整数
ulong	64bit 符号無し整数
float	32bit 実数
double	64bit 実数

各コンテナのファイルは独自形式のバイナリファイルで、各 Rank がそれぞれコンテナ毎、タイムステップ毎に個別のファイルに出力します。また、各コンテナには表 2.1 に示す付加情報を追加することができますが、これらの情報はコンテナのファイルではなく、DFI ファイルに記録されます。DFI ファイルは TextParser 形式のテキストファイルで、Rank0 のみがファイル出力を行います。タイムステップ毎の情報は全て同じファイルに追記されるので、1 回の実行につき 1 つのファイルが生成されます。

ファイルフォーマットの詳細は第 5 章をご参照ください。

表 2.2　コンテナ情報の一覧

変数名	値の意味
Name	コンテナに格納する値の名前
Annotation	このコンテナに対する注釈
Compression	コンテナに対する圧縮形式の指定
Type	コンテナに格納された値の型
Suffix	コンテナファイルの拡張子
nComp	値がスカラー量 (2) がベクトル量（3) か
VectorOrder	ベクトル量の場合に個々のベクトルを要素毎に格納するか

2.2　ファイル圧縮機能

PDMlib では次の 3 種類の圧縮形式をサポートしています。

zip
: zlib に含まれる Deflate アルゴリズムを用いて圧縮/伸張を行います。

fpzip
: 米国ローレンス・リバモア国立研究所で開発された fpzip (http://computation.llnl.gov/casc/fpzip/fpzip.html) を用いて圧縮/伸張を行います。

RLE
: zlib に含まれる RLE (Run Length Encoding) を用いて圧縮/伸張を行います。

これらの圧縮形式はコンテナ毎に選択することができ、複数の形式による圧縮を重ねて行うこともできます。

2.3　MxN ロード機能

PDMlib には読み込み時に Rank 間での粒子数のインバランスを解消する、MxN ロード機能が含まれています。この機能は米国サンディア国立研究所にて開発された Zoltan (http://www.cs.sandia.gov/zoltan/) を用いて実現されています。

実行時のプロセス数を M とすると、あるタイムステップでの PDMlib の出力ファイルは M 個 (x コンテナ数) に分かれています。リスタート実行時のプロセス数を N とすると、M¡N の時は読み込む対象のファイルが存在しないプロセスが生じますし、M¿N かつ M が N で割り切れない時は他の Rank より多くのファイルを読み込む Rank が生じます。

例えば、4 プロセスで実行した結果を 3 プロセスで読み込もうとすると、図 2.3 のように、rank0 が他 Rank に比べて 2 倍の粒子の計算を担当することになります。

PDMlib の MxN ロード機能を用いると、図 2.3 のようにマイグレーション処理 (計算対象の粒子を移動させること) を行ない各 Rank が担当する粒子数が均等になるようにします。

どの粒子をどのプロセスへマイグレーションさせるかの判定には、Zoltan に組込まれている Recursive Coordinate Bisection (RCB) method [1] を用いています。この手法は、粒子の座標を元になるべく近傍にある粒子が同じ Rank に割り当てられるように配置します。その際粒子ごとの重み付けは行なわず全ての粒子を均等に各 Rank へ割り振っています。また、Rank 間でのインバランスが 10% 以下の場合は、均等に配置されているものと見なしてマイグレーション処理を行ないません。

