

Unstructured Data Model Library

User's Manual

Version 1.0.0

2014 年 12 月

目次

0. この文章について	6
1. UDM ライブラリについて	7
1.1 概要	8
1.2 動作環境	9
1.3 必須ライブラリ	10
2. パッケージのビルド	11
2.1 パッケージ構造	11
2.2 必須ライブラリのビルド	13
2.2.1 CGNS ライブラリ	13
2.2.2 Zoltan ライブラリ	15
2.2.3 TextParser	17
2.3 UDM ライブラリのビルド	19
2.4 configure スクリプトのオプション	24
2.5 configure 実行時オプションの例	28
2.6 udm-config コマンド	31
2.7 提供環境の作成	32
2.8 京コンピュータでのビルド	33
2.8.1 CGNS ライブラリ	33
2.8.2 Zoltan ライブラリ	33
2.8.3 TextParser	34
2.8.4 UDM ライブラリ	34
3. UDM ライブラリの利用方法	36
3.1 C/C++によるコンパイル	36
3.2 利用例 (UDMLib-X.X.X/examples)	38
3.2.1 CGNS ファイルの入出力 (cgns)	38
3.2.2 Zoltan による分割 (partition)	45
3.2.3 時系列 CGNS ファイルの出力 (timeslice)	50
3.2.4 DFI ファイルの入出力 (dfi)	67
4. UDM ライブラリ API の使用方法	68
4.1 UDM モデルの構成要素	69
4.1.1 UDM モデルの生成・破棄	70
4.1.2 UDM ゾーンの生成、取得	72
4.1.3 節点 (ノード) 座標クラスの取得	74
4.1.4 UDM セクション管理クラスの取得	74
4.1.5 UDM セクションの生成、取得	75

4.2 DFI ファイルの入出力	78
4.2.1 DFI ファイルの読込	79
4.2.2 DFI ファイルの書込	81
4.2.3 DFI:ファイル情報.....	83
4.2.5 DFI:単位系情報.....	98
4.2.6 CGNS ファイルの読込	103
4.3 節点（ノード）	105
4.3.1 節点（ノード）の生成・追加	105
4.3.2 節点（ノード）の取得	107
4.3.3 節点（ノード）の構成座標.....	108
4.3.4 ランク間接続節点（ノード）	110
4.4 要素（セル）	114
4.4.1 要素（セル）の生成・追加.....	115
4.4.2 要素（セル）の取得.....	118
4.4.3 構成節点（ノード）情報.....	120
4.5 節点（ノード）・要素（セル）の物理量.....	122
4.5.1 物理量定義.....	123
4.5.2 節点（ノード）の物理量の取得.....	130
4.5.2 節点（ノード）への物理量の設定	133
4.5.3 要素（セル）の物理量の取得	136
4.5.4 要素（セル）への物理量の設定.....	140
4.6 分割実行.....	144
4.6.1 分割実行	145
4.6.2 分割パラメータ	146
4.6.3 分割重みの取得、設定、クリア	152
4.7 仮想要素（セル）の転送	156
4.8 隣接節点（ノード）・要素（セル）	158
4.8.1 隣接節点（ノード）の取得.....	158
4.8.2 隣接要素（セル）の取得.....	160
4.9 ユーザ定義情報.....	164
4.9.1 ユーザ定義情報の取得	164
4.9.2 ユーザ定義情報の設定	167
4.9.3 ユーザ定義情報の削除	169
4.10 データ型定義、列挙型.....	171
4.10.1 データ型	171
4.10.2 列挙型.....	172

5. UDM ライブラリ API 一覧	176
6. ファイル仕様	183
6.1 入出力設定ファイル : index.dfi	184
6.1.1 FileInfo : ファイル情報の設定項目	186
6.1.2 FilePath : ファイルパス情報	193
6.1.3 UnitList : 単位系情報	193
6.1.4 TimeSlice : 時系列情報	194
6.1.5 FlowSolution : 物理量定義情報	195
6.2 プロセス情報ファイル : proc.dfi	200
6.2.1 Domain : ドメイン情報	200
6.2.2 MPI : 並列情報	201
6.2.2 Process : プロセス情報	201
6.3 分割パラメータファイル : udmlib.tp	203
6.4 CGNS ファイル	205
6.4.1 CGBSBase_t : CGNS ベースノード	207
6.4.2 SimulationType_t : CGNS シミュレーションタイプ	207
6.4.3 UdmInfo : UDM ライブラリ情報	208
6.4.4 BaseIterativeData_t : CGNS ベース時系列情報	208
6.4.5 Zone_t : CGNS ゾーン	209
6.4.6 ZoneType_t : ゾーンタイプ	210
6.4.7 ZoneIterativeData_t : CGNS ゾーン時系列情報	211
6.4.8 GridCoordinates_t : CGNS グリッド座標	212
6.4.9 Elements_t : CGNS セクション	213
6.4.10 FlowSolution_t : CGNS 物理量データ	214
6.4.11 UdmRankConnectivity : 内部境界情報	215
6.4.12 UdmUserDefinedData : ユーザ定義情報	216
7. ステージングツール	217
7.1 ステージングツールのビルド	217
7.1.1 京フロントエンドでのステージングツールのビルド	218
7.2 使用方法	220
7.3 使用例	222
7.3.1 4 分割 CGNS ファイルを 6 分割で実行	222
7.3.2 8 分割 CGNS ファイルを 3 分割で実行	224
付録 1 : CGNS ライブラリのオプションビルド	228
付録 2 : Zoltan パラメータによる分割の違い	231

変更履歴

Version	日付	説明
0.1.0	2014/11/13	初版
0.1.1	2014/12/18	udmlib : UdmEntity, 部品要素クラス構成変更 doc : API, ファイル仕様追加
1.0.0	2014/12/24	version 1.0.0 リリース

0. この文章について

この文書は、非構造格子並列分散ライブラリ（以下、UDM ライブラリ）のユーザーマニュアルです。

1. UDM ライブラリについて

東京大学生産技術研究所 革新的シミュレーション研究センターにて研究開発を進めている「HPC/PF (High Performance Computing Platform)」システムは、高度なシミュレーション群による迅速な問題解決のサービスを提供するシステムとして開発されています。

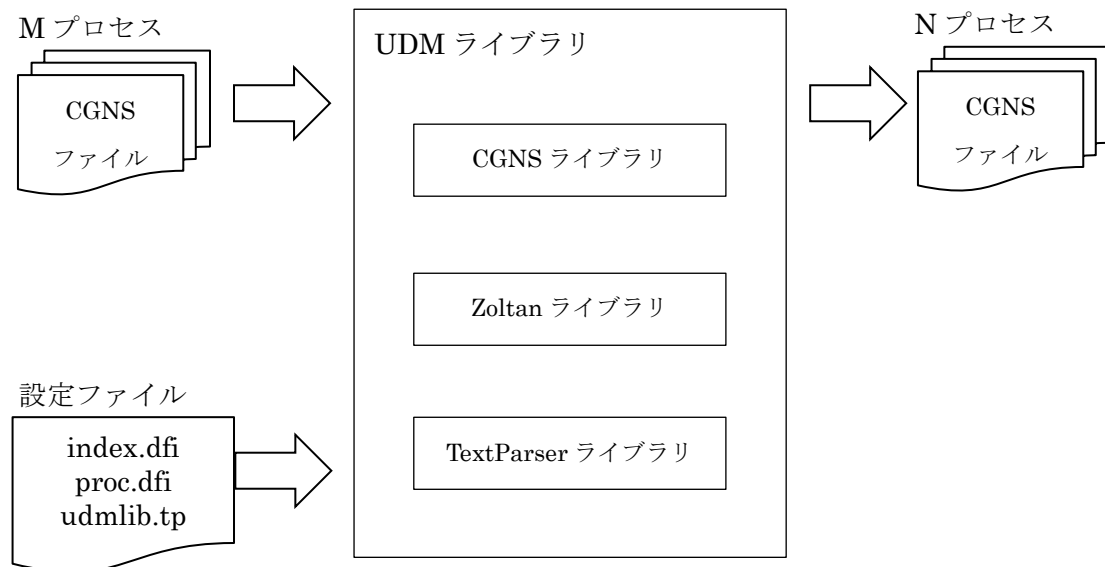
この HPC/PF システムは神戸に設置された「京」をはじめ、大学の情報基盤センターや汎用蔵ウドリソース、小規模なプライベートネットワークなど多様な計算機環境での稼働を想定しています。

UDM ライブラリは HPC/PF のソルバーやポスト処理で利用する非構造格子データの並列分散ファイル管理機能を有し、また、企業の実設計や研究機関における研究開発に資するため、ユーザのニーズに合致した機能や運用方法を提供します。

HPC/PF システムが提供するサービスは多岐にわたるため、既存のオープンソースシステムの再利用によりシステム全体の機能を発源させる方針で本ライブラリは作成しています。

1.1 概要

UDM ライブラリは、M プロセスにて実行されている非構造格子モデルを N プロセスに再分割する機能を提供します。



- (1) M プロセスにて M 々の CGNS ファイルを読み込み、M 分割された非構造格子モデルを形成する。
- (2) UDM ライブラリは設定ファイルに従って、N プロセスにモデルを再分割する。
- (3) N プロセスにて N 々の CGNS ファイルを出力する。

1.2 動作環境

UDM ライブラリは、以下の環境について動作を確認しています。

- Linux/Intel コンパイラ
CentOS6.4 i386/x86_64
Intel C/C++ Compiler Version 13 (icc/icpc)
- MacOS X Snow Leopard
MacOS X Snow Leopard
Intel C/C++ Compiler Version 13 (icc/icpc)
- 京コンピュータ

UDM ライブラリにて分割を行う必要メモリは以下です。

モデル	節点（ノード） 数	要素（セル）数	必要メモリ	
			ロード	分割実行
64x64*64	262144	250047	0.3GB	0.6GB
128x128x128	2097152	2048383	2.5GB	5.0GB
256*256*256	16777216	16581375	20GB	40GB

分割実行を行う場合は、モデル節点（ノード）、要素（セル）の情報を転送する為に、ロードメモリの x2 が必要となります。

（例）128x128x128 モデルの 1x4 分割の場合

- ランク 0 : 2.5GB x 2 = 5.0GB (128x128x128 のロードメモリ x 2)
- ランク 1 : 2.5GB / 4 x 2 = 1.3G (128x128x128 / 4 のロードメモリ x 2)
- ランク 2 : 2.5GB / 4 x 2 = 1.3G (128x128x128 / 4 のロードメモリ x 2)
- ランク 3 : 2.5GB / 4 x 2 = 1.3G (128x128x128 / 4 のロードメモリ x 2)

1.3 必須ライブラリ

UDM ライブラリは、以下の外部ライブラリを必要とします。

(1) MPI ライブラリ

並列・分散プロセス間のメッセージングライブラリです。

MPICH、又は OpenMPI のライブラリを使用可能です。

MPICH-2.0 以上

又は OpenMPI-1.6 以上

(2) CGNS ライブラリ

数値流体力学(CFD)用の階層型データベース管理および I/O サブルーチンのライブラリです。

CGNS - CFD General Notation System

<http://www.cgns.org>

Version 3.1.2 以上

(3) Zoltan ライブラリ

ロードバランス、分割、及びカラーリング機能を提供するライブラリです。

Zoltan Toolkit for Load-balancing, Partitioning, Ordering and Coloring

<http://www.cs.sandia.gov/Zoltan/Zoltan.html>

Version 3.8 以上

(4) TextParser

TextParser 形式の ASCII ファイルの入出力を行うライブラリです。

TextParser - Text Parsing Library

<https://github.com/avr-aics-riken/TextParser>

2. パッケージのビルド

UDM ライブラリのビルド、及び必須ライブラリのビルド方法について説明します。

2.1 パッケージ構造

UDM ライブラリのパッケージは次のようなファイル名で保存されています。

UDMlib-X.X.X.tar.gz (X.X.X にはバージョンが入ります)

このファイルの内部には、次のようなディレクトリ構造が格納されています。

UDMlib-X.X.X

```
|-- AUTHORS
|-- COPYING
|-- ChangeLog
|-- INSTALL
|-- LICENSE
|-- Makefile.am
|-- Makefile.in
|-- NEWS
|-- README
|-- aclocal.m4
|-- config.h.in
|-- configure
|-- configure.ac
|-- depcomp
|-- doc
|-- examples
|   |-- cc
|   |-- cxx
|   `-- data
|-- include
|   |-- config
|   |-- model
|   |-- partition
|   `-- utils
|-- install-sh
|-- missing
```

```
|-- src
|   |-- config
|   |-- model
|   |-- partition
|   `-- utils
|-- tools
|   `-- udm-frm
`-- udm-config.in
```

これらのディレクトリ構成について以下に説明します。

(1) doc

この文書を含む UDM ライブラリの文書が収められています。

(2) examples

CC (C 言語) , CXX (C++言語) による UDM ライブラリを使用したサンプルソースコードが収められています。

make によりサンプルプログラムが作成されますが、**make install** によりインストールは行いません。

(3) include

UDM ライブラリのヘッダファイルが収められています。ここに収められたファイルは **make install** で `$prefix/include` にインストールされます。

(4) src

UDM ライブラリのソースファイルが収められています。

(5) tools/udm-frm

UDM ライブラリを用いたソルバを並列環境にて実行する上でのファイルリンクマッピングを行うプログラムです。

make install で `$prefix/bin` に "udm-frm" (実行モジュール) がインストールされます。

2.2 必須ライブラリのビルド

UDM ライブラリは以下の外部ライブラリを必須とします。

ライブラリ名	機能説明
MPI ライブラリ	並列プログラミングの為のインターフェイス
CGNS ライブラリ	数値流体力学(CFD)用の階層型データベース
Zoltan ライブラリ	ロードバランス、分割、及びカラーリング API
TextParser	TextPaser 形式の ASCII ファイルの入出力

以下、必須ライブラリのビルド方法について説明します。

MPI ライブラリについては並列実行環境にインストールされているものとして、ここでの説明は割愛します。

2.2.1 CGNS ライブラリ

数値流体力学(CFD)用の階層型データベース管理および I/O サブルーチンのライブラリです。

以下 URL からライブラリをダウンロードしてください。

URL : <http://cgns.sourceforge.net/download.html>

ダウンロードファイル : CGNS Version 3.2.1 (cgnslib_3.2.1.tar.gz)

(2014/11/01 現在最新バージョン)

以下のコマンドにてビルド、インストールを行います。

これは、UDM ライブラリに必要な最低限の CGNS ライブラリのビルドです。CGNS ツール、HDF5 のサポートのビルドについては、「付録 1」を参照してください。

(/usr/local/cgnslib_3.2.1 にインストールするものとして、説明します。)

```
$ tar xzvf cgnslib_3.2.1.tar.gz
$ ls cgnslib_3.2.1
CMakeLists.txt  cmake_uninstall.cmake.in  install.lyx  license.txt  readme.txt
changelog       fortran_test               install.txt  readme.lyx   src
$ cd cgnslib_3.2.1/src
$ ./configure --prefix=/usr/local/cgnslib_3.2.1 --enable-64bit
$ make
$ sudo make install
$ tree /usr/local/cgnslib_3.2.1
cgnslib_3.2.1/
```

```
| -- include
|   |-- cgnsBuild.defs
|   |-- cgns_io.h
|   |-- cgnsconfig.h
|   |-- cgnslib.h
|   |-- cgnslib_f.h
|   |-- cgnstypes.h
|   |-- cgnstypes_f.h
|   `-- cgnswin_f.h
|-- lib
|   |-- libcgns.a
```

2.2.2 Zoltan ライブラリ

ロードバランス、分割、及びカラーリング機能を提供するライブラリです。
以下 URL からライブラリをダウンロードしてください。

URL : http://www.cs.sandia.gov/Zoltan/Zoltan_download.html

ダウンロードファイル : Version 3.8 (zoltan_distrib_v3.8.tar.gz)

(2014/11/01 現在最新バージョン)

以下のコマンドにてビルド、インストールを行います。
その他のビルドオプションについては、Zoltan ライブラリのマニュアルを参照してください。

(/usr/local/zoltan_v3.8 にインストールするものとして、説明します。)

```
$ tar xzvf zoltan_distrib_v3.8.tar.gz
$ cd Zoltan_v3.8
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/openmpi-1.6.5_gcc/lib
$ rm configure.ac
$ ./configure --prefix=/usr/local/zoltan_v3.8 \
    --enable-mpi \
    --with-mpi=/usr/local/openmpi-1.6.5_gcc (注)
$ make
$ sudo make install
$ tree /usr/local/zoltan_v3.8
zoltan_v3.8/
|-- include
|   |-- Makefile.export.zoltan
|   |-- Makefile.export.zoltan.macros
|   |-- Zoltan_config.h
|   |-- lbi_const.h
|   |-- zoltan.h
|   |-- zoltan_align.h
|   |-- zoltan_comm.h
|   |-- zoltan_comm_cpp.h
|   |-- zoltan_cpp.h
|   |-- zoltan_dd.h
|   |-- zoltan_dd_cpp.h
|   |-- zoltan_eval.h
```



```
| | -- zoltan_mem.h
| | -- zoltan_timer.h
| | -- zoltan_timer_cpp.h
| `-- zoltan_types.h
`-- lib
    `-- libzoltan.a
```

(注)

Zoltan ライブラリのビルドには **MPI** ライブラリが必要となりますので、適時
設定してください。

又、後述の **UDMLib** のビルド時と同一の **MPI** ライブラリを使用してください。

2.2.3 TextParser

TextParser 形式の ASCII ファイルの入出力を行うライブラリです。

以下 URL からライブラリをダウンロードしてください。

URL : <https://github.com/avr-aics-riken/TextParser>

ダウンロードファイル : TextParser-master.zip

(2014/11/01 現在最新バージョン)

以下のコマンドにてビルド、インストールを行います。

その他のビルドオプションについては、TextParser ライブラリのマニュアルを参照してください。

(/usr/local/textparser にインストールするものとして、説明します。)

```
$ unzip TextParser-master.zip
$ cd TextParser-master
$ ./configure --prefix=/usr/local/textparser
$ make
$ sudo make install
$ tree /usr/local/textparser/
textparser/
|-- bin
|   `-- tp-config
|-- doc
|   |-- program_description.pdf
|   |-- textparser_ug_en.pdf
|   `-- textparser_ug_jp.pdf
|-- include
|   |-- TextParser.h
|   |-- TextParser.inc
|   |-- TextParserCommon.h
|   |-- TextParserElement.h
|   |-- TextParserTree.h
|   `-- tpVersion.h
|-- lib
|   `-- libTP.a
`-- share
    |-- AUTHORS
```

```
| -- COPYING  
| -- ChangeLog  
| -- LICENSE  
|-- README
```

2.3 UDM ライブラリのビルド

いずれの環境でも `shell` で作業するものとします。以下の例では `bash` を用いていますが、`shell` によって環境変数の設定方法が異なるだけで、インストールの他のコマンドは同一です。適宜、環境変数の設定箇所をお使いの環境でのものに読み替えてください。

以下の例では、作業ディレクトリを作成し、その作業ディレクトリに展開したパッケージを用いて、ビルド、インストールする手順を示しています。

但し、オプションに必須ライブラリの指定が必要となりますので、後述の「`configure` スクリプトのオプション」、「`configure` 実行時オプションの例」を参照してください。

\$ mkdir work	(1)
\$ cp [DOWNLOAD_PATH]/UDMLib-X.X.X.tar.gz work	
\$ cd work	(2)
\$ tar xzvf UDMLib-X.X.X.tar.gz	
\$ cd UDMLib-X.X.X	
\$./configure [option]	(3)
\$ make	(4)
\$ make clean	(4)
\$ make	
\$ make distclean	(4)
\$./configure [option]	
\$ make	
\$ make install	(5)
\$ make uninstall	(6)

(1) 作業ディレクトリの構築とパッケージのコピー

まず、作業用のディレクトリを用意し、パッケージをコピーします。ここでは、カレントディレクトリに `work` というディレクトリを作り、そのディレクトリにパッケージをコピーします。

(2) 作業ディレクトリへの移動とパッケージの解凍

先ほど作成した作業ディレクトリに移動し、パッケージを解凍します。
パッケージの解凍により作成された `UDMLib-X.X.X` ディレクトリに移動します。

(3) `configure` スクリプトを実行

`configure` スクリプトを実行します。

`configure` スクリプトの実行時には、お使いの環境に合わせたオプションを指定する

必要があります。

configure オプションに関しては、後述の「**configure** スクリプトのオプション」、
「**configure** 実行時オプションの例」を参照してください。

configure スクリプトを実行することで、環境に合わせた **Makefile** が作成されます。

(4) **make** の実行

"**make**" コマンドを実行し、ライブラリをビルドします。

"**make**"コマンドを実行すると、次のファイルが作成されます。

```
src/libudm.a
```

ビルドをやり直す場合は、"**make clean**"を実行して前回の **make** 実行時に作成されたファイルを削除します。

また、**configure** スクリプトによる設定、**Makefile** の生成をやり直すには、"**make distclean**"を実行して、全ての情報を削除してから、**configure** スクリプトの実行からやり直します。

(5) インストール

"**make install**"により **configure** スクリプトの"**--prefix**"オプションで指定されたディレクトリに、ライブラリ、ヘッダファイルをインストールします。