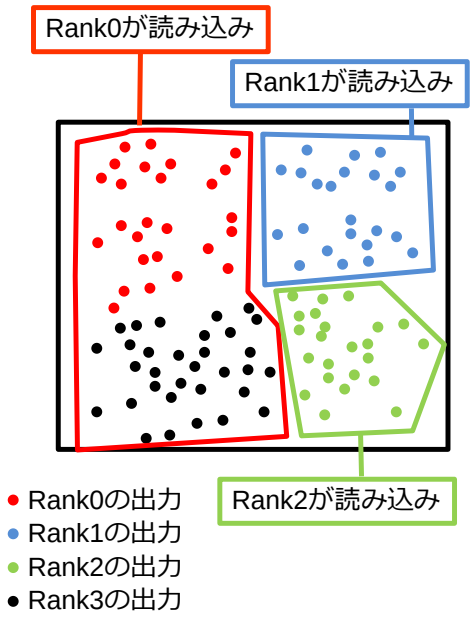

図 2.1　MxN ロード機能未使用時のデータ分散の様子

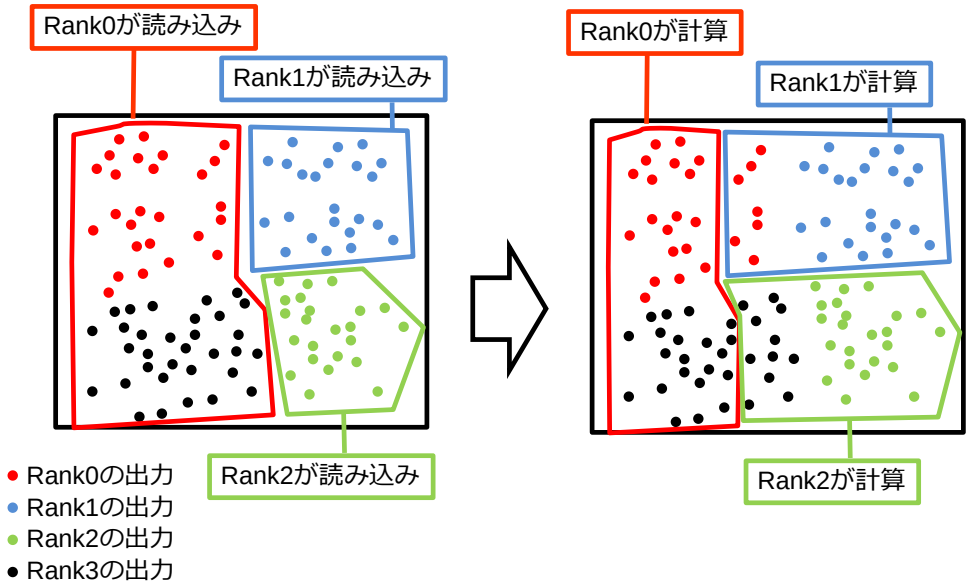

図 2.2　マイグレーション処理の例

第 3 章

API 利用方法

本章では PDMlib の API の利用方法について記述します

3.1 ユーザプログラムへの組込み

ユーザプログラムへ組込む際には、以下の 3 つのソース修正を行う必要があります。

- ヘッダファイルのインクルード
- 初期化処理の組込み
- 入力機能の組込み（optional)
- 出力機能の組込み

3.2 ヘッダファイルのインクルード

PDMlib のヘッダファイルは”PDMlib.h”のみです。PDMlib の API を呼び出すソースコードに以下の include 文を追加してください。

```
#inclde <PDMlib.h>
```

3.3 PDMlib の API 全般に関する注意

PDMlib はシングルトンパターンを使用しており、ユーザコード内で明示的にインスタンスを生成することはできません。PDMlib の API を呼び出す時は、静的メンバ関数である PDMlib::GetInstance() が返す PDMlib オブジェクトの参照を通じて関数を呼び出してください。

3.4 初期化処理の組込み

PDMlib の初期化処理は以下の手順で行います。

1. Init 関数の呼び出し
2. コンテナ情報の設定
3. その他のオプション設定

3.4.1　Init 関数の呼び出し

PDMlib の API を使うには、まず以下の Init 関数を呼び出す必要があります。Init 関数を呼び出すまでは、どの PDMlib の API も機能しません。

```
void Init(const int& argc, char** argv, const std::string& WriteMetaDataFile,
const std::string& ReadMetaDataFile = "");
```

第 3 引数には出力に用いる DFI ファイルの名前を、第 4 引数には入力に用いる DFI ファイルの名前を指定してください。入力機能を用いない場合は、第 4 引数はデフォルト値 (””) のままで構いません。

3.4.2　コンテナ情報の設定

表 2.1 に記載したコンテナ情報は、PDMlib 内部では以下に示す ContainerInfo 構造体として表わされています。

```
struct ContainerInfo
 {
  std::string Name;
  std::string Annotation;
  std::string Compression;
  SupportedType Type;
  std::string Suffix;
  int nComp;
  StorageOrder VectorOrder;
};
```

コンテナ毎に ContainerInfo 構造体のインスタンスを作成し、値を設定した上で、次の AddContainer 関数を用いて、使用するコンテナ情報を登録してください。

```
int AddContainer(const ContainerInfo& Container);
```

なお、ContainerInfo 構造体のメンバは全て public ですので、値を設定したり読み取る時は個々のメンバを直接読み書きしてください。

既存のファイルを読み込む場合は自動的に既存のコンテナ情報が読み込まれるため、この処理は不要です。

3.4.3　その他のオプション設定

出力するフィールドデータのベースファイル名を以下の関数によって設定することができます。実際のファイル名は、この関数で指定された文字列 (FileName) と、Rank 番号、タイムステップ数、ContainerInfo 構造体のメンバ Suffix をもとに決められ、”FileName_Rank 番号_タイムステップ数. 拡張子”となります。

```
int SetBaseFileName(const std::string& FileName);
```