但し、インストール先のディレクトリへの書き込みに管理者権限が必要な場合は、**sudo** コマンドを用いるか、管理者でログインして **make install** を実行します。

```
$ sudo make install
```

又は

```
$ su
password:
# make install
# exit
```

インストールされるファイル構成は以下の通りです。

```
udmlib-0.1.0/
|-- bin
|   |-- udm-config
|   `-- udm-frm
|-- doc
|   |-- UDMLib.pdf
```

```

|   `-- reference.pdf
|-- include
|   |-- UdmBase.h
|   |-- UdmErrorHandler.h
|   |-- UdmStaging.h
|   |-- config
|       |-- UdmConfigBase.h
|       |-- UdmDfiConfig.h
|       |-- UdmDfiKeywords.h
|       |-- UdmDfiValue.h
|       |-- UdmDomainConfig.h
|       |-- UdmFileInfoConfig.h
|       |-- UdmFilePathConfig.h
|       |-- UdmFlowSolutionListConfig.h
|       |-- UdmMpiConfig.h
|       |-- UdmProcessConfig.h
|       |-- UdmSettingsConfig.h
|       |-- UdmSolutionFieldConfig.h
|       |-- UdmTimeSliceConfig.h
|       |-- `-- UdmUnitListConfig.h
|-- model
|   |-- UdmBar.h
|   |-- UdmCell.h
|   |-- UdmCoordsValue.h
|   |-- UdmElements.h
|   |-- UdmEntity.h
|   |-- UdmEntityVoxels.h
|   |-- UdmFlowSolutions.h
|   |-- UdmGeneral.h
|   |-- UdmGlobalRankid.h
|   |-- UdmGridCoordinates.h
|   |-- UdmInfo.h
|   |-- UdmIterativeData.h
|   |-- UdmModel.h
|   |-- UdmNode.h
|   |-- UdmRankConnectivity.h

```

```

| | |-- UdmSections.h
| | |-- UdmShell.h
| | |-- UdmSolid.h
| | |-- UdmSolutionData.h
| | |-- UdmUserDefinedDatas.h
| | |-- UdmZone.h
| |-- partition
| | |-- UdmLoadBalance.h
| |-- udm_define.h
| |-- udm_errorno.h
| |-- udm_memutils.h
| |-- udm_mpiutils.h
| |-- udm_pathutils.h
| |-- udm_version.h
| |-- udmfrm_version.h
| |-- udmllib.h
| |-- utils
| | |-- UdmScannerCells.h
| | |-- UdmSearchTable.h
| | |-- UdmSerialization.h
| | |-- UdmStopWatch.h
|-- lib
| |-- libudm.a
|-- share
| |-- AUTHORS
| |-- COPYING
| |-- ChangeLog
| |-- LICENSE
| |-- NEWS
|-- README

```

(6) アンインストール

インストールした UDM ライブラリをアンインストールするには, "make uninstall" を行います。管理者権限が必要な場合は, `sudo` コマンドを用いるか、管理者でログインして `make uninstall` を実行します。

```
$ sudo make uninstall
```

又は

```
$ su  
password:  
# makeuninstall  
# exit
```


2.4 configure スクリプトのオプション

UDM ライブラリの configure 実行時のオプションには以下を設定します。

オプション	説明	必須
--prefix=PREFIX	パッケージのインストール先を指定します。 デフォルトでは"/usr/local/udmlib"にインストールします。	×
--with-comp=(INTEL FJ GNU)	コンパイラタイプを指定します。	×
--with-mpich=DIR	MPI ライブラリとして MPICH を使用する場合に MPICH のインストールパスを指定します。	△
--with-mpi=DIR	MPI ライブラリとして OpenMPI を使用する場合に OpenMPI のインストールパスを指定します。	△
--with-cgns=DIR	CGNS ライブラリのインストールパスを指定します。	○
--with-tp=DIR	TextParser ライブラリのインストールパスを指定します。	○
--with-zoltan=DIR	Zoltan ライブラリのインストールパスを指定します。	○
--disable-mpi	MPI ライブラリを使用しない場合に指定します。	×
--enable-real8	UdmReal_t データ型を double (8byte)に設定します。 デフォルト : UdmReal_t データ型=float (4byte)	×
--enable-int8	UdmInteger_t データ型を long long (8byte)に設定します。 デフォルト : UdmInteger_t データ型=int (4byte)	×
--enable-size8	UdmSize_t のデータ型を size_t 型とします。 デフォルト : UdmSize_t データ型=unsigned int	×
CXX	C++ コンパイラのコマンドパスを指定します。	×
CXXFLAGS	C++ コンパイラへ渡すコンパイルオプションを指定します。	×
LDFLAGS	リンク時にリンクに渡すリンク時オプションを指定します。	×
LIBS	利用したいライブラリをリンクに渡すリンク時オプションを指定します。	×

(1) --prefix=PREFIX

prefix は、パッケージをどこにインストールするかを指定します。prefix で設定した場所が"--prefix=/home/udm-user/udmlib-X.X.X"の時、

ライブラリ : /home/udm-user/udmlib-X.X.X/lib

ヘッダファイル : /home/udm-user/udmlib-X.X.X/include

にインストールされます。

`prefix` オプションが省略された場合は、デフォルト値として `/usr/local/udmlib` にインストールされます。

(2) `--with-comp=(INTEL|FJ)`

使用するコンパイラのベンダーを指定します。

Intel コンパイラの場合 `INTEL` を、富士通コンパイラるとき `FJ` を指定します。該当しない場合は指定する必要はありません。

(3) `--with-mpich=DIR`

MPI ライブラリとして `MPICH` を使用する場合に `MPICH` のインストールパスを指定します。

`--with-ompi` オプションと `--with-mpich` オプションは同時に指定しないでください。

`--with-ompi` オプションが優先されます。

MPI ライブラリは必須ですので、`--with-ompi` 又は `--with-mpich` が必要となります。但しコンパイラに `mpicxx` 等を指定した場合は必要ありません。

(注) `Zoltan` ライブラリのビルド時と同じ `MPI` ライブラリを使用してください。

(4) `--with-ompi=DIR`

MPI ライブラリとして `OpenMPI` を使用する場合に `OpenMPI` のインストールパスを指定します。

`--with-ompi` オプションと `--with-mpich` オプションは同時に指定しないでください。

`--with-ompi` オプションが優先されます。

MPI ライブラリは必須ですので、`--with-ompi` 又は `--with-mpich` が必要となります。但しコンパイラに `mpicxx` 等を指定した場合は必要ありません。

(注) `Zoltan` ライブラリのビルド時と同じ `MPI` ライブラリを使用してください。

(5) `--with-cgns=DIR`

`CGNS` ライブラリのインストールパスを指定します。

このオプションは必須となります。

(6) `--with-tp=DIR`

`TextParser` ライブラリのインストールパスを指定します。

このオプションは必須となります。

(7) `--with-zoltan=DIR`

Zoltan ライブラリのインストールパスを指定します。

このオプションは必須となります。

(8) `--disable-mpi`

MPI ライブラリをリンクしないで UDM ライブラリをビルドします。

フロントエンドでステージングツールを使用する場合のビルドです。

`examples` プログラムのビルド、Zoltan のライブラリリンクは行いません。

(9) `--enable-real8`

UdmReal_t データ型はデフォルトでは `float` 型ですが、それを `double` に設定します。

(10) `--enable-int8`

UdmInteger_t データ型はデフォルトでは `int` 型ですが、それを `long long` に設定します。

(11) `--enable-size64`

UdmSuze_t データ型はデフォルトでは `unsigned int` 型ですが、それを `size_t` (64bit 環境の場合、8byte 整数) に設定します。

(12) コンパイラ等のオプション

コンパイラ、リンカやそれらのオプションは、`configure` スクリプトで半自動的に探索します。ただし、標準ではないコマンドやオプション、ライブラリ、ヘッダファイルの場所は探索出来ないことがあります。また、標準でインストールされたものでないコマンドやライブラリを指定して利用したい場合があります。そのような場合、これらの指定を `configure` スクリプトのオプションとして指定することができます。

(12-1) CXX

C++ コンパイラのコマンドパスです。

(12-2) CXXFLAGS

C++ コンパイラへ渡すコンパイルオプションです。

(12-3) LDFLAGS

リンク時にリンカに渡すリンク時オプションです。例えば、使用するライブラリが標準でないの場所<libdir> にある場合、`-L<libdir>` としてその場所を指定します

(12-4) LIBS

利用したいライブラリをリンカに渡すリンク時オプションです。例えば、ライブラリ<library> を利用する場合、`-l<library>` として指定します。

2.5 configure 実行時オプションの例

以下 configure 実行時のオプションの記述例について説明します。

(1) Linux / MacOS X : g++

Linux / MacOS X で、ビルド環境が以下の場合について説明します。

UDM ライブラリの prefix : /usr/local/udmlib

MPI ライブラリ : OpenMPI, /usr/local/openmpi

CGNS ライブラリ : /usr/local/cgnslib

Zoltan ライブラリ : /usr/local/zoltan

TextParser ライブラリ : /usr/local/textparser

C++ コンパイラ : g++

以下のように configure コマンドを実行します。

```
$ ./configure --prefix=/usr/local/udmlib ¥
--with-cgns=/usr/local/cgnslib ¥
--with-tp=/usr/local/textparser ¥
--with-zoltan=/usr/local/zoltan ¥
--with-mpi=/usr/local/openmpi
```

(2) Linux / MacOS X : INTEL

Linux / MacOS X で、ビルド環境が以下の場合について説明します。

UDM ライブラリの prefix : /usr/local/udmlib_intel

MPI ライブラリ : OpenMPI, /usr/local/openmpi

CGNS ライブラリ : /usr/local/cgnslib

Zoltan ライブラリ : /usr/local/zoltan

TextParser ライブラリ : /usr/local/textparser

C++ コンパイラ : INTEL/icpc

C++ コンパイルオプション : -O3

以下のように configure コマンドを実行します。

```
$ ./configure --prefix=/usr/local/udmlib_intel ¥
--with-cgns=/usr/local/cgnslib ¥
--with-tp=/usr/local/textparser ¥
--with-zoltan=/usr/local/zoltan ¥
```

```
--with-mpi=/usr/local/openmpi ¥  
--with-comp=INTEL ¥  
CXX=/opt/intel/icpc ¥  
CXXFLAGS=-O3
```

(注) INTEL コンパイラを使用する場合、"--with-comp=INTEL"を指定する必要があります。

(3) Linux / MacOS X : mpicxx

Linux / MacOS X で、ビルド環境が以下の場合について説明します。

UDM ライブラリの prefix : /usr/local/udmlib_mpicxx
CGNS ライブラリ : /usr/local/cgnslib
Zoltan ライブラリ : /usr/local/zoltan
TextParser ライブラリ : /usr/local/textparser
C++ コンパイラ : mpicxx

以下のように configure コマンドを実行します。

```
$ ./configure --prefix=/usr/local/udmlib_mpicxx ¥  
--with-cgns=/usr/local/cgnslib ¥  
--with-tp=/usr/local/textparser ¥  
--with-zoltan=/usr/local/zoltan ¥  
--with-comp=INTEL ¥  
CXX=/usr/local/openmpi/bin/mpicxx
```

(注) MPI ライブラリを指定する"--with-mpich"、"--with-mpi"は指定する必要はありません。

(4) 京コンピュータの場合

京コンピュータで、ビルド環境が以下の場合について説明します。

UDM ライブラリの prefix : /home/user/udmlib
CGNS ライブラリ : /home/user/cgnslib
Zoltan ライブラリ : /home/user/zoltan
TextParser ライブラリ : /home/user/textparser
C++ コンパイラ : mpiFCCpx
C++ コンパイルオプション : -Kfast

以下のように `configure` コマンドを実行します.

```
$ ./configure --prefix=/home/user/udmlib    ¥
--with-cgns=/home/user/cgnslib    ¥
--with-tp=/home/user/textparser    ¥
--with-zoltan=/home/user/zoltan    ¥
--with-comp=FJ    ¥
CXX=mpiFCCpx    ¥
CXXFLAGS=-Kfast
```

(注) 富士通コンパイラを使用する場合、"`--with-comp=FJ`"を指定する必要があります。

(5) フロントエンドの場合

フロントエンドで、MPI ライブラリの無いビルド環境が以下の場合について説明します。

UDM ライブラリの `prefix` : `/home/user/udmlib_front`

CGNS ライブラリ : `/home/user/cgnslib`

TextParser ライブラリ : `/home/user/textparser`

C++ コンパイラ : `g++`

以下のように `configure` コマンドを実行します.

```
$ ./configure --prefix=/home/user/udmlib_front    ¥
--with-cgns=/home/user/cgnslib    ¥
--with-tp=/home/user/textparser    ¥
--disable-mpi
```

(注) MPI ライブラリを使用しない場合、"`--disable-mpi`"を指定する必要があります。

2.6 udm-config コマンド

UDM ライブラリをインストールすると、`$prefix/bin/udm-config` コマンド（シェルスクリプト）が生成されます。

このコマンドを利用することで、ユーザーが作成したプログラムをコンパイル、リンクする際に、UDM ライブラリを参照するために必要なコンパイルオプション、リンク時オプションを取得することができます。

`udm-config` コマンドは、次に示すオプションを指定して実行します。

オプション	説明
<code>--cxx</code>	UDM ライブラリの構築時に使用した C++ コンパイラを取得します。
<code>--cflags</code>	C++ コンパイラオプションを取得します。
<code>--libs</code>	UDM ライブラリのリンクに必要なリンク時オプションを取得します。

ただし、`udm-config` コマンドで取得できるオプションは、UDM ライブラリを利用する上で最低限必要なオプションのみとなります。

最適化オプション等は必要に応じて指定してください。

また、具体的な `udm-config` コマンドの使用方法は、「UDM ライブラリの利用方法」を参照してください。

2.7 提供環境の作成

提供環境の作成を行うには、`configure` スクリプト実行後に、以下のコマンドを実行します。

```
$ ./make dist
```

上記コマンドを実行すると、提供環境が"UDMLib-X.X.X.tar.gz"という圧縮ファイルに保存されます。(X.X.X にはバージョンが入ります)

2.8 京コンピュータでのビルド

京コンピュータでの Fujitsu コンパイラによるビルド方法について説明します。

ライブラリ名	機能説明
CGNS ライブラリ	数値流体力学(CFD)用の階層型データベース
Zoltan ライブラリ	ロードバランス、分割、及びカラーリング API
TextParser	TextPaser 形式の ASCII ファイルの入出力
UDM ライブラリ	非構造格子モデルの MxN 分割ライブラリ

インストール先`--prefix`についてはユーザ環境に合わせて適時変更してください。

2.8.1 CGNS ライブラリ

数値流体力学(CFD)用の階層型データベース管理および I/O サブルーチンのライブラリです。

```
$ tar xzvf cgnslib_3.2.1.tar.gz
$ cd cgnslib_3.2.1/src
$.CC=mpifccpx ¥
CFLAGS="-Kfast,ocl,preex,simd=2,uxsimd,array_private,parallel,openmp,optmsg=2 -V -Nsrc -x0" ¥
./configure --prefix=/volumeXX/k9999/udm_project/local/cgnslib_3.2.1 ¥
--enable-64bit ¥
--host=sparc64-unknown-linux-gnu
$ make
$ make install
```

2.8.2 Zoltan ライブラリ

ロードバランス、分割、及びカラーリング機能を提供するライブラリです。

```
$ tar xzvf zoltan_distrib_v3.8.tar.gz
$ cd Zoltan_v3.8
$ rm configure.ac
$. ./configure --prefix=/volumeXX/k9999/udm_project/local/zoltan_v3.8 ¥
--with-mpi=/opt/FJSVtclang/GM-1.2.0-15 ¥
--enable-mpi ¥
```

```

CXX=mpiFCCpx ¥
CXXFLAGS="-Kfast,ocl,preex,simd=2,uxsimd,array_private,parallel,openmp,optmsg=2
-V -Nsrc -x0" ¥
CC=mpifccpx ¥
CFLAGS="-Kfast,ocl,preex,simd=2,uxsimd,array_private,parallel,openmp,optmsg=2 -V
-Nsrc -x0" ¥
--host=sparc64-unknown-linux-gnu ¥
$ make
$ make install

```

2.8.3 TextParser

TextPaser 形式の ASCII ファイルの入出力を行うライブラリです。

```

$ unzip TextParser-master.zip
$ cd TextParser-master
$ ./configure --prefix=/volumeXX/k9999/udm_project/local/TextParser ¥
CXX=mpiFCCpx ¥
CXXFLAGS="-Kfast,ocl,preex,simd=2,uxsimd,array_private,parallel,openmp,optmsg=2
-V -Nsrc -x0" ¥
--host=sparc64-unknown-linux-gnu
$ make
$ make install

```

2.8.4 UDM ライブラリ

TextPaser 形式の ASCII ファイルの入出力を行うライブラリです。

```

$ tar xzvf UDMlib-X.X.X.tar.gz
$ cd UDMlib-X.X.X
$ ./configure --prefix=/volumeXX/k9999/udm_project/local/udmlib_X.X.X ¥
CXX=mpiFCCpx ¥
CXXFLAGS="-Kfast,ocl,preex,simd=2,uxsimd,array_private,parallel,openmp,optmsg=2
-V -Nsrc -x0" ¥

```

```
CC=mpiFCCpx      ¥
CFLAGS="-Kfast,ocl,preex,simd=2,uxsimd,array_private,parallel,openmp,optmsg=2  -V
-Nsrc -x0" ¥
--host=sparc64-unknown-linux-gnu ¥
--with-comp=FJ ¥
--with-cgns=/volumeXX/k9999/udm_project/local/cgnslib_3.2.1 ¥
--with-tp=/volumeXX/k9999/udm_project/local/TextParser ¥
--with-zoltan=/volumeXX/k9999/udm_project/local/zoltan_v3.8
$ make
$ make install
```

3. UDM ライブラリの利用方法

UDM ライブラリは、C++ プログラム内で利用できます。以下に、ユーザーが作成する UDM ライブラリを利用するプログラムのビルド方法を示します。

以下の `configure` スクリプトで "`--prefix=/usr/local/udmlib`" を指定して UDM ライブラリをビルド、インストールしているものとして示します。

```
$ ./configure --prefix=/usr/local/udmlib ¥
--with-cgns=/usr/local/cgnslib ¥
--with-tp=/usr/local/textparser ¥
--with-zoltan=/usr/local/zoltan ¥
--with-mpi=/usr/local/openmpi ¥
--with-comp=INTEL ¥
CXX=/opt/intel/icpc ¥
CXXFLAGS=-O3
```

3.1 C/C++によるコンパイル

UDM ライブラリを利用している C/C++ のプログラム `main.cpp` を `icc/icpc` でコンパイルする場合は、次のようにコンパイル、リンクします。

```
$ /opt/intel/bin/icpc -o program main.cpp ¥
    -I/usr/local/udmlib/bin/udm-config --cflags` ¥
    -L/usr/local/udmlib/bin/udm-config --libs`
```

これは、以下のコンパイルオプションを設定したものと同等となります。

```
$ /opt/intel/bin/icpc -o program main.cpp ¥
    -I/usr/local/udmlib/include ¥
    -D_HAS_OMPI_ ¥ (注 1)
    -I/usr/local/openmpi/include ¥
    -I/usr/local/cgnslib/include ¥
    -I/usr/local/textparser/include ¥
    -I/usr/local/zoltan/include ¥
```

<code>-L/usr/local/udmlib/lib</code>	¥
<code>-L/usr/local/zoltan/lib</code>	¥
<code>-L/usr/local/cgnslib/lib</code>	¥
<code>-L/usr/local/openmpi/lib</code>	¥
<code>-ludm -lstdc++ -lzoltan -lcgns</code>	¥
<code>-L/usr/local/textparser/lib -lTP</code>	¥
<code>-lmpi -lmpi_cxx</code>	(注 2)

(注 1) MPICH ライブラリを指定した場合は"`-D_HAS_MPICH_`"となります。

(注 2) `configure` で指定した `CXXFLAGS`、`LDFLAGS`、`LIBS` は含まれません

3.2 利用例（UDMLib-X.X.X/examples）

UDM ライブラリの利用例として、パッケージの **examples** に利用プログラムがあります。UDM ライブラリのビルドにより実行モジュールが作成されます。但し、インストールは行われません。

利用例のプログラムのディレクトリ構成は以下となっています。

ディレクトリ	利用例
cgns	CGNS ファイルの入出力
partition	Zoltan による分割
timeslice	時系列 CGNS ファイルの出力
dfi	DFI ファイルの入出力

以下、利用例のプログラムについて説明します。

3.2.1 CGNS ファイルの入出力（cgns）

CGNS ファイルの入出力の確認の為の以下のソースファイル、実行モジュールについて説明します。

実行モジュール	ソースファイル	説明
create_cgns	create_cgns.cpp	CGNS ファイルを生成します。
read_cgns	read_cgns.cpp	CGNS ファイルを読み込み、出力します。
vtk2cgns	vtk2cgns.cpp	VTK ファイル(torus.tetra.vtk)から座標、形状データを読み込み、分割、CGNS 出力を行います。
create_multi	create_multi.cpp	接続情報を持つ CGNS ファイルを生成し、分割、CGNS 出力を行います。
user_defined	user_defined.cpp	CGNS ユーザ定義情報の書き込み、読み込みを行います。
cgns_run.sh	/	上記の実行モジュールを実行します。