ただし、Rank 番号とタイムステップ数の部分は以下の関数の引数に”step_rank”を指定して実行することで入れ替えることができます。

```
int SetFileNameFormat(const std::string& format);
```

また、フィールドデータはデフォルトでは”pdm”ディレクトリ以下に生成されます。以下の関数を用いることで、このディレクトリを変更することができます。

```
int SetPath(const std::string& path);
```

なお、path には絶対 PATH または DFI ファイルが出力されたディレクトリからの相対パスを指定することができます。

解析領域全体の BoundingBox を次の関数によって設定することができます。bbox は 6 要素の配列で、先頭から順に、頂点 1 の x 座標, y 座標, z 座標, 頂点 2 の x 座標, y 座標, z 座標 を格納して渡してください。

```
void SetBoundingBox(double bbox[6]);
```

PDMlib を MPI_COMM_WORLD 内の全プロセスではなく、特定のグループ内のプロセスのみが使う場合は、次の関数によって PDMlib を呼び出すグループのコミュニケータを設定してください。

```
void SetComm(const MPI_Comm& comm);
```

PDMlib がフィールドデータの出力時に使うバッファの最大サイズを次の関数によって指定することができます。

```
void SetBufferSize(const int& BufferSize);
```

PDMlib がフィールドデータの出力時に行うバッファリングの最大回数を次の関数によって指定することができます。

```
void SetMaxBufferingTime(const int& MaxBufferingTime);
```

SetBufferSize で指定された容量を越えるか、SetMaxBufferingTime で設定された回数を越えるまで、フィールドデータの出力は PDMlib 内部のバッファに保持され実際にファイル出力は行なわれません。ただし、どちらかの条件を満たした場合でも明示的な flush 操作はしていませんので、さらに OS レベルでバッファリングが行なわれている場合はファイルに出力されない可能性があります。

なお、圧縮が指定されたコンテナは、圧縮後の状態で内部バッファに保持されます。

3.5　入力機能

PDMlib の入力機能に関する API は次の 3 種類があります。

1. コンテナ情報の読み取り
2. フィールドデータの読み取り
3. フィールドデータの一括読み取り

3.5.1　コンテナ情報の読み取り

以下の関数を使うことで、DFI ファイル内で指定されているコンテナ情報の一覧を取得することができます。

```
std::vector<ContainerInfo> GetContainerInfo(void);
```

3.5.2　フィールドデータの読み取り

以下の関数により指定されたコンテナのフィールドデータを読み取ります。

```
template<typename T> int Read(const std::string& Name, size_t* ContainerLength,
                              T** Container, int* TimeStep = NULL,
                               bool read_all_files = false);
```

Name で指定した名前のコンテナを**Container で指定した領域に対して読み込みます。**Container が NULL の時はライブラリ内部で動的に領域を確保して返すので、ユーザコード側で必ず delete してください。なお、Container 引数には、ポインタではなくポインタのポインタを渡す必要があるので注意してください。

TimeStep として NULL ポインタが渡された場合や*TimeStep が負の値だった場合は、ディレクトリ内の最新のタイムステップのデータを読み込みます。それ以外の場合は、指定された TimeStep のデータを読み込みます。

オプション引数の read_all_files が true の時は全 Rank が重複して全てのファイルを読み込みます。デフォルトでは、ファイル単位でブロック分割して自 Rank が担当するファイルのみを読み込むので、これ以外の方法でデータ分散を独自に行う場合にお使いください。

3.5.3　フィールドデータの一括読み取り

Read() 関数を用いる方法では、1 コンテナづつファイルを読み込みますが、読み込み対象のコンテナを事前に登録しておいて 1 回のルーチンコールで複数のコンテナを読み込むこともできます。この場合は読み込んだデータに対してロードバランスを調整するために、マイグレーション処理を行なうこともできます。

コンテナの登録には以下の関数を用います。

```
template<typename T> int RegisterContainer(const std::string& Name, T** Container) const;
```

Name および Container の意味は PDMlib::Read() と同じです。

一括読み取りには以下の関数を用います。

```
size_t ReadAll(int* TimeStep = NULL, const bool& MigrationFlag = false,
               const std::string& CoordinateContainer = "Coordinate");
```

引数の TimeStep は Read() と同様です。

MigrationFlag に true を設定すると、2.3 章に記載したマイグレーション処理を行います。

マイグレーションを行う場合は、CoordinateContainer に座標データを保持するコンテナの名前を指定してください。マイグレーションの要否の判断や、どの粒子をどの Rank へマイグレーションするかを決めるために、ここで指定されたコンテナに含まれるデータを使います。なお、マイグレーションを行わない場合は、この引数は未指定のままでも正常に動作します。