(1) create_cgns

モデルの節点（ノード）の座標値、要素（セル）の形状、構成節点（ノード）を設定します。設定モデルは CGNS ファイルとして出力します。

また、index.dfi, proc.dfi ファイルを出力します。

以下のオプションが指定可能です。

```
$ ./create_cgns --help
OPTIONS:
    -n, --name=[CGNS_FILENAME]          出力 CGNS ファイル名
```

	デフォルト = cgns_model.cgns
-s, --size [X_SIZE],[Y_SIZE],[Z_SIZE]	X,Y,Z サイズ
	デフォルト = 16,16,16
-c, --coords [X_COORD],[Y_COORD],[Z_COORD]	X,Y,Z 座標幅
	デフォルト = 1.0,1.0,1.0
-h --help	ヘルプ出力

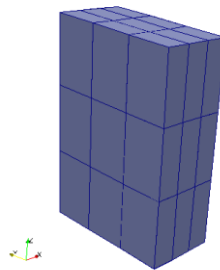
(例)

```
create_cgns      --name=output/cgns_hexa.cgns      --size=32,32,32
--coords=1.0,1.5,2.0
```

以下、create_cgns の実行例を示します。

```
$ ./create_cgns --name=model_4x4x4/cgns_hexa.cgns ¥
--size=4,4,4 ¥
--coords=1.0,2.0,3.0
$ ls model_4x4x4/
cgns_hexa_0000000000_id000000.cgns    index.dfi    cgns_hexa_id000000.cgns
proc.dfi
```

(cgns_hexa_id000000.cgns の ParaView 表示)



(2) read_cgns

CGNS ファイルを読み込み、モデル情報（座標、構成）をターミナル出力します。
また、読み込みモデルを"output"ディレクトリに CGNS ファイルを出力します。

以下、read_cgns の実行例を示します。

```
$ ./read_cgns model_4x4x4/cgns_hexa.cgns
Base name = UdmBase
Base id = 1
```



```

要素（セル）次元数 = 3
ノード次元数 = 3
シミュレーションタイプ = TimeAccurate
UdmInfo:UDMLibraryVersion = 0.1.0
UdmInfo:ProcessSize = 1
UdmInfo:Rankno = 0
Zone name = UdmZone#1
Zone id = 1
ノード（頂点）サイズ = 64
要素（セル）サイズ = 27
ゾーンタイプ = 非構造格子(Unstructured)
GridCoordinates
GridCoordinates[1] : x,y,z = 0.000000e+00,0.000000e+00,0.000000e+00 <- 1[0]
GridCoordinates[2] : x,y,z = 1.000000e+00,0.000000e+00,0.000000e+00 <- 2[0]
（以下省略）

$ ls output/
cgns_hexa_id000000_0000000000_id000000.cgns  cgns_hexa_id000000_id000000.cgns
index.dfi  proc.dfi

```

(3) vtk2cgns

VTK ファイル(**torus.tetra.vtk**)から節点（ノード）の座標値、要素（セル）の形状、構成節点（ノード）を読み込み、Zoltan 分割します。

分割結果は分割 CGNS ファイル、**index.dfi**, **proc.dfi**, **udmlib.tp** の出力を行います。

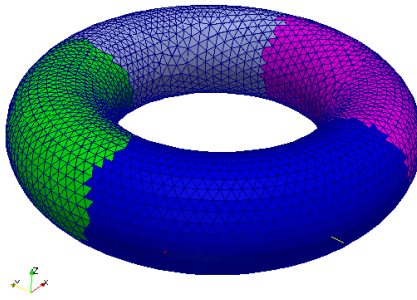
以下、**vtk2cgns** の実行例を示します。

```

$ mpiexec -np 4 ./vtk2cgns ../../data/torus.tetra.vtk
$ ls torus.tetra
index.dfi  proc.dfi  udmlib.tp
torus.tetra_0000000000_id000000.cgns  torus.tetra_0000000000_id000001.cgns
torus.tetra_0000000000_id000002.cgns  torus.tetra_0000000000_id000003.cgns
torus.tetra_id000000.cgns  torus.tetra_id000001.cgns
torus.tetra_id000002.cgns  torus.tetra_id000003.cgns

```

(torus.tetra_id000000[0-3].cgns の ParaView 表示)



4 分割結果

(4) create_multi

接続情報を持つ CGNS ファイルを生成し、分割、CGNS 出力を行います。

以下のオプションが指定可能です。

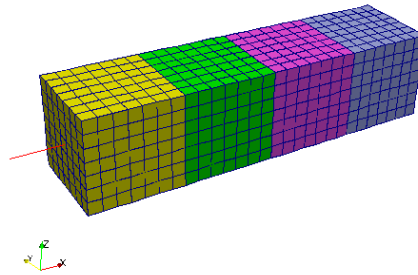
```
$ ./create_multi --help
OPTIONS:
  -n, --name=[CGNS_FILENAME]          出力 CGNS ファイル名
                                       デフォルト = cgns_model.cgns
  -s, --size [X_SIZE],[Y_SIZE],[Z_SIZE] X,Y,Z サイズ
                                       デフォルト = 16,16,16
  -c, --coords [X_COORD],[Y_COORD],[Z_COORD] X,Y,Z 座標幅
                                       デフォルト = 1.0,1.0,1.0
  -h --help                            ヘルプ出力
```

以下、create_multi の実行例を示します。

```
$ /usr/local/openmpi-1.6.5_gcc/bin/mpirun -np 4 ./create_multi \
--name=cgns_multi/cgns_hexa.cgns --size=8,8,8
```

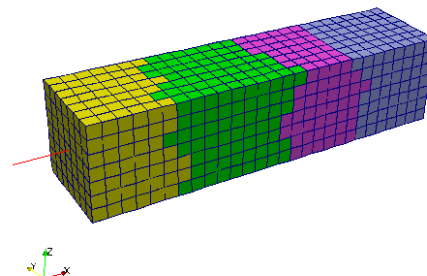
- a) 8x8x8 のモデルが 4 つ X 方向に接続された CGNS を作成します。
- b) cgns_multi ディレクトリに出力します。
- c) Zoltan 分割した結果を cgns_multi/partition ディレクトリに出力します。

(cgns_multi/cgns_hexa_id000000[0-3].cgns の ParaView 表示 : 分割前)



(4 モデル生成)

(cgns_multi/partition/cgns_hexa_id00000[0-3].cgns の ParaView 表示 : 分割後)



(4 モデル分割)

Zoltan 分割により接続面が変更される。

(5) user_defined

index.dfi ファイルから定義されている CGNS ファイルを読み込み、ユーザ定義情報を設定します。

ユーザ定義情報を設定した CGNS ファイルを"output"ディレクトリに出力します。

再度、CGNS ファイルを読み込み、ユーザ定義情報をターミナル出力します。

以下、user_defined の実行例を示します。

```

$ ./user_defined model_4x4x4/index.dfi
Start : user_defined
Start :: loadModel!
End :: loadModel!
/**** write user defined data ****/
UserMatrix
    size = 3 x 10
    value = 0.000000 0.500000 1.000000

```

```

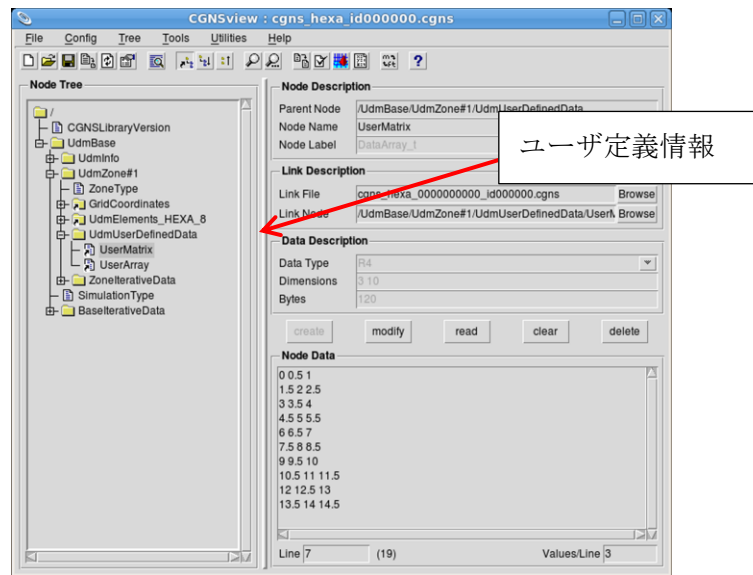
1.500000 2.000000 2.500000
3.000000 3.500000 4.000000
4.500000 5.000000 5.500000
6.000000 6.500000 7.000000
7.500000 8.000000 8.500000
9.000000 9.500000 10.000000
10.500000 11.000000 11.500000
12.000000 12.500000 13.000000
13.500000 14.000000 14.500000

UserArray
  size = 5
  value = 1 3 5 7 9
Start :: writeModel!
End :: writeModel!
/**** read user defined data ****/
cgns file = model_4x4x4/output/./cgns_hexa_id000000.cgns
UserMatrix
  size = 3 x 10
  value = 0.000000 0.500000 1.000000
1.500000 2.000000 2.500000
3.000000 3.500000 4.000000
4.500000 5.000000 5.500000
6.000000 6.500000 7.000000
7.500000 8.000000 8.500000
9.000000 9.500000 10.000000
10.500000 11.000000 11.500000
12.000000 12.500000 13.000000
13.500000 14.000000 14.500000

UserArray
  size = 5
  value = 1 3 5 7 9
End : user_defined

```

(model_4x4x4/output/cgns_hexa_id000000.cgns の cgnsview 表示)



(6) cgns_run.sh

CGNS ファイルの入出力例の実行モジュールを順次実行します。

以下、cgns_run.sh の実行例を示します。

```
$ export LD_LIBRARY_PATH=/usr/local/openmpi/lib
$ export PATH=$PATH:/usr/local/openmpi/bin
$ ./cgns_run.sh
$ tree                # /cgns_run.sh による出力ディレクトリ
|-- cgns_multi
|   |-- partitionc
|-- model_4x4x4
|   |-- output
|-- output
`-- torus.tetra
```

(注) MPI ライブラリの LD_LIBRARY_PATH, PATH を設定する必要があります。

3.2.2 Zoltan による分割 (partition)

Zoltan による CGNS ファイルの分割の確認の為に以下のソースファイル、実行モジュールについて説明します。

実行モジュール	ソースファイル	説明
partition	partition.cpp	CGNS ファイルの分割を行います。
partition_weight	partition_weight.cpp	要素 (セル) に重みを持った分割を行います。
partition_solutions	partition_solutions.cpp	物理量を持つモデルの分割を行います。
partition_run.sh	/	上記の実行モジュールを実行します。

(1) partition

index.dfi ファイルから CGNS ファイルを読み込み、Zoltan 分割を行います。

分割結果は出力パスに出力を行います。

又、Zoltan 分割パラメータの設定、ランク番号による読込パスの設定も可能です。

以下のオプションが指定可能です。

```
$ ./partition --help
OPTIONS:
  --output=[OUTPUT_PATH]  出力パス (デフォルト="./output")
  --enable_hyper           Zoltan::PACKAGE=HYPERGRAPH (デフォルト)
  --enable_graph           Zoltan::PACKAGE=GRAPH
  --enable_partition       Zoltan::LB_APPROACH=PARTITION (デフォルト)
  --enable_repartition     Zoltan::LB_APPROACH=REPARTITION
  --step=[STEP_NO]        ロードを行う時系列ステップ番号を指定します。
  --with-mpirank-path=[MPIRANK_PATH]
                           MPI ランク番号のディレクトリ毎に index.dfi を配置します。
                           INDEX_DFI は無視されます。
  -h --help               ヘルプ出力
```

a) --step

CGNS ファイルから読み込む時系列ステップ番号を指定します。

b) --with-mpirank-path=[MPIRANK_PATH]

MPI ランク番号のディレクトリ構成から index.dfi ファイルを読み込みます。

[MPIRANK_PATH]

| -- 000000

 |-- index.dfi

| -- 000001

```

`-- index.dfi
`-- 000003
`-- index.dfi

```

udm-frm（ファイルリンクマッピングツール）にて配置されたディレクトリ構成から読み込みを行います。

以下、partition の実行例を示します。

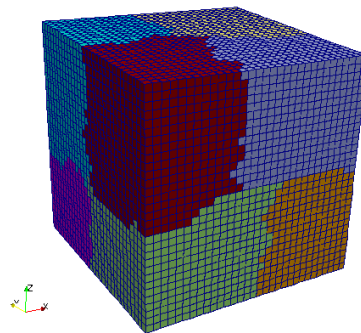
```

$ mpiexec -np 8 ./partition input_32/index.dfi
$ ls ./output
cgns_hexa_0000000000_id000000.cgns  cgns_hexa_id000002.cgns
cgns_hexa_0000000000_id000001.cgns  cgns_hexa_id000003.cgns
cgns_hexa_0000000000_id000002.cgns  cgns_hexa_id000004.cgns
cgns_hexa_0000000000_id000003.cgns  cgns_hexa_id000005.cgns
cgns_hexa_0000000000_id000004.cgns  cgns_hexa_id000006.cgns
cgns_hexa_0000000000_id000005.cgns  cgns_hexa_id000007.cgns
cgns_hexa_0000000000_id000006.cgns  index.dfi
cgns_hexa_0000000000_id000007.cgns  proc.dfi
cgns_hexa_id000000.cgns              udmlib.tp
cgns_hexa_id000001.cgns

```

a) index.32/index.dfi の 32x32x32 の CGNS ファイルを 8 分割します。

(output/cgns_hexa_id00000[0-7].cgns の ParaView 表示)



(2) partition_weight

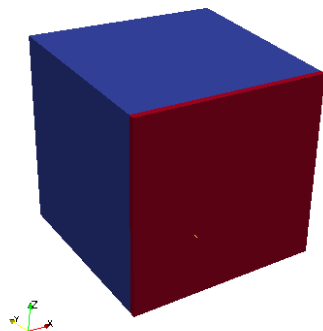
読み込みモデルの-J 面の要素（セル）に重みを設定して分割を行います。

重み設定の分割 CGNS ファイル出力（output_weight）と重み設定なしの分割出力（output_noneweight）を行います。

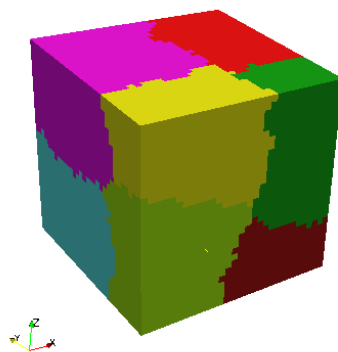
以下、partition_weight の実行例を示します。

```
$ mpiexec -np 8 partition_weight input_32/index.dfi
```

(output_weight/cgns_hexa_id000000[0-7].cgns の ParaView 表示)

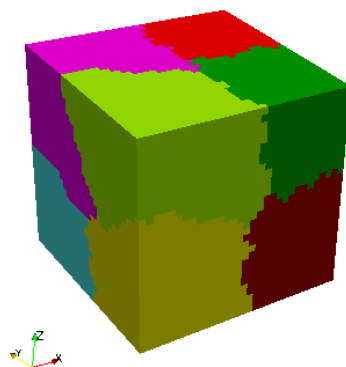


重み設定要素（セル）表示



重み設定 8 分割

(output_noneweight/cgns_hexa_id000000[0-7].cgns の ParaView 表示)



重み設定なし 8 分割

(3) partition_solutions

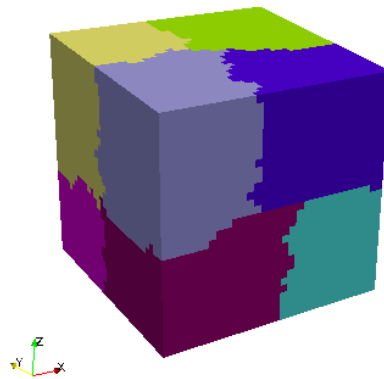
物理量を持つモデルを分割する場合、物理量も分割先に転送します。

分割結果の CGNS ファイルは output_solution に出力します。

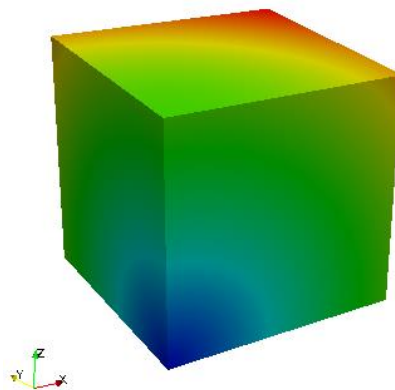
以下、partition_solution の実行例を示します。

```
$ mpiexec -np 8 partition_solution input_32/index.dfi
```

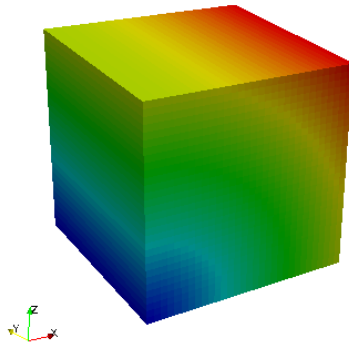
(output_solution/cgns_hexa_id00000[0-7].cgns の ParaView 表示)



8 分割表示



節点（ノード）設定物理量：Pressure 表示



要素（セル）設定物理量：Temperature 表示

(4) partition_run.sh

Zoltan による分割例の実行モジュールを順次実行します。

32x32x32 の 1 つのモデルを 1->4->8->16->8->4->1 分割を順次行います。

以下、partition_run.sh の実行例を示します。

```
$ export LD_LIBRARY_PATH=/usr/local/openmpi/lib
$ export PATH=$PATH:/usr/local/openmpi/bin
$ ./partition_run.sh
```

実行結果は以下のディレクトリに出力されます。

出力 順	出力ディレクトリ	実行モジュール 出力説明
1	input_32	cgns/create_cgns 32x32x32 モデルを生成します。
2	output_32-m1-p4	mpiexec -np 4 partition input_32/index.dfi 1 つの 32x32x32 モデルを 4 分割します。
3	output_32-m4-p8	mpiexec -np 8 partition output_32-m1-p4/index.dfi 4 つの分割モデルを 8 分割します。
4	output_32-m8-p16	mpiexec -np 16 partition output_32-m4-p8/index.dfi 8 つの分割モデルを 16 分割します。
5	output_32-m16-p8	mpiexec -np 8 partition output_32-m8-p16/index.dfi 16 つの分割モデルを 8 分割します。
6	output_32-m8-p4	mpiexec -np 4 partition output_32-m16-p8/index.dfi 8 つの分割モデルを 4 分割します。
7	output_32-m4-p1	mpiexec -np 1 partition output_32-m8-p4/index.dfi

出力 順	出力ディレクトリ	実行モジュール 出力説明
		4つの分割モデルを1分割します。
8	output_weight	mpexec -np 8 partition_weight input_32/index.dfi 重みを設定した 32x32x32 モデルを 8 分割します
	output_noneweight	重みの設定解除した 32x32x32 モデルを 8 分割します
9	output_solution	mpexec -np 8 partition_solution input_32/index.dfi 物理量の設定した 32x32x32 モデルを 8 分割します

3.2.3 時系列 CGNS ファイルの出力 (timeslice)

時系列ステップによる物理量の変化をステップ毎に CGNS ファイルの出力します。
以下のソースファイル、実行モジュールについて説明します。

実行モジュール	ソースファイル	説明
create_timeslice	create_timeslice.cpp	時系列出力用の index.dfi の作成を行います。
timeslice	timeslice.cpp	物理量の変化をステップ毎に CGNS ファイルを出力します。
timeslice_run.sh	/	上記の実行モジュールを実行します。

(1) create_timeslice

UDM ライブラリは CGNS ファイル構成、出力ディレクトリ構成にて様々な出力形態、構成をサポートしています。

create_timeslice プログラムはサポートする出力形態、構成の index.dfi、CGNS ファイルを作成します。

出力形態、構成の設定の為に以下のオプションが指定します。

```
$ ./create_timeslice --help
OPTIONS:
  --input=[INDEX_DFI]          入力 INDEX_DFI ファイル
  --output=[OUTPUT_PATH]      出力パス
  /**** CGNS:GridCoordinates/FlowSolution 出力 ****/
  --includegrid                CGNS:GridCoordinates と CGNS:FlowSolution を
                              1つのファイルに出力します。(デフォルト)
  --excludegrid                CGNS:GridCoordinates と CGNS:FlowSolution を
                              別ファイルに出力します。
```

```

/**** CGNS:FlowSolution 時系列出力 ****/
--appendstep          CGNS:FlowSolution を時系列毎に 1 つのファイルに
                        出力します。
--eachstep            CGNS:FlowSolution を時系列毎に別ファイルにします。
                        (デフォルト)

/**** CGNS:GridCoordinates 時系列出力 ****/
--gridconstant        CGNS:GridCoordinates は初期値のみ出力を行います。
                        (デフォルト)
--gridtimeslice       CGNS:GridCoordinates は時系列毎に出力を行います。

/**** output directory options ****/
--with_directorypath=[DIR]      フィールド出力ディレクトリ
--with_timeslice_directory      時系列ディレクトリ作成

/*****
-h --help                      ヘルプ出力

```

(2) timeslice

読み込みモデルを MPI プロセス数で分割を行い、分割モデルに対して物理量を時系列毎に設定、10 ステップ毎に CGNS ファイルを出力します。

出力構成は、index.dfi の設定に従います。

以下のオプションが指定可能です。

```

$ ./timeslice --help
usage: timeslice [INDEX_DFI] OPTIONS.
OPTIONS:
  --output=[OUTPUT_PATH]  出力パス (デフォルト="output")
  --without_cell           要素 (セル) の CGNS:FlowSolution を出力しません。
                           (デフォルト=出力する)
  --without_node          節点 (ノード) の CGNS:FlowSolution を出力しません。
                           (デフォルト=出力する)
  -h --help               ヘルプ出力

```

以下、timeslice の実行例を示します。

```
$ ../cgns/create_cgns --name=input/cgns_hexa.cgns ¥
```

```

--size=8,8,8
$ ./create_timeslice --input=input/index.dfi ¥
--output=includegrid_appendstep_gridconstant ¥
--includegrid --appendstep --gridconstant
$ mpiexec -np 4 ./timeslice ./includegrid_appendstep_gridconstant/index.dfi
]$ tree ./includegrid_appendstep_gridconstant/output
./includegrid_appendstep_gridconstant/output
|-- cgns_hexa_id000000.cgns
|-- cgns_hexa_id000001.cgns
|-- cgns_hexa_id000002.cgns
|-- cgns_hexa_id000003.cgns
|-- index.dfi
|-- proc.dfi
`-- udmlib.tp

```

a) ../cgns/create_cgns ...

8x8x8 の CGNS ファイルを input ディレクトリに作成します

b) ./create_timeslice ...

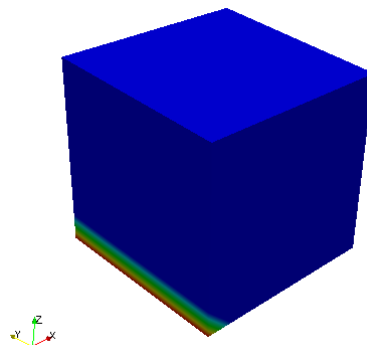
input ディレクトリに作成した index.dfi, CGNS ファイルを
{includegrid, appendstep, gridconstant}の構成に変更して、
"includegrid_appendstep_gridconstant"に出力します。

c) mpiexec -np 4 ./timeslice ./includegrid_appendstep_gridconstant...

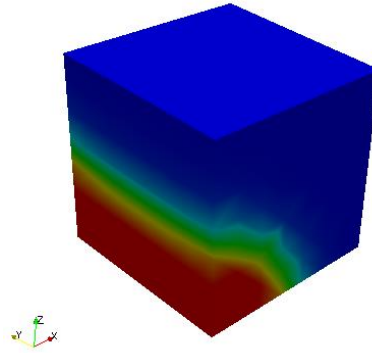
4 分割して、100 ステップ実行を行います。

10 ステップ毎に output ディレクトリに CGNS ファイルを出力
します。

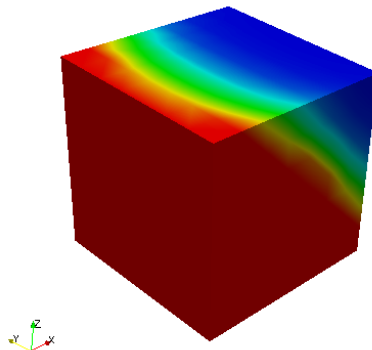
(output/cgns_hexa_id000000[0-4].cgns の ParaView 表示)



step=0 : Pressure



step=50 : Pressure



step=100 : Pressure

(3) timeslice_run.sh

`create_timeslice` プログラムにより出力構成を変更して、`timeslice` により分割を行い、分割モデルに対して物理量を時系列毎に設定、10 ステップ毎に CGNS ファイルを "output" ディレクトリに出力します。

以下、`timeslice_run.sh` による出力ディレクトリ、及び UDM ライブラリの CGNS ファイル構成を示します。

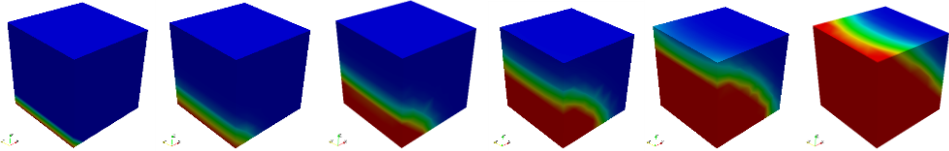
出力ディレクトリ	<code>create_timeslice</code> オプション
<code>includegrid_eachstep_gridconstant</code>	<code>--includegrid</code> <code>--eachstep</code> <code>--gridconstant</code>
CGNS ファイル構成	
(ファイル出力)	
<code>includegrid_eachstep_gridconstant</code>	
-- <code>cgns_hexa_0000000000_id000000.cgns</code>	
-- <code>cgns_hexa_id000000.cgns</code>	

<pre> -- index.dfi -- output -- cgns_hexa_0000000000_id000000.cgns -- cgns_hexa_0000000000_id000001.cgns -- cgns_hexa_0000000000_id000002.cgns -- cgns_hexa_0000000000_id000003.cgns -- cgns_hexa_0000000010_id000000.cgns -- cgns_hexa_0000000010_id000001.cgns -- cgns_hexa_0000000010_id000002.cgns -- cgns_hexa_0000000010_id000003.cgns ... -- cgns_hexa_id000000.cgns -- cgns_hexa_id000001.cgns -- cgns_hexa_id000002.cgns -- cgns_hexa_id000003.cgns -- index.dfi -- proc.dfi -- udmlib.tp -- proc.dfi </pre>
<p>CGNS:GridCoordinates と CGNS:FlowSolution を 1 つのファイルに出力します。</p> <p>時系列毎にステップ番号の付加した CGNS ファイルを出力します。</p> <p>--gridconstant は無視されます。時系列毎に出力する為にそのステップの GridCoordinates 値を出力します。</p> <p>(注) ParaView のアニメーション表示では、グリッド表示と時系列が一致しません。</p>

出力ディレクトリ	create_timeslice オプション
includegrid_eachstep_gridtimeslice	<pre> --includegrid --eachstep --gridtimeslice </pre>
CGNS ファイル構成	
<p>(ファイル出力)</p> <pre> includegrid_eachstep_gridtimeslice -- cgns_hexa_0000000000_id000000.cgns -- cgns_hexa_id000000.cgns -- index.dfi </pre>	

<pre> -- output -- cgns_hexa_0000000000_id000000.cgns -- cgns_hexa_0000000000_id000001.cgns -- cgns_hexa_0000000000_id000002.cgns -- cgns_hexa_0000000000_id000003.cgns -- cgns_hexa_0000000010_id000000.cgns -- cgns_hexa_0000000010_id000001.cgns -- cgns_hexa_0000000010_id000002.cgns -- cgns_hexa_0000000010_id000003.cgns ... -- cgns_hexa_id000000.cgns -- cgns_hexa_id000001.cgns -- cgns_hexa_id000002.cgns -- cgns_hexa_id000003.cgns -- index.dfi -- proc.dfi `-- udmlib.tp `-- proc.dfi </pre>	
<p>CGNS:GridCoordinates と CGNS:FlowSolution を 1 つのファイルに出力します。</p> <p>時系列毎にステップ番号の付加した CGNS ファイルを出力します。</p> <p>時系列毎にそのステップの GridCoordinates 値を出力します。</p> <p>(注) ParaView のアニメーション表示では、グリッド表示と時系列が一致しません。</p>	

出力ディレクトリ	create_timeslice オプション
includegrid_appendstep_gridconstant	<pre> --includegrid --appendstep --gridconstant </pre>
CGNS ファイル構成	
<p>(ファイル出力)</p> <pre> includegrid_appendstep_gridconstant -- cgns_hexa_id000000.cgns -- index.dfi -- output -- cgns_hexa_id000000.cgns -- cgns_hexa_id000001.cgns </pre>	

<pre> -- cgns_hexa_id000002.cgns -- cgns_hexa_id000003.cgns -- index.dfi -- proc.dfi `-- udmlib.tp -- proc.dfi</pre>
<p>CGNS:GridCoordinates と CGNS:FlowSolution を 1 つのファイルに出力します。</p> <p>時系列データは 1 つの CGNS ファイルに追加します。</p> <p>CGNS:GridCoordinates は初期値のみ出力を行います。</p> <div style="text-align: center;">  <p>step0,20,40,60,80,100 の Pressure : ParaView 表示</p> </div>

出力ディレクトリ	create_timeslice オプション
includegrid_appendstep_gridtimeslice	<pre>--includegrid --appendstep --gridtimeslice</pre>
CGNS ファイル構成	
<p>(ファイル出力)</p> <pre>includegrid_appendstep_gridconstant -- cgns_hexa_id000000.cgns -- index.dfi -- output -- cgns_hexa_id000000.cgns -- cgns_hexa_id000001.cgns -- cgns_hexa_id000002.cgns -- cgns_hexa_id000003.cgns -- index.dfi -- proc.dfi `-- udmlib.tp -- proc.dfi</pre>	
<p>CGNS:GridCoordinates と CGNS:FlowSolution を 1 つのファイルに出力します。</p> <p>時系列データは 1 つの CGNS ファイルに追加します。</p>	

時系列毎にそのステップの GridCoordinates 値を出力します。

(注) ParaView のアニメーション表示では、グリッド表示と時系列が一致しません。

出力ディレクトリ	create_timeslice オプション
excludegrid_eachstep_gridconstant	--excludegrid --eachstep --gridconstant
CGNS ファイル構成	
(ファイル出力) excludegrid_eachstep_gridconstant -- cgns_hexa_grid_id000000.cgns -- cgns_hexa_id000000.cgns -- cgns_hexa_solution_0000000000_id000000.cgns -- index.dfi -- output -- cgns_hexa_grid_id000000.cgns -- cgns_hexa_grid_id000001.cgns -- cgns_hexa_grid_id000002.cgns -- cgns_hexa_grid_id000003.cgns -- cgns_hexa_id000000.cgns -- cgns_hexa_id000001.cgns -- cgns_hexa_id000002.cgns -- cgns_hexa_id000003.cgns -- cgns_hexa_solution_0000000000_id000000.cgns -- cgns_hexa_solution_0000000000_id000001.cgns -- cgns_hexa_solution_0000000000_id000002.cgns -- cgns_hexa_solution_0000000000_id000003.cgns -- cgns_hexa_solution_0000000010_id000000.cgns -- cgns_hexa_solution_0000000010_id000001.cgns -- cgns_hexa_solution_0000000010_id000002.cgns -- cgns_hexa_solution_0000000010_id000003.cgns ... -- index.dfi -- proc.dfi `-- udmlib.tp	

<code>-- proc.dfi</code>
CGNS:GridCoordinates と CGNS:FlowSolution を別のファイルに出力します。 時系列毎にステップ番号の付加した CGNS ファイルを出力します。 CGNS:GridCoordinates は初期値のみ出力を行います。

出力ディレクトリ	create_timeslice オプション
excludegrid_eachstep_gridtimeslice	<code>--excludegrid</code> <code>--eachstep</code> <code>--gridtimeslice</code>
CGNS ファイル構成	
(ファイル出力) excludegrid_eachstep_gridtimeslice <pre>-- cgns_hexa_grid_0000000000_id000000.cgns -- cgns_hexa_id000000.cgns -- cgns_hexa_solution_0000000000_id000000.cgns -- index.dfi -- output -- cgns_hexa_grid_0000000000_id000000.cgns -- cgns_hexa_grid_0000000000_id000001.cgns -- cgns_hexa_grid_0000000000_id000002.cgns -- cgns_hexa_grid_0000000000_id000003.cgns -- cgns_hexa_grid_0000000010_id000000.cgns -- cgns_hexa_grid_0000000010_id000001.cgns -- cgns_hexa_grid_0000000010_id000002.cgns -- cgns_hexa_grid_0000000010_id000003.cgns ... -- cgns_hexa_id000000.cgns -- cgns_hexa_id000001.cgns -- cgns_hexa_id000002.cgns -- cgns_hexa_id000003.cgns -- cgns_hexa_solution_0000000000_id000000.cgns -- cgns_hexa_solution_0000000000_id000001.cgns -- cgns_hexa_solution_0000000000_id000002.cgns -- cgns_hexa_solution_0000000000_id000003.cgns</pre>	

<pre> -- cgns_hexa_solution_0000000010_id000000.cgns -- cgns_hexa_solution_0000000010_id000001.cgns -- cgns_hexa_solution_0000000010_id000002.cgns -- cgns_hexa_solution_0000000010_id000003.cgns ... -- index.dfi -- proc.dfi -- udmlib.tp -- proc.dfi</pre>
<p>CGNS:GridCoordinates と CGNS:FlowSolution を別のファイルに出力します。</p> <p>時系列毎にステップ番号の付加した CGNS ファイルを出力します。</p> <p>時系列毎にそのステップの GridCoordinates 値を出力します。</p> <p>(注) ParaView のアニメーション表示では、グリッド表示と時系列が一致しません。</p>

出力ディレクトリ	create_timeslice オプション
excludegrid_appendstep_gridconstant	<pre>--excludegrid --appendstep --gridconstant</pre>
CGNS ファイル構成	
<p>(ファイル出力)</p> <pre>excludegrid_appendstep_gridconstant -- cgns_hexa_grid_id000000.cgns -- cgns_hexa_id000000.cgns -- cgns_hexa_solution_id000000.cgns -- index.dfi -- output -- cgns_hexa_grid_id000000.cgns -- cgns_hexa_grid_id000001.cgns -- cgns_hexa_grid_id000002.cgns -- cgns_hexa_grid_id000003.cgns -- cgns_hexa_id000000.cgns -- cgns_hexa_id000001.cgns -- cgns_hexa_id000002.cgns -- cgns_hexa_id000003.cgns -- cgns_hexa_solution_id000000.cgns</pre>	

<pre> -- cgns_hexa_solution_id000001.cgns -- cgns_hexa_solution_id000002.cgns -- cgns_hexa_solution_id000003.cgns -- index.dfi -- proc.dfi `-- udmlib.tp `-- proc.dfi</pre>
<p>CGNS:GridCoordinates と CGNS:FlowSolution を別のファイルに出力します。</p> <p>時系列データは1つの CGNS ファイルに追加します。</p> <p>CGNS:GridCoordinates は初期値のみ出力を行います。</p>

出力ディレクトリ	create_timeslice オプション
excludegrid_appendstep_gridtimeslice	<pre>--excludegrid --appendstep --gridtimeslice</pre>
CGNS ファイル構成	
<p>(ファイル出力)</p> <pre>excludegrid_appendstep_gridtimeslice -- cgns_hexa_grid_id000000.cgns -- cgns_hexa_id000000.cgns -- cgns_hexa_solution_id000000.cgns -- index.dfi -- output -- cgns_hexa_grid_id000000.cgns -- cgns_hexa_grid_id000001.cgns -- cgns_hexa_grid_id000002.cgns -- cgns_hexa_grid_id000003.cgns -- cgns_hexa_id000000.cgns -- cgns_hexa_id000001.cgns -- cgns_hexa_id000002.cgns -- cgns_hexa_id000003.cgns -- cgns_hexa_solution_id000000.cgns -- cgns_hexa_solution_id000001.cgns -- cgns_hexa_solution_id000002.cgns -- cgns_hexa_solution_id000003.cgns</pre>	

<pre> -- index.dfi -- proc.dfi `-- udmlib.tp -- proc.dfi</pre>
<p>CGNS:GridCoordinates と CGNS:FlowSolution を別のファイルに出力します。</p> <p>時系列データは1つの CGNS ファイルに追加します。</p> <p>時系列毎にそのステップの GridCoordinates 値を出力します。</p> <p>(注) ParaView のアニメーション表示では、グリッド表示と時系列が一致しません。</p>

出力ディレクトリ	create_timeslice オプション
field_directory	<pre>--excludegrid --eachstep --gridtimeslice --with_directorypath=field_data</pre>
CGNS ファイル構成	
<p>(ファイル出力)</p> <pre>field_directory -- field_data -- cgns_hexa_grid_0000000000_id000000.cgns -- cgns_hexa_id000000.cgns `-- cgns_hexa_solution_0000000000_id000000.cgns -- index.dfi -- output -- field_data -- cgns_hexa_grid_0000000000_id000000.cgns -- cgns_hexa_grid_0000000000_id000001.cgns -- cgns_hexa_grid_0000000000_id000002.cgns -- cgns_hexa_grid_0000000000_id000003.cgns -- cgns_hexa_grid_0000000010_id000000.cgns -- cgns_hexa_grid_0000000010_id000001.cgns -- cgns_hexa_grid_0000000010_id000002.cgns -- cgns_hexa_grid_0000000010_id000003.cgns ... -- cgns_hexa_id000000.cgns -- cgns_hexa_id000001.cgns</pre>	

<pre> -- cgns_hexa_id000002.cgns -- cgns_hexa_id000003.cgns -- cgns_hexa_solution_0000000000_id000000.cgns -- cgns_hexa_solution_0000000000_id000001.cgns -- cgns_hexa_solution_0000000000_id000002.cgns -- cgns_hexa_solution_0000000000_id000003.cgns -- cgns_hexa_solution_0000000010_id000000.cgns -- cgns_hexa_solution_0000000010_id000001.cgns -- cgns_hexa_solution_0000000010_id000002.cgns -- cgns_hexa_solution_0000000010_id000003.cgns ... -- index.dfi -- proc.dfi `-- udmlib.tp `-- proc.dfi</pre>	
<p>CGNS:GridCoordinates と CGNS:FlowSolution を別のファイルに出力します。</p> <p>時系列毎にステップ番号の付加した CGNS ファイルを出力します。</p> <p>時系列毎にそのステップの GridCoordinates 値を出力します。</p> <p>フィールドディレクトリ"field_data"配下に CGNS ファイルを出力します。</p> <p>(注) ParaView のアニメーション表示では、グリッド表示と時系列が一致しません。</p>	

出力ディレクトリ	create_timeslice オプション
timeslice_directory	<pre>--excludegrid --eachstep --gridtimeslice --with_timeslice_directory</pre>
CGNS ファイル構成	
<p>(ファイル出力)</p> <pre>timeslice_directory -- 0000000000 -- cgns_hexa_grid_0000000000_id000000.cgns `-- cgns_hexa_solution_0000000000_id000000.cgns -- cgns_hexa_id000000.cgns -- index.dfi -- output</pre>	

```

| |-- 0000000000
| | |-- cgns_hexa_grid_0000000000_id000000.cgns
| | |-- cgns_hexa_grid_0000000000_id000001.cgns
| | |-- cgns_hexa_grid_0000000000_id000002.cgns
| | |-- cgns_hexa_grid_0000000000_id000003.cgns
| | |-- cgns_hexa_solution_0000000000_id000000.cgns
| | |-- cgns_hexa_solution_0000000000_id000001.cgns
| | |-- cgns_hexa_solution_0000000000_id000002.cgns
| | |-- cgns_hexa_solution_0000000000_id000003.cgns
| |-- 0000000010
| | |-- cgns_hexa_grid_0000000010_id000000.cgns
| | |-- cgns_hexa_grid_0000000010_id000001.cgns
| | |-- cgns_hexa_grid_0000000010_id000002.cgns
| | |-- cgns_hexa_grid_0000000010_id000003.cgns
| | |-- cgns_hexa_solution_0000000010_id000000.cgns
| | |-- cgns_hexa_solution_0000000010_id000001.cgns
| | |-- cgns_hexa_solution_0000000010_id000002.cgns
| | |-- cgns_hexa_solution_0000000010_id000003.cgns
| |-- 0000000020
| |-- 0000000030
| |-- 0000000040
| |-- 0000000050
| |-- 0000000060
| |-- 0000000070
| |-- 0000000080
| |-- 0000000090
| |-- 0000000100
| |-- cgns_hexa_id000000.cgns
| |-- cgns_hexa_id000001.cgns
| |-- cgns_hexa_id000002.cgns
| |-- cgns_hexa_id000003.cgns
| |-- index.dfi
| |-- proc.dfi
| |-- udmlib.tp
|-- proc.dfi

```

CGNS:GridCoordinates と CGNS:FlowSolution を別のファイルに出力します。

時系列毎にステップ番号の付加した CGNS ファイルを出力します。

時系列毎にそのステップの GridCoordinates 値を出力します。

時系列の 10 桁のステップ番号ディレクトリ配下に CGNS ファイルを出力します。

(注) ParaView のアニメーション表示では、グリッド表示と時系列が一致しません。

出力ディレクトリ	create_timeslice オプション
field_timeslice	--excludegrid --eachstep --gridtimeslice --with_directorypath=field_data --with_timeslice_directory
CGNS ファイル構成	
(ファイル出力) field_timeslice -- field_data -- 0000000000 -- cgns_hexa_grid_0000000000_id000000.cgns `-- cgns_hexa_solution_0000000000_id000000.cgns `-- cgns_hexa_id000000.cgns -- index.dfi -- output -- field_data -- 0000000000 -- cgns_hexa_grid_0000000000_id000000.cgns -- cgns_hexa_grid_0000000000_id000001.cgns -- cgns_hexa_grid_0000000000_id000002.cgns -- cgns_hexa_grid_0000000000_id000003.cgns -- cgns_hexa_solution_0000000000_id000000.cgns -- cgns_hexa_solution_0000000000_id000001.cgns -- cgns_hexa_solution_0000000000_id000002.cgns `-- cgns_hexa_solution_0000000000_id000003.cgns -- 0000000010 -- cgns_hexa_grid_0000000010_id000000.cgns -- cgns_hexa_grid_0000000010_id000001.cgns -- cgns_hexa_grid_0000000010_id000002.cgns	