3.6　出力機能

PDMlib のファイル出力機能は以下の関数を呼び出すことで使用します。PDMlib::Write() はコンテナ毎に呼び出す必要がありますが、引数として渡されたコンテナ情報を出力するだけではなく DFI ファイルも同時に更新します。

```
template<typename T> int Write(const std::string& Name, const size_t& ContainerLength,
                               T* Container, T MinMax[8], const int& NumComp,
                               const int& TimeStep, const double& Time);
```

引数の MinMax には、出力するコンテナの最小/最大値に関するデータを登録することができます。この引数に NULL 以外の値を設定した場合は、DFI ファイルの末尾にそのタイムステップの最小/最大値として情報が追記されます。MinMax の各要素に何の値を指定するかは表 3.1 をご参照ください。

表 3.1　MinMax に設定する値

	スカラーの場合	ベクトルの場合
MinMax[0]	最小値	最小値
MinMax[1]	最大値	最大値
MinMax[2]	不使用	x 成分の最小値
MinMax[3]	不使用	x 成分の最大値
MinMax[4]	不使用	y 成分の最小値
MinMax[5]	不使用	y 成分の最大値
MinMax[6]	不使用	z 成分の最小値
MinMax[7]	不使用	z 成分の最大値

第 4 章

付属ユーティリティ

本章では、PDMlib に付属している 2 種類のツールに関して記述します。

4.1 ステージングツール

京コンピュータでリスタート計算を行う時に前回の計算結果をどのプロセスに送り込むかを指定するステージング指示行を生成します。実行方法は以下のとおりです。

```
$ python2 stage.py NPROC DFI_FILE
```

NPROC には、これから実行する時に使用するプロセス数を、DFI_FILE には、リスタート計算に用いる計算結果の DFI_FILE をそれぞれ指定してください。

4.2 ファイルコンバータ

PDMlib の出力を可視化ソフトから読める形式に変換する 3 種類のコンバータが付属しています。

FV14Converter
: FieldView version 14 以降で使われる Particle Path 形式へのコンバータ

H5PartConverter
: VisIt などで読み込むことができる H5Part 形式へのコンバータ

VtkConverter
: ParaView などで読み込むことができる Vtk 形式 (XML format) へのコンバータ

いずれのコンバータもに MPI 並列化されており、ファイル単位でのデータ分散を行っていますが、FV14Converter および H5Part Converter には以下の制限があります。

FieldView v14 の ParticlePath 形式では解析領域内に存在する全粒子を 1 タイムステップあたり 1 ファイルに出力する必要があります。このため、FV14Converter のデータ分散は時間方向でブロック分割を行なっています。

VisIt の H5Part reader にはこのような制限は無く同一ステップのデータが複数のファイルに分散されていても読み込むことができますが、各ファイルに TimeStep0 のデータが含まれていなければ正常に読み込めません。このため、H5PartConverter はソルバ実行時のデータ分散をそのまま使用し、1 つのファイルに全ステップのデータを出力しています。

なお、プロセス数を変更してリスタート計算を行なった場合のように、タイムステップによってプロセス数（=フィールドデータファイル数）が異なる場合は、全タイムステップを通じた、最小のプロセス数でコンバータが動作するように制限されています。これは、同一ファイル内に不連続なタイムステップが存在すると、VisIt での描画に不具合が生じていたため、この現象を回避するための対策です。

4.2.1　FV14Converter の使用方法

FV14Converter を使うには次のコマンドを入力します。

```
$FV14Converter -f DFI_FILE {-c Coordinate Container name} \
{-s start time} {-e end time} {-d directory} {-b}
```

各オプションの意味は表 4.1 のとおりです。

表 4.1　FV14Converter に指定可能なオプション一覧

オプション	オプションの意味	デフォルト値
-f	計算結果の DFI ファイル	
-c	座標コンテナの名前	Coordinate
-s	変換対象の開始ステップ	-1
-e	変換対象の最終ステップ	INT_MAX
-b	解析領域全体の BoundingBox 情報を出力するかどうかのスイッチ	

-f オプションは指定が必須のオプションで、指定されなかった場合はエラー終了します。

-s オプションに負の数を指定した場合は、指定されたディレクトリ内に存在する最小ステップ以降を変換対象とします。

指定されたディレクトリに-e オプションで指定されたタイムステップよりも小さいステップ数のファイルしか存在しない場合は、ディレクトリ内に存在する最大のタイムステップまでを変換の対象とします。