			-- cgns_hexa_grid_0000000010_id000003.cgns
			-- cgns_hexa_solution_0000000010_id000000.cgns
			-- cgns_hexa_solution_0000000010_id000001.cgns
			-- cgns_hexa_solution_0000000010_id000002.cgns
			`-- cgns_hexa_solution_0000000010_id000003.cgns
			-- 0000000020
			-- 0000000030
			-- 0000000040
			-- 0000000050
			-- 0000000060
			-- 0000000070
			-- 0000000080
			-- 0000000090
			-- 0000000100
			-- cgns_hexa_id000000.cgns
			-- cgns_hexa_id000001.cgns
			-- cgns_hexa_id000002.cgns
			`-- cgns_hexa_id000003.cgns
			-- index.dfi
			-- proc.dfi
			`-- udmlib.tp
			`-- proc.dfi
<p>CGNS:GridCoordinates と CGNS:FlowSolution を別のファイルに出力します。</p> <p>時系列毎にステップ番号の付加した CGNS ファイルを出力します。</p> <p>時系列毎にそのステップの GridCoordinates 値を出力します。</p> <p>フィールドディレクトリ"field_data"配下に、時系列の 10 桁のステップ番号ディレクトリを作成して CGNS ファイルを出力します。</p> <p>(注) ParaView のアニメーション表示では、グリッド表示と時系列が一致しません。</p>			

(注) グリッド座標が時系列毎に変化する CGNS ファイルは ParaView のアニメーション表示では、グリッド表示と時系列が一致しません。

これは、節点（ノード）の物理量データと要素（セル）の物理量データの両方が含まれる場合に発生します。

節点（ノード）の物理量データと要素（セル）の物理量データの方のみの出力の場合は、正常にアニメーション表示します。

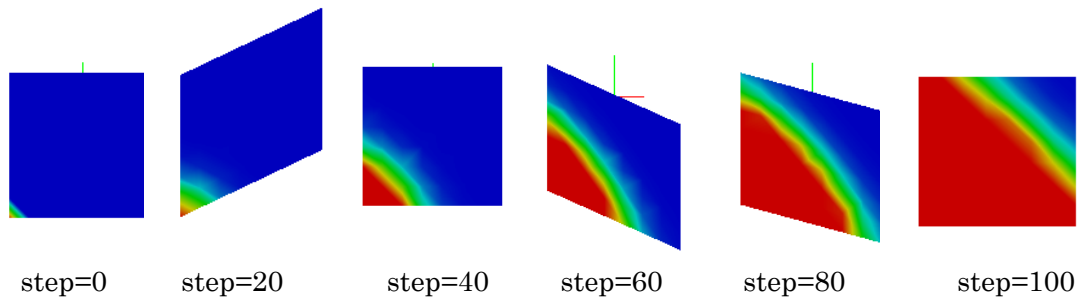
グリッドの時系列表示が正常に ParaView 表示するデータは以下に出力されます。

節点（ノード）のみの物理量データ

```
includegrid_appendstep_gridtimeslice/output_without_cell
```

要素（セル）のみの物理量データ

```
includegrid_appendstep_gridtimeslice/output_without_node
```



ParaView のアニメーション表示 : Pressure

3.2.4 DFI ファイルの入出力 (dfi)

index.dfi に記述されている情報の入出力の確認の為以下のソースファイル、実行モジュールについて説明します。

実行モジュール	ソースファイル	説明
dfi_unit	dfi_unit.cpp	index.dfi に記述されている単位系の読み書きを行います。
dfi_run.sh	/	上記の実行モジュールを実行します。

(1) dfi_unit

DFI ファイルに記述の単位系の取得、設定を行います。

読込 index.dfi に単位系を追加、取得を行い、単位系の追加した index.dfi を出力します。

以下、dfi_unit の実行例を示します。

```
./dfi_unit model_4x4x4/index.dfi
Start :: loadModel!
End :: loadModel!
add Unit : Length
add Unit : Pressure
add Unit : Temperature
add Unit : Velocity
add Unit : Mass
Remove Unit : Mass
print UnitList
Length : unit=m, Reference=3.000000e-03
Pressure : unit=Pa, Reference=0.000000e+00, Difference=5.100000e+02
Temperature : unit=C, Reference=1.000000e+01, Difference=1.000000e-01
Velocity : unit=m/s, Reference=3.400000e+00
Mass : not exists[removed].
Start :: writeModel!
End :: writeModel!
```

(2) dfi_run.sh

DFI ファイルの入出力の実行モジュールを順次実行します。

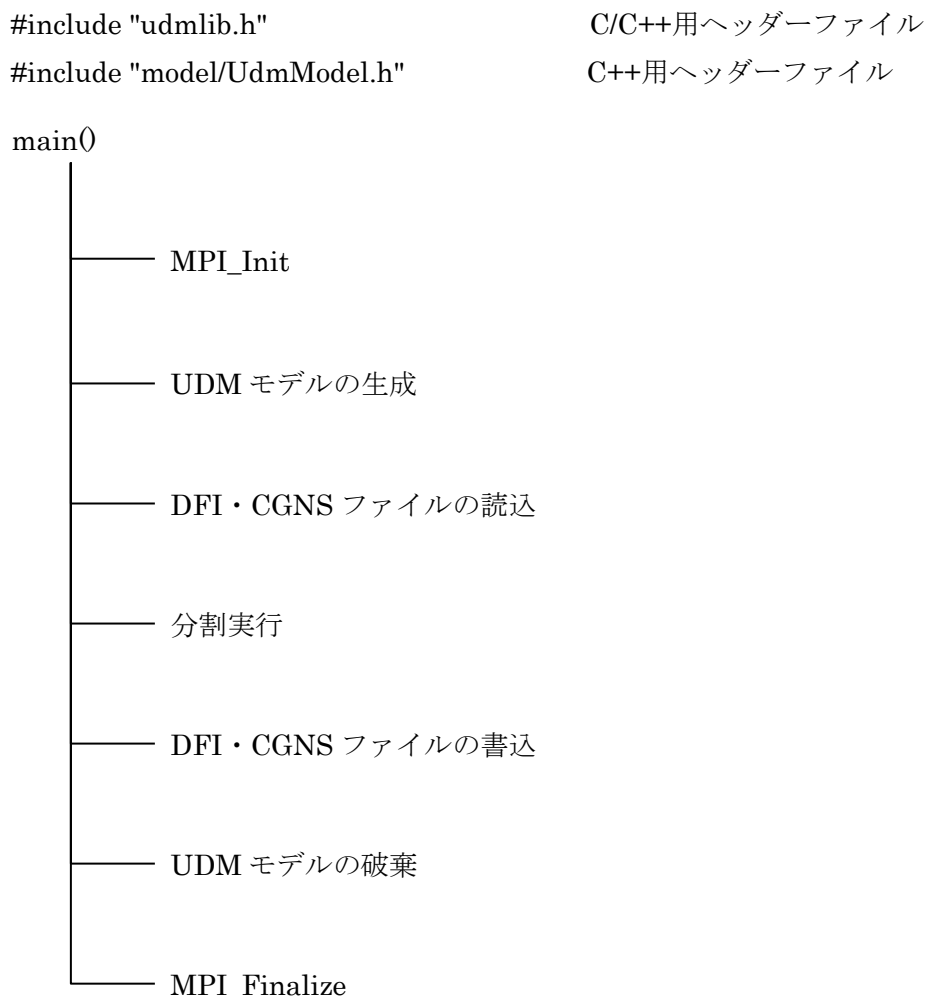
4. UDM ライブラリ API の使用方法

以下、UDM ライブラリの C/C++言語の API について説明します。

UDM ライブラリは C++プログラミングにてすべての機能を利用できます。C プログラミングでは、一部の機能しか利用できません。C++プログラミングでの利用を推奨します。

実際のプログラミングについては「3.2 利用例 (UDMLib-X.X.X/examples)」を参照してください。

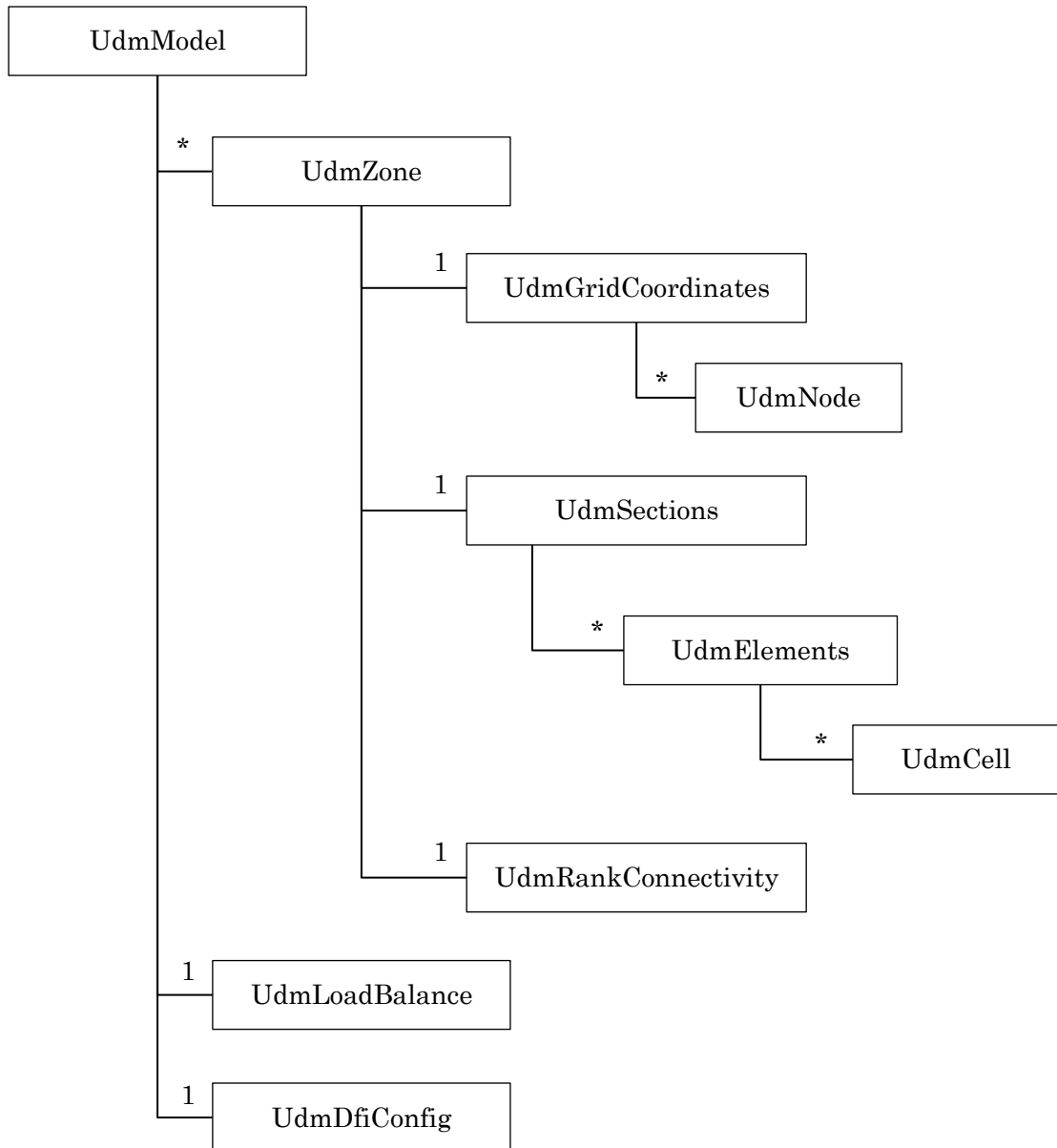
UDM ライブラリを使用したプログラムの基本的な処理を以下に示します。



以下、CXX API (C++言語の API) と CC API (C 言語の API) の説明をします。

4.1 UDM モデルの構成要素

UDM ライブラリは以下のデータ構成を持ちます。



(1) UdmModel :: UDM モデルクラス

UDM ライブラリのデータ構造のすべてを管理するクラスとなります。

(2) UdmZone :: UDM ゾーンクラス

形状、ソルバモデルにより複数の UDM ゾーンを作成することができます。

節点（ノード）、要素（セル）の構成データは UDM ゾーン単位で管理します。分割も

UDM ゾーン単位にて行い、他の UDM ゾーンには影響を与えません。

(3) UdmGridCoordinates :: 節点（ノード）座標クラス

節点（ノード）を管理します。UDM ゾーンクラスに 1 つだけ存在します。

(4) UdmNode :: 節点（ノード）クラス

モデルを構成する節点（ノード）です。座標、及び節点（ノード）に定義された複数の物理量を持ちます。

(5) UdmSections :: セクション管理クラス

セクションクラスを管理します。UDM ゾーンクラスに 1 つだけ存在します。

(5) UdmElements :: セクションクラス

要素（セル）を管理します。UDM ゾーンクラスに複数生成することができます。

(6) UdmCell :: 要素（セル）クラス

モデルを構成する要素（セル）です。構成節点（ノード）、及び要素（セル）に定義された複数の物理量を持ちます。

(7) UdmRankConnectivity :: ランク間接続情報

ランク間の接続節点（ノード）を管理します。ランク間接続節点（ノード）が内部境界となり、仮想要素（セル）の作成、分割に使用します。

(8) UdmLoadBalance :: UDM 分割クラス

UDM モデルを MPI プロセス数に分割を行います。

(9) UdmDfiConfig :: DFI 定義クラス

DFI ファイルの管理を行います。

4.1.1 UDM モデルの生成・破棄

(1) UDM モデルの生成

UDM ライブラリのデータ構造のすべてを管理するクラスを生成します。

(CXX API)

```
UdmModel::UdmModel();
```

UDM モデルクラスを生成します。

```
UdmModel::UdmModel(const MPI_Comm& comm);
```

MPI コミュニケータを指定して、UDM モデルクラスを生成します。

comm MPI コミュニケータ

使用例

```
UdmModel *model = new UdmModel();
```

...

```
delete model;
```

(CC API)

```
UdmHanler_t udm_create_model()
```

UDM モデルクラスを生成します。

戻り値 生成 UdmModel クラスポインタ

使用例

```
UdmHandler model = udm_create_model();
```

...

```
udm_delete_model(model);
```

ユーザプログラムでは、生成 UdmModel クラス、及び UdmHanler_t によりすべての API へのアクセスを行います。

(2) UDM モデルの破棄

生成した UDM モデルは終了時に破棄しなければなりません。

(CXX API)

```
UdmModel::~~UdmModel()
```

UDM モデルクラスを破棄します。

new にて UDM モデルクラスを生成した場合は、delete を行ってください。

(CC API)

```
void udm_delete_model(UdmHanler_t udm_handler)
```

UdmModel クラスオブジェクトを破棄する.

udm_handler UdmModel クラスポインタ

4.1.2 UDM ゾーンの生成、取得

UDM モデルには必ず 1 つの UDM ゾーンが必要となります。

(1) UDM ゾーンの生成

UDM ゾーンを生成します。

(CXX API)

```
UdmZone* UdmModel::createZone();
```

UDM ゾーンを生成して、UDM ゾーンを UDM モデルに追加する.

既定の命名規約の UDM ゾーン名称にてゾーンを作成する.

戻り値 生成ゾーン、ゾーンの生成に失敗した場合は NULL を返す.

```
UdmZone* UdmModel::createZone(const std::string &zone_name)
```

UDM ゾーンを生成して、UDM ゾーンを UDM モデルに追加する.

既存のゾーン名称は作成できない。

zone_name ゾーン名称

戻り値 生成ゾーン、ゾーンの生成に失敗した場合は NULL を返す.

使用例

```
UdmModel *model = new UdmModel();
```

```
UdmZone *zone = model->createZone();
```

(CC API)

```
int udm_create_zone(UdmHanler_t udm_handler);
```

UDM ゾーンを生成して、UDM ゾーンを UDM モデルに追加する.

既定の命名規約の UDM ゾーン名称にてゾーンを作成する.

udm_handler UdmModel クラスポインタ

戻り値 ゾーン ID (1 ~) : 0 の場合は生成エラー

使用例

```
UdmHandler model = udm_create_model();  
int zone_id = udm_create_zone(model);
```

既定の UDM ゾーン名称は次の書式となっています。

UdmZone#%d %d 生成 UDM ゾーンの連番 (1 ~)

UDM ゾーン名称は CGNS ファイルに出力した場合の CGNS::Zone の名前となります。

(2) UDM ゾーンの取得

生成済みの UDM ゾーンを取得を行います。

(CXX API)

```
int UdmModel::getNumZones() const
```

UDM ゾーン数を取得する.

戻り値 UDM ゾーン数

```
UdmZone* UdmModel::getZone(int zone_id) const
```

UDM ゾーンを取得する.

zone_id UDM ゾーン ID (1 ~)

戻り値 UDM ゾーン

使用例

```
UdmModel *model = new UdmModel();  
for (int zone_id = 1; zone_id <= model->getNumZones(); zone_id++) {  
    UdmZone *zone = model->getZone(zone_id);  
}
```

(CC API)

```
int udm_getnum_zones(UdmHanler_t udm_handler)
```

UDM ゾーン数を取得する.

udm_handler UdmModel クラスポインタ

戻り値 UDM ゾーン数

使用例

```
UdmHandler model = udm_create_model();  
int num_zone = udm_getnum_zones(model);
```

4.1.3 節点（ノード）座標クラスの取得

節点（ノード）管理クラスは節点（ノード）を管理します。UDM ゾーンクラスに 1 つだけ存在します。

(CXX API)

UdmGridCoordinates* UdmZone::getGridCoordinates() const

グリッド座標クラスを取得する.

戻り値 グリッド座標クラス

使用例

```
UdmModel *model = new UdmModel();  
UdmZone *zone = model->createZone();  
UdmGridCoordinates *grid = zone->getGridCoordinates();
```

4.1.4 UDM セクション管理クラスの取得

UDM ゾーンには 1 つの UDM セクション管理クラスが存在します。

UDM セクション管理クラスを取得します。

(CXX API)

UdmSections* UdmZone::getSections() const;

要素管理クラスを取得する.

戻り値 要素管理クラス

使用例

```
UdmModel *model = new UdmModel();  
UdmZone *zone = model->createZone();  
UdmSections *sections = zone->getSections();
```

4.1.5 UDM セクションの生成、取得

UDM ゾーンには必ず 1 つの UDM セクションが必要となります。

(1) UDM セクションの生成

UDM セクションを生成します。

(CXX API)

<pre>UdmElements * UdmSections::createSection(UdmElementType_t element_type);</pre> <p>セクション（要素構成）を作成する。 セクション（要素構成）を作成して、セクション（要素構成）リストに追加する。 既定の命名規約に従いセクション（要素構成）名称を作成する。</p> <p>element_type 要素タイプ 戻り値 生成セクション（要素構成）</p>
<pre>UdmElements * UdmSections::createSection(const std::string &section_name, UdmElementType_t element_type);</pre> <p>セクション（要素構成）を作成する。 セクション（要素構成）を作成して、セクション（要素構成）リストに追加する。 既存セクション（要素構成）名称は、作成できない。（NULL を返す）</p> <p>section_name セクション（要素構成）名称 element_type 要素タイプ 戻り値 生成セクション（要素構成）</p>
<p>使用例</p> <pre>UdmZone *zone = model->createZone(); UdmSections *sections = zone->getSections() // HEX_8 要素の設定 UdmSections *sections = zone->getSections(); UdmElements *elements = sections->createSection(Udm_HEX_8);</pre>

(CC API)

<pre>int udm_create_section(</pre>

<pre> UdmHanler_t udm_handler, int zone_id, UdmElementType_t element_type); </pre> <p>セクション（要素構成）を作成する.</p> <p>セクション（要素構成）を作成して、セクション（要素構成）リストに追加する.</p> <p>既定の命名規約に従いセクション（要素構成）名称を作成する.</p> <p>udm_handler UdmModel クラスポインタ</p> <p>zone_id ゾーン ID</p> <p>element_type 要素タイプ</p> <p>戻り値 生成セクション（要素構成）ID（1～）：0の場合は生成エラー</p>
<p>使用例</p> <pre> UdmHandler model = udm_create_model(); int zone_id = udm_create_zone(model); int element_id = udm_create_section(model, zone_id, Udm_HEXA_8); </pre>

既定の UDM セクション名称は次の書式となっています。

UdmElements_%s %s：要素タイプの文字列表現

UDM セクション名称は CGNS ファイルに出力した場合の CGNS::Elements の名前となります。

(2) セクションの取得

生成済みのセクションを取得します。

(CXX API)

<pre> int UdmSections::getNumSections() const; </pre> <p>セクション（要素構成）数を取得する.</p> <p>戻り値 セクション（要素構成）数</p>
<pre> UdmElements* UdmSections::getSection(int section_id); </pre> <p>セクション ID からセクション（要素構成）を取得する</p> <p>section_id セクション ID（1～）</p> <p>戻り値 セクション（要素構成）</p>
<pre> UdmElements* UdmSections::getSection(const std::string& section_name); </pre>

セクション名称からセクション（要素構成）を取得する.

section_name セクション名称

戻り値 セクション（要素構成）

使用例

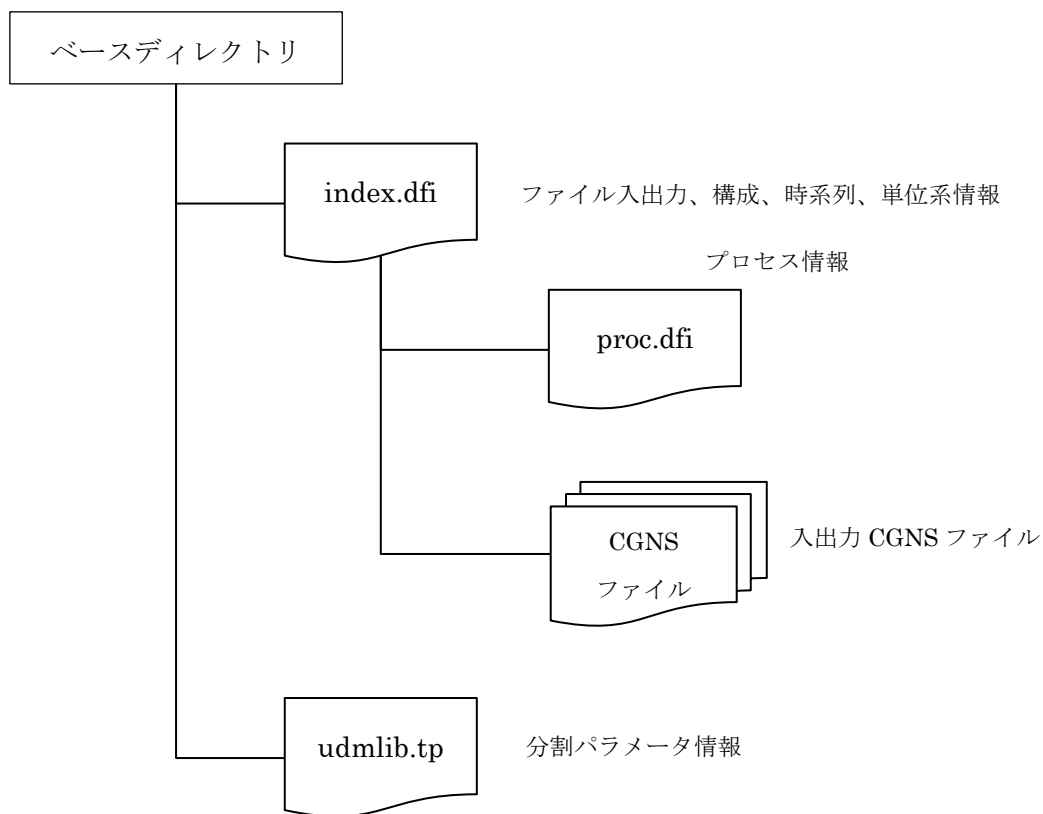
```
UdmZone *zone = model->createZone();
UdmSections *sections = zone->getSections()
// HEX_8 要素の設定
UdmSections *sections = zone->getSections();
for (int section_id=1; section_id<=sections->getNumSections(); section_id++) {
    UdmElements *elements = sections->getSection(section_id);
}
```

4.2 DFI ファイルの入出力

ソルバパラメータ、ファイルの入出力パラメータ等の情報は、ユーザプログラムから設定を行うことも可能ですが、それらを DFI ファイルに記述することによりファイルの入出力の一連の処理を UDM ライブラリが行います。

このことによりユーザは、ソルバのプログラム部分に専念することができます。

UDM ライブラリの DFI ファイルを含めたファイル構成を以下に示します。



UDM ライブラリファイル構成

a) ベースディレクトリ

読込を行った DFI (index.dfi) ファイルのディレクトリとなります。

UDM ライブラリの相対パスの設定は、すべてベースディレクトリを基準とした相対パスとなります。

b) index.dfi

読込を行う DFI ファイルとなります。ファイル入出力、構成、時系列、単位系の情報を持ちます。

任意のファイル名を指定可能です。

c) proc.dfi

プロセス情報が出力されます。

DFI (index.dfi) ファイルに記述された"proc.dfi"ファイルを読み込み、書込みを行います。

ユーザが直接編集することはありません。

d) CGNS ファイル

入出力を行う CGNS ファイルです。

出力ディレクトリ、CGNS 構成は、DFI (index.dfi) ファイルに記述します。

e) udmlib.tp

分割パラメータを記述、又は分割時のパラメータを出力します。

ファイル名は udmlib.tp 固定です。

4.2.1 DFI ファイルの読込

DFI ファイルを指定して、記述されている CGNS ファイルを読み込みます。

読み込む時系列ステップ番号を指定してリスタートを行うことも可能です。

(CXX API)

<pre>UdmError_t UdmModel::loadModel(const char* dfi_filename, int timeslice_step);</pre> <p>DFI ファイルの設定情報に従って、CGNS ファイルを読み込む。</p> <p>dfi_filename index.dfi ファイル名</p> <p>timeslice_step 読込ステップ番号</p> <p>戻り値 エラー番号 : UDM_OK UDM_ERROR</p>
<p>使用例</p> <pre>int myrank, num_procs; const char dfiname[] = {"index.dfi"}; MPI_Init(&argc, &argv); MPI_Comm_rank(MPI_COMM_WORLD, &myrank); MPI_Comm_size(MPI_COMM_WORLD, &num_procs); // モデルの生成 UdmModel *model = new UdmModel(); // DFI ファイルの読込 if (model->loadModel(dfiname, 0) != UDM_OK) { printf("Error : can not load model[index.dfi=%s].%n", dfiname); delete model;</pre>


```

        MPI_Finalize();
        return -1;
    }
    printf("[rank=%d] End :: loadModel!¥n", myrank);

```

(CC API)

```

UdmError_t udm_load_model(
    UdmHanler_t udm_handler,
    const char* dfi_filename,
    int timeslice_step);

```

DFI ファイルの設定情報に従って、CGNS ファイルを読み込む。

udm_handler UdmModel クラスポインタ

dfi_filename index.dfi ファイル名 (-1 = 最初の時系列データ)

timeslice_step 読込ステップ番号

戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```

int myrank, num_procs;
char dfilename[] = {"index.dfi"};

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

// モデルの生成
UdmHanler_t model = udm_create_model();
// DFI ファイルの読込
if (udm_load_model(model, dfilename, 0) != UDM_OK) {
    printf("[rankno=%d] Error : can not load model[index.dfi=%s].¥n",
        myrank, dfilename);
    udm_delete_model(model);
    MPI_Finalize();
    return -1;
}
printf("[rankno=%d] End :: loadModel!¥n", myrank);

```

4.2.2 DFI ファイルの書込

DFI ファイルの記述に従って CGNS ファイルを出力します。
出力パラメータ、時系列情報を DFI ファイルに上書き出力します。

(CXX API)

```
UdmError_t UdmModel::writeModel(int timeslice_step, float timeslice_time);
```

DFI 設定に従って、CGNS ファイル, index.dfi を出力する.

timeslice_step 時系列ステップ数

timeslice_time 時系列ステップ時間

戻り値 エラー番号 : UDM_OK | UDM_ERROR

```
UdmError_t UdmModel::writeModel(
```

```
int timeslice_step,
```

```
float timeslice_time,
```

```
int average_step,
```

```
float average_time);
```

DFI 設定に従って、CGNS ファイル, index.dfi を出力する.

timeslice_step 時系列ステップ数

timeslice_time 時系列ステップ時間

average_step 平均ステップ数

average_time 平均ステップ時間

戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```
int myrank, num_procs;
```

```
const char dfiname[] = {"index.dfi"};
```

```
MPI_Init(&argc, &argv);
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

```
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
```

```
// モデルの生成
```

```
UdmModel *model = new UdmModel();
```

```
// DFI ファイルの書込
```

```

ret = model->writeModel(0, 0.0);
if (ret != UDM_OK) {
    printf("Error : can not write model.¥n");
    delete model;
    MPI_Finalize();
    return -1;
}
printf("[rank=%d] End :: writeModel!¥n", myrank);

```

(CC API)

```

UdmError_t udm_write_model( UdmHanler_t udm_handler,
                           int timeslice_step,
                           float timeslice_time);

```

DFI 設定に従って、CGNS ファイル, index.dfi を出力する.

udm_handler UdmModel クラスポインタ

timeslice_step 時系列ステップ数

timeslice_time 時系列ステップ時間

戻り値 エラー番号 : UDM_OK | UDM_ERROR

```

UdmError_t udm_write_model_average(
                           UdmHanler_t udm_handler,
                           int timeslice_step,
                           float timeslice_time,
                           int average_step,
                           float average_time);

```

DFI 設定に従って、CGNS ファイル, index.dfi を出力する.

udm_handler UdmModel クラスポインタ

timeslice_step 時系列ステップ数

timeslice_time 時系列ステップ時間

average_step 平均ステップ数

average_time 平均ステップ時間

戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```

int myrank, num_procs;

```

```

char dfname[] = {"index.dfi"};

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

// モデルの生成
UdmHanler_t model = udm_create_model();
// DFI ファイルの書込
UdmError_t ret = udm_write_model(model, 0, 0.0);
if (ret != UDM_OK) {
    printf("[rankno=%d] Error : can not write model.¥n", myrank);
    udm_delete_model(model);
    MPI_Finalize();
    return -1;
}
printf("[rankno=%d] End :: loadModel!¥n", myrank);

```

4.2.3 DFI: ファイル情報

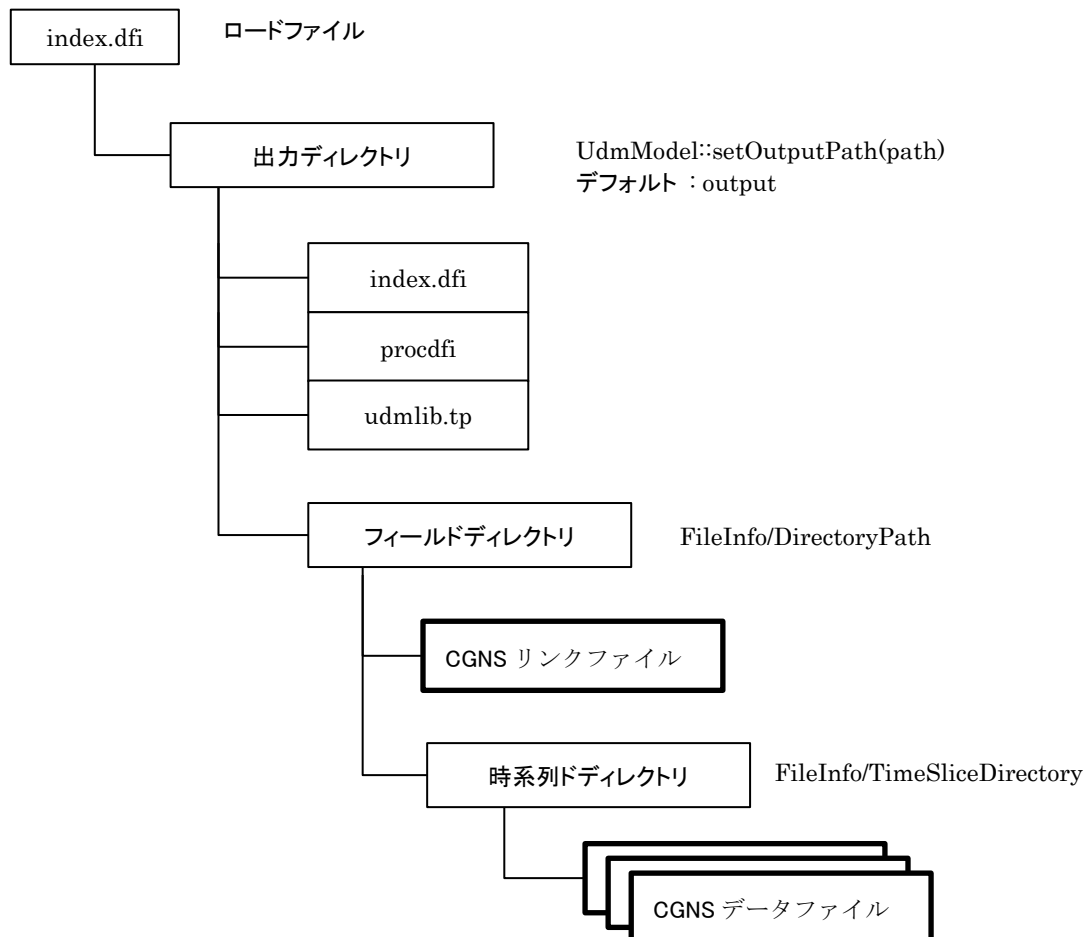
DFI ファイルに記述されているファイル情報を取得、設定します。

(ファイル情報 : FileInfo)

リーフ	ラベル	値	オプション (default)	項目名
FileInfo	ファイル情報の設定項目を記述します。			
	DFIType	"Uns"	Option default="Uns"	DFI 種別 "Uns"固定
	DirectoryPath	相対パス 絶対パス	Option default="."/	出力フィールドデ ータディレクトリ
	TimeSliceDirectory	on off	Option default="off"	時刻ディレクトリ 作成オプション
	Prefix	ベースファイル名	Request	ベースファイル名
	FileFormat	"cgns"	Option default="cgns"	ファイルフォーマ ット

リーフ	ラベル	値	オプション (default)	項目名
	FieldFilenameFormat	step_rank rank_step	Option default="step_rank"	ファイル命名書式
	FileCompositionType	IncludeGrid ExcludeGrid AppendStep EachStep GridConstant GridTimeSlice	Option default= @list("IncludeGrid", "EachStep")	CGNS ファイル構 成タイプ

ファイル情報（FileInfo）の設定による出力ディレクトリ構成は以下のようになります。



(1) 出力ディレクトリ

出力ディレクトリは、DFI の設定項目でなく、ユーザプログラムからの設定項目となります。デフォルト出力ディレクトリは、"output"です。

(CXX API)

<pre>const std::string& UdmDfiConfig::getOutputPath() const</pre> <p>出力ディレクトリを取得する。</p> <p>戻り値 出力ディレクトリ</p>
<pre>void UdmDfiConfig::setOutputPath(const std::string path)</pre> <p>出力ディレクトリを設定する。</p> <p>path 出力ディレクトリ</p>
<pre>void UdmModel::setOutputPath(const char* path)</pre> <p>出力ディレクトリを設定する。</p> <p>path 出力ディレクトリ</p>
<p>使用例</p> <pre>UdmModel *model = new UdmModel(); UdmDfiConfig* config = model->getDfiConfig(); config->setOutputPath("solutions_data"); std::string path = config->getOutputPath();</pre>

(CC API)

<pre>const char* udm_config_getoutputpath(UdmHanler_t udm_handler, char* path)</pre> <p>出力ディレクトリを取得する。</p> <p>[in] udm_handler UdmModel クラスポインタ</p> <p>[out] prefix 出力ディレクトリ</p> <p>戻り値 出力ディレクトリポインタ : NULL の場合は取得エラー</p>
<pre>void udm_config_setoutputpath(UdmHanler_t udm_handler, const char* path)</pre> <p>出力ディレクトリを設定する。</p> <p>udm_handler UdmModel クラスポインタ</p> <p>path 出力ディレクトリ</p>
<p>使用例</p>

```
char path[128] = {0x00};
UdmHanler_t model = udm_create_model();
udm_config_setoutputpath(model, "solutions_data");
udm_config_getoutputpath(model, path);
```

(2) フィールドディレクトリ : FileInfo/DirectoryPath

フィールドディレクトリは、CGNS ファイルの出力先ディレクトリ名です。デフォルトは"./"であり前述の出力ディレクトリに出力されます。

(CXX API)

```
UdmError_t UdmFileInfoConfig::getDirectoryPath(std::string& directorypath) const
    フィールドデータディレクトリを取得する.
[out] directorypath    フィールドデータディレクトリ
戻り値    エラー番号 : UDM_OK | UDM_ERROR
```

```
void UdmFileInfoConfig::setDirectoryPath(const std::string& directory_path)
    フィールドデータディレクトリを設定する.
directory_path    フィールドデータディレクトリ
```

使用例

```
UdmModel *model = new UdmModel();
UdmDfiConfig* config = model->getDfiConfig();
UdmFileInfoConfig* fileinfo = config->getFileinfoConfig();
std::string path;
fileinfo->setDirectoryPath("cgns_fields");
fileinfo->getDirectoryPath(path);
```

(CC API)

```
const char* udm_config_getfielddirectory(
    UdmHanler_t udm_handler,
    char* directory)
    フィールドデータディレクトリを取得する.
[in] udm_handler    UdmModel クラスポインタ
```

[out] directorypath フィールドデータディレクトリ
 戻り値 取得フィールドデータディレクトリポインタ

```
void udm_config_setfielddirectory(
    UdmHanler_t udm_handler,
    const char* directory)
    フィールドデータディレクトリを設定する.
    udm_handler      UdmModel クラスポインタ
    directory_path      フィールドデータディレクトリ
```

使用例

```
char path[128] = {0x00};
UdmHanler_t model = udm_create_model();
udm_config_setfielddirectory(model, "cgns_fields");
udm_config_getfielddirectory(model, path);
```

(3) 時系列ディレクトリ : FileInfo/TimeSliceDirectory

時系列ディレクトリは、CGNS ファイルの出力を時系列ステップ番号別のディレクトリに出力するフラグです。デフォルトは"false"であり時系列ステップ番号別のディレクトリには出力しません。

時系列ディレクトリは 10 桁の時系列ステップ番号となります。

以下、4 並列、100 ステップ実行の時系列ディレクトリの出力例です。

```
.
|-- 0000000000
|   |-- cgns_hexa_grid_0000000000_id000000.cgns
|   |-- cgns_hexa_grid_0000000000_id000001.cgns
|   |-- cgns_hexa_grid_0000000000_id000002.cgns
|   |-- cgns_hexa_grid_0000000000_id000003.cgns
|   |-- cgns_hexa_solution_0000000000_id000000.cgns
|   |-- cgns_hexa_solution_0000000000_id000001.cgns
|   |-- cgns_hexa_solution_0000000000_id000002.cgns
|   |-- cgns_hexa_solution_0000000000_id000003.cgns
|-- 0000000010
|-- 0000000020
```



```

|-- 0000000030
|-- 0000000040
|-- 0000000050
|-- 0000000060
|-- 0000000070
|-- 0000000080
|-- 0000000090
|-- 0000000100
|   |-- cgns_hexa_grid_0000000100_id000000.cgns
|   |-- cgns_hexa_grid_0000000100_id000001.cgns
|   |-- cgns_hexa_grid_0000000100_id000002.cgns
|   |-- cgns_hexa_grid_0000000100_id000003.cgns
|   |-- cgns_hexa_solution_0000000100_id000000.cgns
|   |-- cgns_hexa_solution_0000000100_id000001.cgns
|   |-- cgns_hexa_solution_0000000100_id000002.cgns
|   |-- cgns_hexa_solution_0000000100_id000003.cgns
|-- cgns_hexa_id000000.cgns
|-- cgns_hexa_id000001.cgns
|-- cgns_hexa_id000002.cgns
|-- cgns_hexa_id000003.cgns
|-- index.dfi
|-- proc.dfi
`-- udmllib.tp

```

(CXX API)

```
bool UdmFileInfoConfig::isTimeSliceDirectory() const
```

時刻ディレクトリ作成オプションを取得する.

戻り値 時刻ディレクトリ作成オプション

```
void UdmFileInfoConfig::setTimeSliceDirectory(bool timeslice_directory)
```

時刻ディレクトリ作成オプションを設定する.

timeslice_directory 時刻ディレクトリ作成オプション

使用例

```
UdmModel *model = new UdmModel();
```

```
UdmDfiConfig* config = model->getDfiConfig();
UdmFileInfoConfig* fileinfo = config->getFileinfoConfig();
fileinfo->setTimeSliceDirectory(true);
```

(CC API)

```
bool udm_config_istimeslicedirectory(
    UdmHanler_t udm_handler)
    時刻ディレクトリ作成オプションを取得する.
    udm_handler          UdmModel クラスポインタ
    戻り値              時刻ディレクトリ作成オプション

void udm_config_settimeslicedirectory(
    UdmHanler_t udm_handler,
    bool timeslice_directory)
    時刻ディレクトリ作成オプションを設定する.
    udm_handler          UdmModel クラスポインタ
    timeslice_directory  時刻ディレクトリ作成オプション
```

使用例

```
UdmHanler_t model = udm_create_model();
udm_config_settimeslicedirectory(model, true);
```

(4) ベースファイル名 : FileInfo/Prefix

出力 CGNS ファイル名の接頭文字を取得、設定します。

出力 CGNS ファイル名の命名規約は以下です。

[接頭文字]_[ステップ番号:10 桁]_id[ランク番号:6 桁].cgns

(例) cgns_hexa_0000000000_id000000.cgns

[接頭文字]	cgns_hexa	
[ステップ番号:10 桁]	0000000000	(0 ステップ)
[ランク番号:6 桁]	00000000	(ランク 0)

(CXX API)

`UdmError_t UdmFileInfoConfig::getPrefix(std::string& prefix) const`

ベースファイル名を取得する.