-b オプションが指定された時の BoundingBox 情報は、FV-UNS(text) 形式として別のファイルに出力されます。

4.2.2　H5PartConverter の使用方法

H5PartConverter を使うには次のコマンドを入力します。

```
$H5PartConverter -f DFI_FILE {-c Coordinate Container name} \
{-s start time} {-e end time} {-d directory} {-b}
```

各オプションの意味は表 4.2 のとおりです。

表 4.2　H5PartConverter に指定可能なオプション一覧

オプション	オプションの意味	デフォルト値
-f	計算結果の DFI ファイル	
-c	座標コンテナの名前	Coordinate
-s	変換対象の開始ステップ	-1
-e	変換対象の最終ステップ	INT_MAX

なお、-b オプションが無いことを除いて全て FV14Converter と同じオプションが使用できます。

4.2.3　VtkConverter の使用方法

VtkConverter を使うには次のコマンドを入力します。

```
$ VtkConverter DFI_File {-c Coordinate Container name} \
{-s start time} {-e end time}  {-f format}
```

各オプションの意味は表 4.3 のとおりです。-b オプションには ”ascii”または”binary”のいずれかを指定することがで

表 4.3　VtkConverter に指定可能なオプション一覧

オプション	オプションの意味	デフォルト値
-c	座標コンテナの名前	Coordinate
-s	変換対象の開始ステップ	-1
-e	変換対象の最終ステップ	INT_MAX
-b	変換後のファイル形式	ascii

きます。なお、本コンバータのみ、DFI ファイルの指定には-f オプションを使わず、引数として渡します。

第 5 章

ファイル仕様

本章では PDMlib が出力するファイルの仕様について記述します。

5.1 フィールドデータファイルの仕様

フィールドデータファイルは、ファイルの先頭から順に表 5.1 に示す形式で、小規模なヘッダと実データが記録されています。コンテナ部分のデータ型の情報などのメタデータは前述の DFI ファイルにしか含まれていないため、フィールドデータファイルのみでは正常にデータを読み取ることはできません。

表 5.1 フィールドデータファイルのフォーマット

サイズ (word)	型	値の意味
1	char	sizeof(int)
1	char	sizeof(siez_t)
1	int	エンディアン判定に用いるマジックナンバー
1	size_t	格納されているデータの元のサイズ（Byte)
1	size_t	格納されているデータの実際のサイズ (Byte)
*	*	実データ

第 3 バイト以降に書かれているマジックナンバーの実際の値は (9615) です。読み込み時にこの値が正常に読めるかどうかによって、出力された時のエンディアンが現在の処理系と同じか違うかを確認しています。マジックナンバーの次に出力される size_t の値は、そのファイルに含まれるデータの圧縮前のサイズで、その次に出力される size_t の値は圧縮後のサイズです。したがって圧縮されていない場合はこの 2 つは同じ値になります。

5.2 DFI ファイルの仕様

DFI ファイルは TextParser 形式のファイルで表 5.2 のようなノード構成を持ちます。

ContainerList ノードの下には、出力するコンテナの情報が出力されます。1 コンテナが 1 ノードに相当し、ノード名はコンテナ名となります。その下に ContainerInfo 構造体の Name 以外のメンバがリーフとして記述されます。

TimeSlice 情報には、各ステップごと/コンテナごとの MinMax 値が出力されます。もし PDMlib::Write() の出力時に MinMax 値が設定されていなければ何も出力されません。

表 5.2　DFI ファイルのフォーマット

Header	Version	PDMlib のバージョン
	Endian	出力時のエンディアン
	Prefix	出力するファイルの BaseFileName
	DirectoryPath	フィールドデータを格納するディレクトリ
	FieldFilenameFormat	フィールドファイルのファイル名で rank 番号とタイムステップのどちらを先にするかを指定する
	NumContainer	コンテナの数
ContainerList	内容は本文を参照	
MPI	NumCommWorldProc	MPI_COMM_WORLD 内のプロセス数
	Communicator	PDMlib を使うプロセスが所属するコミュニケータ
	NumProc	上記のコミュニケータに所属するプロセス数
DomainInfo	NumProc	
	GlobalOrigin	解析領域の原点座標
	GlobalRegion	解析領域のサイズ
	BoundingBox	解析領域の BoundingBox となる 2 点の座標
TimeSlice	内容は本文を参照	

参考文献

[1] M. Berger and S. Bokhari. ”A partitioning strategy for nonuniform problems on multiprocessors.” IEEE Trans. Computers, C-36 (1987) 570-580.