[out] prefix ベースファイル名

戻り値 エラー番号 : UDM_OK | UDM_ERROR

`void UdmFileInfoConfig::setPrefix(const std::string& prefix)`

ベースファイル名を設定する.

prefix ベースファイル名

使用例

```
UdmModel *model = new UdmModel();
UdmDfiConfig* config = model->getDfiConfig();
UdmFileInfoConfig* fileinfo = config->getFileinfoConfig();
fileinfo->setPrefix("cgns_hexa");
```

(CC API)

`const char* udm_config_getfileprefix(UdmHanler_t udm_handler, char* prefix)`

ベースファイル名を取得する.

[in] udm_handler UdmModel クラスポインタ

[out] prefix ベースファイル名

戻り値 取得ベースファイル名ポインタ : NULL の場合は取得エラー

`void udm_config_setfileprefix(UdmHanler_t udm_handler, const char* prefix)`

ベースファイル名を設定する.

udm_handler UdmModel クラスポインタ

prefix ベースファイル名

使用例

```
UdmHanler_t model = udm_create_model();
udm_config_setfileprefix(model, "cgns_hexa");
```

(5) ファイル命名書式 : FileInfo/FieldFilenameFormat

出力 CGNS ファイル名のファイル命名書式を取得、設定します。

出力 CGNS ファイル名のファイル命名書式は以下です。

ファイル命名書式	ファイル命名書式
step_rank (デフォルト)	[接頭文字]_[ステップ番号:10桁]_id[ランク番号:6桁].cgns
rank_step	[接頭文字]_id[ランク番号:6桁]_[ステップ番号:10桁].cgns

(CXX API)

`UdmFieldFilenameFormat_t UdmFileInfoConfig::getFieldfilenameFormat() const`

ファイル命名書式を取得する。

戻り値 ファイル命名書式

`void UdmFileInfoConfig::setFieldfilenameFormat(`

`UdmFieldFilenameFormat_t fieldfilename_format)`

ファイル命名書式を設定する。

fieldfilename_format ファイル命名書式

使用例

```
UdmModel *model = new UdmModel();
UdmDfiConfig* config = model->getDfiConfig();
UdmFileInfoConfig* fileinfo = config->getFileinfoConfig();
fileinfo->setFieldfilenameFormat(Udm_rank_step);
```

(6) CGNS ファイル構成タイプ : FileInfo/FileCompositionType

出力 CGNS ファイルの出力構成を取得、設定します。

出力 CGNS ファイルを節点（ノード）の座標、要素（セル）の構成の形状データと物理量データを別々、又は時系列毎の出力構成を設定します。

以下の入出力ファイルの構成タイプの組合せにより出力構成を設定します。

CGNS:GridCoordinates と CGNS:FlowSolution の出力方法	
IncludeGrid (デフォルト)	CGNS:GridCoordinates と CGNS:FlowSolution を 1 つのファイルに出力します。

	ExcludeGrid と同時に使用することはできません。
ExcludeGrid	CGNS:GridCoordinates と CGNS:FlowSolution を別ファイルに出力します。 IncludeGrid と同時に使用することはできません。
CGNS:FlowSolution のステップ出力方法	
AppendStep	CGNS:FlowSolution をステップ毎に追加して 1 つのファイルに出力します。 EachStep と同時に使用することはできません。
EachStep (デフォルト)	CGNS:FlowSolution をステップ毎に別ファイルにします。 AppendStep と同時に使用することはできません。
CGNS:GridCoordinates の時系列出力方法	
GridConstant (デフォルト)	CGNS:GridCoordinates は最初の 1 回のみ出力を行います。 ノード座標が時系列毎に変化するしない、又は変位データは物理量として出力を行い CGNS:GridCoordinates の座標データは変化しない場合に使用します。 GridTimeSlice と同時に使用することはできません。
GridTimeSlice	CGNS:GridCoordinates を時系列毎に出力を行う。 ノード座標が時系列毎に変化する場合に使用します。 出力前に CGNS:GridCoordinates の座標データを更新する必要があります。 GridConstant と同時に使用することはできません。

CGNS ファイル構成タイプ：設定項目一覧

FileCompositionType (CGNS ファイル構成タイプ) の組合せにより以下のファイル構成となります。



(C) Constant 物理量データ

IncludeGrid	AppendStep	GridConstant	出力ファイル	出力動作
ExcludeGrid	EachStep	GridTimeSlice		

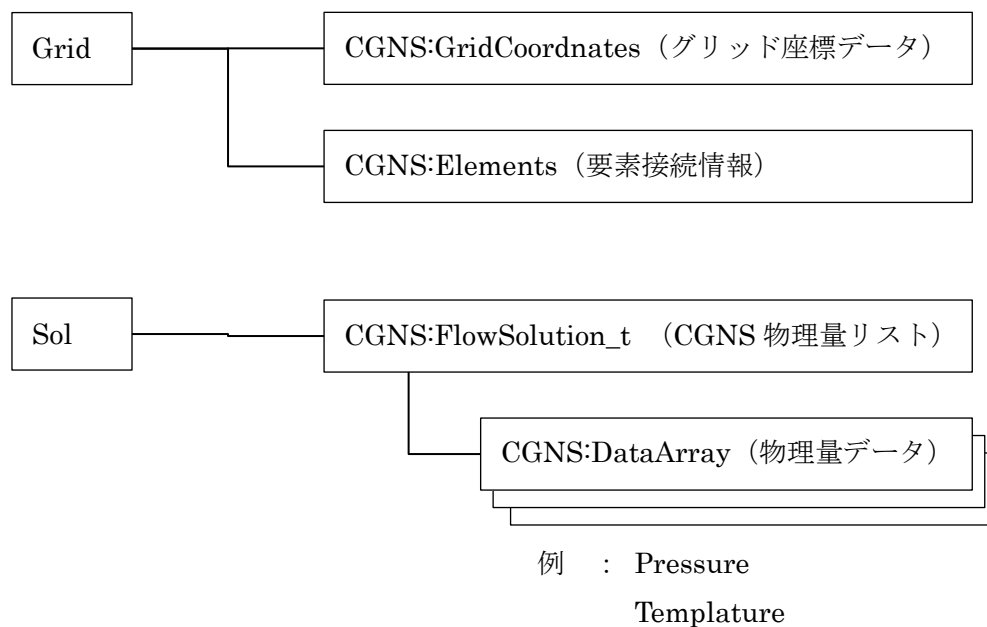
IncludeGrid ExcludeGrid	AppendStep EachStep	GridConstant GridTimeSlice	出力ファイル	出力動作
IncludeGrid	AppendStep	GridConstant	CGNS File : 1	1 つの CGNS ファイルに 1 つの CGNS:GridCoordinates と 時系列の CGNS:FlowSolution を出 力します。 デフォルト出力設定です。
		<div> <div>Link</div> <div>出力なし</div> <div> <div>1 座標データ</div> <div>Grid</div> <div>Sol1(C) Sol2 Sol3</div> <div>時系列物理データ</div> </div> </div>		
		GridTimeSlice	CGNS File : 1	1 つの CGNS ファイルに 時 系 列 の CGNS:GridCoordinates と CGNS:FlowSolution を出力しま す。
		<div> <div>Link</div> <div>出力なし</div> <div> <div>時系列座標データ</div> <div>Grid1 Grid2 Grid3</div> <div>Sol1(C) Sol2 Sol3</div> <div>時系列物理データ</div> </div> </div>		

IncludeGrid ExcludeGrid	AppendStep EachStep	GridConstant GridTimeSlice	出力ファイル	出力動作
IncludeGrid	EachStep	無効	Step File : * Link File : 1	ステップ毎の CGNS ファイルに CGNS:GridCoordinates と 1 ステップの CGNS:FlowSolution を出力します。 また、すべてのステップの CGNS ファイルへのリンクファイルを作成します。
<div style="text-align: center;"> <p>グリッドデータリンク x 3 時系列物理データリンク x 3</p> <pre> graph LR Link[Link] --- Grid1[Grid1 ----- Sol1(C)] Link --- Grid2[Grid2 ----- Sol2] Link --- Grid3[Grid3 ----- Sol3] </pre> </div>				

IncludeGrid ExcludeGrid	AppendStep EachStep	GridConstant GridTimeSlice	出力ファイル	出力動作
ExcludeGrid	AppendStep	GridConstant	Grid File : 1 Solution File : 1 Link File : 1	CGNS:GridCoordinates は別ファイルとして出力します。 1つのCGNSファイルに全ステップのCGNS:FlowSolutionを出力します。 CGNS:GridCoordinates とCGNS:FlowSolution をリンクしたリンクファイルを作成します。
		<div style="text-align: center;"> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;">Link</div> <div style="margin-top: 10px;"> グリッドデータリンク x 1 時系列物理データリンク x 3 </div> <div style="display: flex; justify-content: center; align-items: center; margin-top: 10px;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Grid</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Sol1(C)</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Sol2</div> <div style="border: 1px solid black; padding: 5px;">Sol3</div> </div> </div>		
		GridTimeSlice	Grid File : 1 Solution File : 1 Link File : 1	CGNS:GridCoordinates は別ファイルとして出力します。 1つのCGNSファイルに全ステップのCGNS:FlowSolutionを出力します。 CGNS:GridCoordinates ファイルにCGNS:FlowSolutionのリンクを追加します。
		<div style="text-align: center;"> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;">Link</div> <div style="margin-top: 10px;"> グリッドデータリンク x 3 時系列物理データリンク x 3 </div> <div style="display: flex; justify-content: center; align-items: center; margin-top: 10px;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Grid1</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Grid2</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Grid3</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Sol1(C)</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Sol2</div> <div style="border: 1px solid black; padding: 5px;">Sol3</div> </div> </div>		

IncludeGrid	AppendStep	GridConstant	出力ファイル	出力動作
ExcludeGrid	EachStep	GridTimeSlice		
ExcludeGrid	EachStep	GridConstant	Grid File : 1 Solution File : * Link File : 1	CGNS:GridCoordinates, CGNS:FlowSolution は別ファイル として出力します。 時系列毎の CGNS:FlowSolution を 出力します。
<div> <div>Link</div> <div> グリッドデータリンク x 1 時系列物理データリンク x 3 </div> <div> <div>Grid</div> <div> <div>Sol1(C)</div> <div>Sol2</div> <div>Sol3</div> </div> </div> </div>				
		GridTimeSlice	Grid File : * Solution File : * Link File : 1	CGNS:GridCoordinates, CGNS:FlowSolution は別ファイル として出力します。 時 系 列 毎 の CGNS:GridCoordinates, CGNS:FlowSolution を出力しま す。
<div> <div>Link</div> <div> グリッドデータリンク x 3 時系列物理データリンク x 3 </div> <div> <div>Grid1</div> <div>Grid2</div> <div>Grid3</div> <div> <div>Sol1(C)</div> <div>Sol2</div> <div>Sol3</div> </div> </div> </div>				

上記の Grid, Sol はそれぞれ座標データ、物理量データを示すものですが、以下の CGNS ノードを出力します。



"Link"は CGNS のリンクファイルを意味し、リンクされた時系列のリンク情報を持つことにより、すべての時系列、グリッド、物理量データを包括します。

(CXX API)

```
void UdmFileInfoConfig::setFileCompositionType(UdmFileCompositionType_t type)
```

CGNS ファイル構成タイプを設定する.

同一設定が不可なタイプが存在する場合は、置換を行う。

CGNS:GridCoordinates 出力方法 : [IncludeGrid | ExcludeGrid]

CGNS:FlowSolution のステップ出力方法 : [AppendStep | EachStep]

CGNS:GridCoordinates の時系列出力方法 : [GridConstant | GridTimeSlice]

type	CGNS ファイル構成タイプ ^o
------	-----------------------------

```
bool UdmFileInfoConfig::existsFileCompositionType(
```

UdmFileCompositionType_t type) const

CGNS ファイル構成タイプをが設定されているかチェックする

type	CGNS ファイル構成タイプ ^o
------	-----------------------------

戻り値 true=設定済み

使用例

```

UdmModel *model = new UdmModel();
UdmDfiConfig* config = model->getDfiConfig();
UdmFileInfoConfig* fileinfo = config->getFileInfoConfig();
fileinfo->setFileCompositionType(Udm_IncludeGrid);
fileinfo->setFileCompositionType(Udm_AppendStep);

```

(CC API)

```

void udm_config_setfilecomposition(
    UdmHanler_t udm_handler,
    UdmFileCompositionType_t type)
    CGNS ファイル構成タイプを設定する.
    同一設定が不可なタイプが存在する場合は、置換を行う.
    CGNS:GridCoordinates 出力方法          : [IncludeGrid | ExcludeGrid]
    CGNS:FlowSolution のステップ出力方法    : [AppendStep | EachStep]
    CGNS:GridCoordinates の時系列出力方法    : [GridConstant | GridTimeSlice]
    udm_handler          UdmModel クラスポインタ
    type                CGNS ファイル構成タイプ

bool udm_config_existsfilecomposition(
    UdmHanler_t udm_handler,
    UdmFileCompositionType_t type)
    CGNS ファイル構成タイプをが設定されているかチェックする
    udm_handler          UdmModel クラスポインタ
    type                CGNS ファイル構成タイプ
    戻り値              true=設定済み

```

使用例

```

UdmHanler_t model = udm_create_model();
udm_config_setfilecomposition(model, Udm_IncludeGrid);
udm_config_setfilecomposition(model, Udm_AppendStep);

```

4.2.5 DFI:単位系情報

DFI ファイルに記述されている単位系情報を取得、設定します。

(単位系情報 : UnitList)

リーフ	ラベル	値	オプション (default)	項目名
UnitList	単位系情報の設定項目を記述します。			単位系情報
[単位系名称]		Length Velocity Pressure Temperature	Option	単位系名称
	Unit	単位 (文字列)	Request	単位
	Reference	基準値 (実数)	Option	基準値
	Difference	差分値 (実数)	Option	差分値

(CXX API)

```
UdmError_t UdmUnitListConfig::getUnit(  
    const std::string &unit_name,  
    std::string& unit) const  
単位系名称の単位系の単位を取得する.  
[in] unit_name      単位系名称  
[out] unit          単位  
戻り値             エラー番号 : UDM_OK | UDM_ERROR
```

```
UdmError_t UdmUnitListConfig::getReference(  
    const std::string &unit_name,  
    float &reference) const  
単位系名称の単位系の基準値を取得する.  
[in] unit_name      単位系名称  
[out] reference      基準値  
戻り値             エラー番号 : UDM_OK | UDM_ERROR
```

```
UdmError_t UdmUnitListConfig::getDifference(  
    const std::string &unit_name,  
    float &difference) const  
単位系名称の単位系の差分値を取得する.  
[in] unit_name      単位系名称
```

[out] difference 差分値
戻り値 エラー番号 : UDM_OK | UDM_ERROR

```
UdmError_t UdmUnitListConfig::setUnitConfig(  
    const std::string &unit_name,  
    const std::string &unit,  
    float reference)
```

単位系を設定する（差分値を除く）.
差分値を除き、単位系を設定する.
同名の単位系が存在していた場合、上書きする.
同名の単位系が存在しない場合、追加する.

unit_name 単位系名称
unit 単位
reference 基準値
戻り値 エラー番号 : UDM_OK | UDM_ERROR,etc

```
UdmError_t UdmUnitListConfig::setUnitConfig(  
    const std::string &unit_name,  
    const std::string &unit,  
    float reference,  
    float difference)
```

単位系を設定する（差分値を含む）.
差分値を含めて、単位系を設定する.
同名の単位系が存在していた場合、上書きする.
同名の単位系が存在しない場合、追加する.

unit_name 単位系名称
unit 単位
reference 基準値
difference 差分値
戻り値 エラー番号 : UDM_OK | UDM_ERROR,etc

```
bool UdmUnitListConfig::existsUnitConfig(const std::string &unit_name) const  
    単位系が存在するかチェックする.  
unit_name          単位系名称  
戻り値          true=単位系が存在する
```

UdmError_t UdmUnitListConfig::removeUnitConfig(const std::string &unit_name)

単位系を削除する.

unit_name 単位系名称

戻り値 エラー番号 : UDM_OK | UDM_ERROR,etc

使用例

```
UdmModel *model = new UdmModel();
UdmDfiConfig* config = model->getDfiConfig();
UdmUnitListConfig *units_config = config->getUnitListConfig();
// 単位系の追加
// Length { Unit = "M", Reference = 1.000000e-03}
if (!units_config->existsUnitConfig("Length")) {
    units_config->setUnitConfig("Length", "m", 0.003);
}
std::string unit = "";
float reference = 0.0, difference = 0.0;
units_config->getUnit("Length", unit);
units_config->getReference("Length", reference);
units_config->getDifference("Length", difference);

// 単位系:Length 削除
units_config->removeUnitConfig("Length");
```

(CC API)

UdmError_t udm_config_getunit(

UdmHanler_t udm_handler,

const char* unit_name,

char* unit,

float* reference,

float* difference)

単位系情報を取得する.

[in] udm_handler UdmModel クラスポインタ

[in] unit_name 単位系名称

[out] unit 単位

[out] reference 基準値

[out] difference 差分値

戻り値 エラー番号 : UDM_OK | UDM_ERROR,etc

```
UdmError_t udm_config_setunit(  
    UdmHanler_t udm_handler,  
    const char* unit_name,  
    const char* unit,  
    float reference)
```

単位系を設定する（差分値を除く）.

差分値を除き、単位系を設定する.

同名の単位系が存在していた場合、上書きする.

同名の単位系が存在しない場合、追加する.

udm_handler UdmModel クラスポインタ

unit_name 単位系名称

unit 単位

reference 基準値

戻り値 エラー番号 : UDM_OK | UDM_ERROR,etc

```
UdmError_t udm_config_setunitwithdiff(  
    UdmHanler_t udm_handler,  
    const char* unit_name,  
    const char* unit,  
    float reference,  
    float difference)
```

単位系を設定する（差分値を含む）.

差分値を含めて、単位系を設定する.

同名の単位系が存在していた場合、上書きする.

同名の単位系が存在しない場合、追加する.

udm_handler UdmModel クラスポインタ

unit_name 単位系名称

unit 単位

reference 基準値

difference 差分値

戻り値 エラー番号 : UDM_OK | UDM_ERROR,etc

```
bool udm_config_existsunit(UdmHanler_t udm_handler, const char* unit_name)
```

単位系が存在するかチェックする.

udm_handler UdmModel クラスポインタ

unit_name 単位系名称

戻り値 true=単位系が存在する

```
void udm_config_removeunit(
```

UdmHanler_t udm_handler,

const char* unit_name)

単位系を削除する.

udm_handler UdmModel クラスポインタ

unit_name 単位系名称

使用例

```
UdmHanler_t model = udm_create_model();
```

```
char unit[128];
```

```
float reference = 0.0, difference = 0.0;
```

```
// Length
```

```
if (!udm_config_existsunit(model, "Length")) {
```

```
    udm_config_setunit(model, "Length", "m", 0.003);
```

```
}
```

```
udm_config_getunit(model, "Length", unit, &reference, &difference);
```

```
// 単位系:Length 削除
```

```
udm_config_removeunit(model, "Length");
```

4.2.6 CGNS ファイルの読込

CGNS ファイル名を指定して、CGNS ファイルを直接読込を行います。

(CXX API)

```
UdmError_t UdmModel::readCgns(
```

```
    const char* cgns_filename,
```

```
    int timeslice_step,
```

```
    const char* element_path = NULL)
```


CGNS ファイルを読みこむ.

cgns_filename CGNS ファイル名
timeslice_step CGNS 読込ステップ回数 (default=-1)
element_path CGNS 読込パス (default=NULL) : 未使用
戻り値 エラー番号 : UDM_OK | UDM_ERROR, etc.

使用例

```
char filename[256] = "cgns_haxa.cgns";  
UdmModel *model = new UdmModel();  
model->readCgns(filename);
```

(CC API)

```
UdmError_t udm_read_cgns(    UdmHanler_t udm_handler,  
                              const char* cgns_filename,  
                              int timeslice_step)
```

CGNS ファイルを読みこむ.

udm_handler UdmModel クラスポインタ
cgns_filename CGNS ファイル名
timeslice_step CGNS 読込ステップ番号 (-1 = 最初の時系列データ)
戻り値 エラー番号 : UDM_OK | UDM_ERROR, etc.

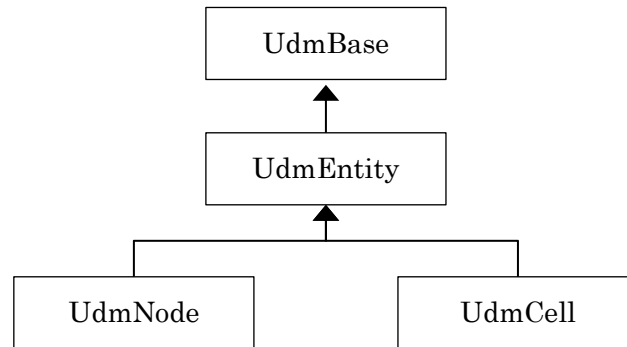
使用例

```
char filename[256] = "cgns_haxa.cgns";  
UdmHanler_t model = udm_create_model();  
ret = udm_read_cgns(model, filename, -1);
```

4.3 節点（ノード）

モデルを構成する節点（ノード）です。座標、及び節点（ノード）に定義された複数の物理量を持ちます。

節点（ノード）のクラス構成を以下に示します。



節点（ノード）クラス構成

(1) **UdmBase** : UDM ベースクラス

UDM ライブラリのすべてのクラスの基底クラスとなります。

(2) **UdmGeneral** : UDM モデル基本クラス

モデルの ID、name 等の基本情報のクラスとなります。

(3) **UdmEntity** : UDM 構成クラス

節点（ノード）、要素（セル）の基底クラスであり、物理量を管理するクラスとなります。

(4) **UdmNode** : UDM 節点(ノード)クラス

節点（ノード）のクラスとなります。

(5) **UdmCell** : UDM 要素（セル）クラス

要素（セル）のクラスとなります。

4.3.1 節点（ノード）の生成・追加

節点（ノード）を生成し、UDM ゾーンに追加します。

(CXX API)

```
template<class DATA_TYPE>
UdmSize_t UdmGridCoordinates::insertGridCoordinates(
    DATA_TYPE x,
    DATA_TYPE y,
```

DATA_TYPE z);	
節点（ノード）グリッド座標を追加する。 座標値は float, double のどちらでも設定可能です。	
x	グリッド座標 X : float or double
y	グリッド座標 Y : float or double
z	グリッド座標 Z : float or double
戻り値	節点（ノード）ID
<p>使用例</p> <pre> UdmZone *zone = model->createZone(); UdmGridCoordinates *grid = zone->getGridCoordinates(); float x = 0.0, y = 1.0, z = 1.5; UdmSize_t node_id = grid->insertGridCoordinates(x, y, z); </pre>	

(CC API)

UdmSize_t udm_insert_gridcoordinates(UdmHanler_t udm_handler, int zone_id, UdmReal_t x, UdmReal_t y, UdmReal_t z)	
節点（ノード）グリッド座標を追加する。 座標値はビルドの configure オプションの <code>-enable-real8</code> 指定時は double となります。 デフォルトは float です。	
udm_handler	UdmModel クラスポインタ
zone_id	ゾーン ID
x	グリッド座標 X
y	グリッド座標 Y
z	グリッド座標 Z
戻り値	節点（ノード）ID（1～）：0 の場合は追加エラー
<p>使用例</p> <pre> UdmHandler model = udm_create_model(); int zone_id = udm_create_zone(model); UdmReal_t x = 0.0, y = 1.0, z = 1.5; udm_insert_gridcoordinates(model, zone_id, x, y, z); </pre>	

4.3.2 節点（ノード）の取得

節点（ノード）クラスの取得を行います。UdmZone クラスと UdmGridCoordinates クラスから取得することができます。

C++のみの API となります。

(CXX API)

```
UdmSize_t UdmZone::getNumNodes() const;
```

グリッド構成節点（ノード）数を取得する.

戻り値 グリッド構成節点（ノード）数

```
UdmSize_t UdmGridCoordinates::getNumNodes() const
```

グリッド構成節点（ノード）数を取得する.

戻り値 グリッド構成節点（ノード）数

```
UdmNode* UdmZone::getNode(UdmSize_t node_id) const;
```

節点（ノード）を取得する.

node_id 節点（ノード）ID : 1～getNumNodes()

戻り値 節点（ノード）

```
UdmNode* UdmGridCoordinates::getNodeById(UdmSize_t node_id) const;
```

節点（ノード）を取得する.

node_id 節点（ノード）ID : 1～getNumNodes()

戻り値 節点（ノード）

使用例

```
UdmZone *zone = model->createZone();
UdmGridCoordinates *grid = zone->getGridCoordinates();
for (UdmSize_t node_id=1; node_id<=grid->getNumNodes(); node_id++) {
    UdmNode *node = grid->getNodeById(node_id);
}
```

(CC API)

```
UdmSize_t udm_getnum_nodes(UdmHanler_t udm_handler, int zone_id);
```

グリッド構成ノード数を取得する.

udm_handler UdmModel クラスポインタ

zone_id ゾーン ID (1 ~)

戻り値 グリッド構成ノード数

使用例

```
UdmZone *zone = model->createZone();
```

```
int zone_id = 1;
```

```
int num_nodes = udm_getnum_nodes(model, zone_id);
```

4.3.3 節点（ノード）の構成座標

節点（ノード）の座標値の取得、設定を行います。

(CXX API)

```
UdmError_t UdmNode::getCoords(float& x, float& y, float& z) const;
```

```
UdmError_t UdmNode::getCoords(double& x, double& y, double& z) const;
```

ノード座標を取得する

[out] x X 座標

[out] y Y 座標

[out] z Z 座標

戻り値 エラー番号 : UDM_OK | UDM_ERROR

```
UdmError_t UdmNode::setCoords(float x, float y, float z);
```

```
UdmError_t UdmNode::setCoords(double x, double y, double z)
```

ノード座標を設定する

x X 座標

y Y 座標

z Z 座標

戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```
UdmZone *zone = model->createZone();
```

```
UdmGridCoordinates *grid = zone->getGridCoordinates();
```

```

for (UdmSize_t node_id=1; node_id<=grid->getNumNodes(); node_id++) {
    UdmNode *node = grid->getNodeById(node_id);
    float x = 0.0, y = 1.0, z = 1.5;
    node->setCoords(x, y, z);
    node->getCoords(x, y, z);
}

```

(CC API)

```

UdmError_t udm_get_gridcoordinates(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t node_id,
    UdmReal_t* x,
    UdmReal_t* y,
    UdmReal_t* z)

```

節点（ノード）座標を取得する.

[in] udm_handler UdmModel クラスポインタ

[in] zone_id ゾーン ID（1～）

[in] node_id 節点（ノード）ID（1～）

[out] x X 座標

[out] y Y 座標

[out] z Z 座標

戻り値 エラー番号：UDM_OK | UDM_ERROR

```

UdmError_t udm_set_gridcoordinates(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t node_id,
    UdmReal_t x,
    UdmReal_t y,
    UdmReal_t z)

```

節点（ノード）座標を設定する.

udm_handler UdmModel クラスポインタ

zone_id ゾーン ID（1～）

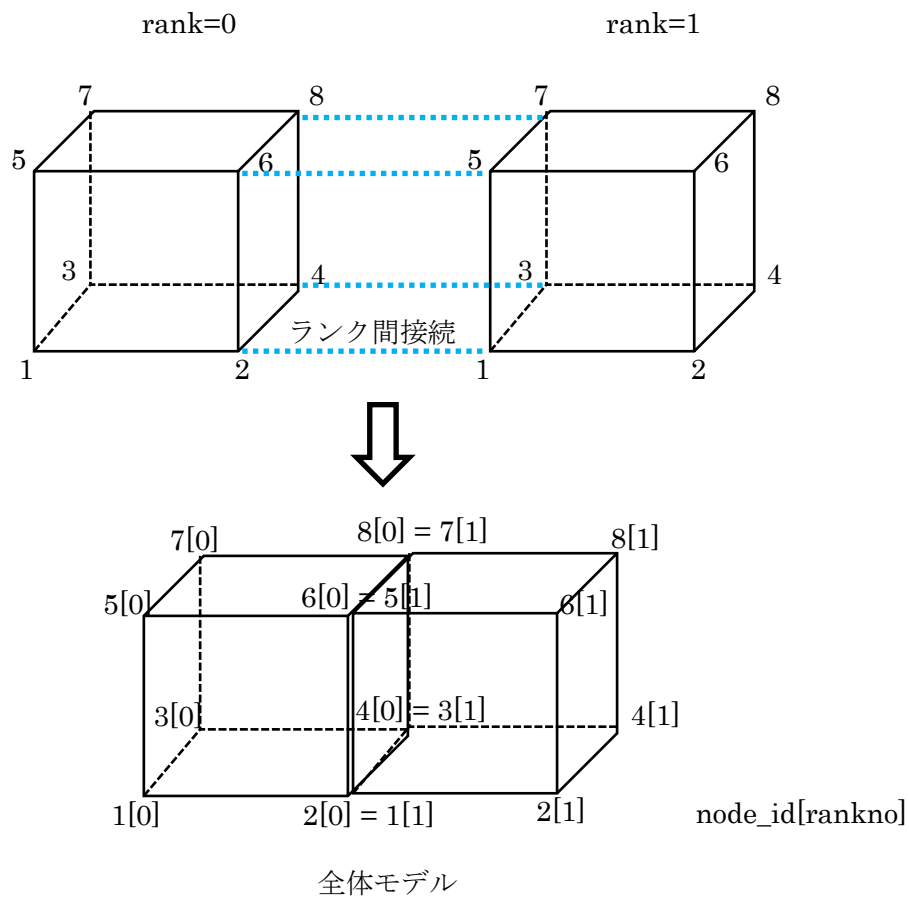
node_id	節点（ノード）ID（1～）
x	X 座標
y	Y 座標
z	Z 座標
戻り値	エラー番号：UDM_OK UDM_ERROR

使用例

```
UdmHandler model = udm_create_model();
int zone_id = 1;
int num_nodes = udm_getnum_nodes(model, zone_id);
for (UdmSize_t node_id=1; node_id<=num_nodes; node_id++) {
    float x = 0.0, y = 1.0, z = 1.5;
    udm_set_gridcoordinates(model, zone_id, x, y, z);
    udm_get_gridcoordinates(model, zone_id, x, y, z);
}
```

4.3.4 ランク間接続節点（ノード）

MPI ランク間の共通節点（ノード）を設定します。ランク間接続節点（ノード）が内部境界となり、仮要素（セル）の作成、分割に使用します。



(CXX API)

```
UdmError_t UdmGridCoordinates::insertRankConnectivity(
    UdmSize_t node_id,
    int rankno,
    UdmSize_t localid)
```

内部境界情報を節点（ノード）に追加する.

node_id 節点（ノード）ID（1～）

rankno 接続先 MPI ランク番号（0～）

localid 接続先節点（ノード）ID（1～）

戻り値 エラー番号：UDM_OK | UDM_ERROR

使用例

```
UdmGridCoordinates *grid = zone->getGridCoordinates();
if (myeank==0) {
```



```

        grid->insertRankConnectivity(2, 1, 1);
        grid->insertRankConnectivity(4, 1, 3);
        grid->insertRankConnectivity(6, 1, 5);
        grid->insertRankConnectivity(8, 1, 7);
    }
    else if (myeank==1) {
        grid->insertRankConnectivity(1, 0, 2);
        grid->insertRankConnectivity(3, 0, 4);
        grid->insertRankConnectivity(5, 0, 6);
        grid->insertRankConnectivity(7, 0, 8);
    }

```

(CC API)

```

UdmError_t udm_insert_rankconnectivity(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t node_id,
    int rankno,
    UdmSize_t localid)

```

内部境界情報を節点（ノード）に追加する.

udm_handler UdmModel クラスポインタ

zone_id ゾーン ID

node_id 節点（ノード）ID（1～）

rankno 接続先 MPI ランク番号（0～）

localid 接続先節点（ノード）ID（1～）

戻り値 エラー番号：UDM_OK | UDM_ERROR

使用例

```

UdmHandler model = udm_create_model();
int zone_id = 1;
if (myeank==0) {
    udm_insert_rankconnectivity(model, zone_id,2, 1, 1);
    udm_insert_rankconnectivity(model, zone_id,4, 1, 3);
    udm_insert_rankconnectivity(model, zone_id,6, 1, 5);
    udm_insert_rankconnectivity(model, zone_id,8, 1, 7);
}

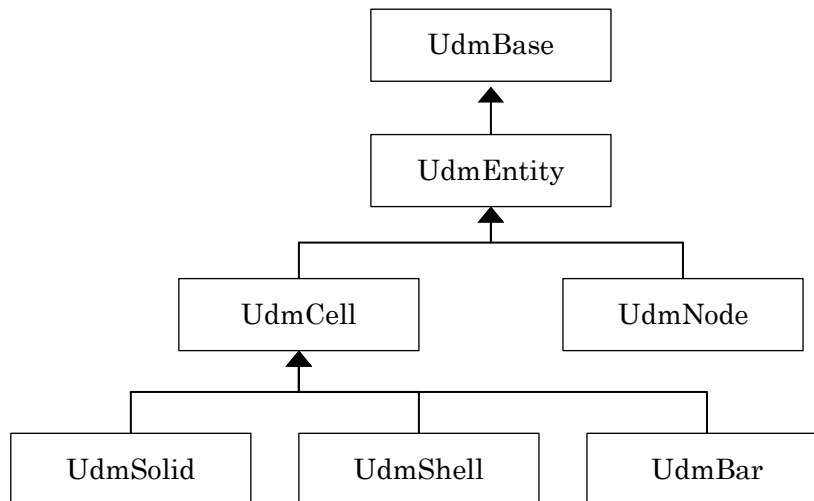
```

```
}  
else if (myeank==1) {  
    udm_insert_rankconnectivity(model, zone_id,1, 0, 2 );  
    udm_insert_rankconnectivity(model, zone_id,3, 0, 4);  
    udm_insert_rankconnectivity(model, zone_id,5, 0, 6);  
    udm_insert_rankconnectivity(model, zone_id,7, 0, 8);  
}
```

4.4 要素（セル）

モデルを構成する要素（セル）です。要素（セル）を構成する節点（ノード）、形状、要素（セル）に定義された複数の物理量を持ちます。

要素（セル）のクラス構成を以下に示します。



要素（セル）クラス構成

- (1) **UdmBase** : UDM ベースクラス

UDM ライブラリのすべてのクラスの基底クラスとなります。

- (2) **UdmGeneral** : UDM モデル基本クラス

モデルの ID、name 等の基本情報のクラスとなります。

- (3) **UdmEntity** : UDM 構成クラス

節点（ノード）、要素（セル）の基底クラスであり、物理量を管理するクラスとなります。

- (4) **UdmNode** : UDM 節点(ノード)クラス

節点（ノード）のクラスとなります。

- (5) **UdmCell** : UDM 要素（セル）クラス

要素（セル）のクラスとなります。

- (6) **UdmSolid** : UDM3D 要素（セル）クラス

3D 要素（セル）のクラスとなります。

以下の 3D 形状をサポートします。

3D 要素形状	UdmElementType_t
四面体要素	Udm_TETRA_4
ピラミッド要素	Udm_PYRA_5
五面体要素	Udm_PENTA_6

六面体要素	Udm_HEX8_8
-------	------------

サポート 3D 要素（セル）形状

(7) UdmSolid : UDM2D 要素（セル）クラス

2D 要素（セル）のクラスとなります。

以下の 2D 形状を定義していますが、ユーザプログラムでは使用しません。

2D 要素形状	UdmElementType_t
三角形要素	Udm_TRI_3
四角形要素	Udm_QUAD_4

定義 2D 要素（セル）形状

(8) UdmBar : UDM1D 要素（セル）クラス

1D 要素（セル）のクラスとなります。

以下の 1D 形状を定義していますが、ユーザプログラムでは使用しません。

1D 要素形状	UdmElementType_t
BAR 要素	Udm_BAR_2

定義 1D 要素（セル）形状

4.4.1 要素（セル）の生成・追加

要素（セル）を生成し、UDM セクションに追加します。

(CXX API)

<pre>UdmSize_t UdmElements::insertCellConnectivity(UdmElementType_t elem_type, UdmSize_t* node_ids);</pre> <p>要素（セル）をセクション（要素構成）に追加する.</p> <p>elem_type 要素タイプ</p> <p>node_ids 要素接続情報（ノードリスト）</p> <p>戻り値 要素（セル）ID（1～）：0 の場合は挿入エラー</p>
<p>使用例</p> <pre>UdmSections *sections = zone->getSections(); UdmElements *elements = sections->createSection(Udm_HEX8_8); UdmSize_t elem_nodes[8] = {1, 2, 8, 9, 65, 66, 72, 73}; // 要素の追加</pre>

```
elements->insertCellConnectivity(Udm_HEXA_8, elem_nodes);
```

(CC API)

```
UdmSize_t udm_insert_cellconnectivity(  
    UdmHanler_t udm_handler,  
    int zone_id,  
    UdmElementType_t elem_type,  
    UdmSize_t* node_ids)
```

要素（セル）をセクション（要素構成）に追加する.

udm_handler UdmModel クラスポインタ

zone_id ゾーン ID

elem_type 要素タイプ

node_ids 要素接続情報（ノードリスト）

戻り値 要素（セル）ID（1～）：0の場合は挿入エラー

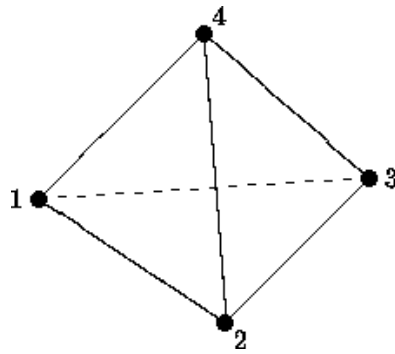
使用例

```
UdmHandler model = udm_create_model();  
int zone_id = udm_create_zone(model);  
int element_id = udm_create_section(model, zone_id, Udm_HEXA_8);  
UdmSize_t elem_nodes[8] = {1, 2, 8, 9, 65, 66, 72, 73};  
// 要素の追加  
UdmSize_t cell_id = udm_insert_cellconnectivity(  
    model, zone_id, Udm_HEXA_8, elem_nodes);
```

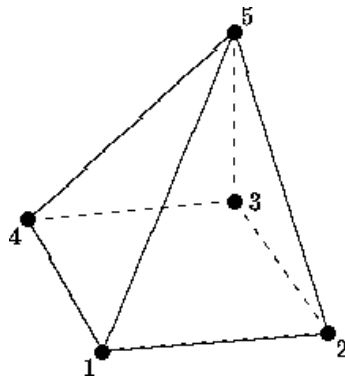
要素接続情報（ノードリスト）は節点（ノード）ID のリストであり、要素タイプと同じ節点（ノード）数を定義しなければなりません。

また、節点（ノード）ID の設定順番は、次の順番に従わなければなりません。

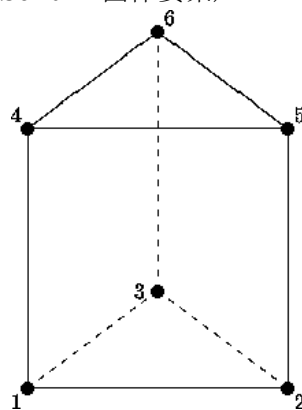
(Udm_TETRA_4 :: Solid:四面体要素)



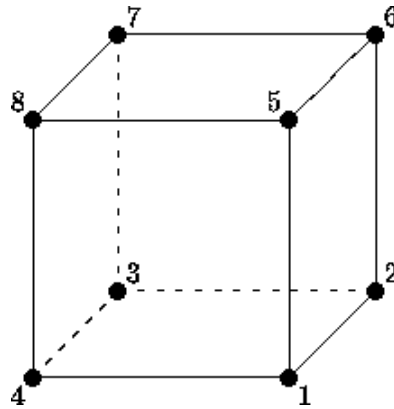
(Udm_PYRA_5 :: Solid:ピラミッド要素)



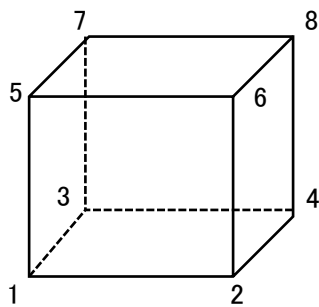
(Udm_PENTA_6 :: Solid:五面体要素)



(Udm_HEXA_8 :: Solid:六面体要素)



(設定例)



```
UdmSize_t elem_nodes[8] =  
    { 1, 2, 4, 3, 5, 6, 8, 7};
```

4.4.2 要素（セル）の取得

要素（セル）クラスの取得を行います。

UdmZone クラスと UdmElements クラスから取得することができます。

UdmZone クラスはすべての要素（セル）、UdmElements クラスはそのセクションのみの要素（セル）を取得します。

C++のみの API となります。

(CXX API)

```
UdmSize_t UdmZone::getNumCells() const
```

要素（セル）数を取得する.

戻り値 要素（セル）数

```
UdmCell* UdmZone::getCell(UdmSize_t cell_id) const;
```

要素（セル）を取得する.

cell_id 要素（セル）ID : 1~getNumCells()

戻り値 要素（セル）

UdmSize_t UdmElements::getNumCells() const

セクション（要素構成）の要素（セル）数を取得する.

戻り値 要素（セル）数

UdmCell* UdmElements::getCell(UdmSize_t cell_id) const;

セル ID の要素（セル）を取得する.

cell_id セクション内ローカルセル ID（1～）

戻り値 要素（セル）

使用例

```
int zone_id = 1;
int element_id = 1;
UdmZone *zone = model->cgetZone(zone_id);
UdmElements *elements = zone->getSections()->getSection(section_id);
for (UdmSize_t cell_id=1; cell_id<=elements->getNumCells(); cell_id++) {
    UdmCell *cell = grid->getCell(cell_id);
}
```

(CC API)

UdmSize_t udm_getnum_cells(UdmHanler_t udm_handler, int zone_id);

セクション（要素構成）の要素（セル）数を取得する.

udm_handler UdmModel クラスポインタ

zone_id ゾーン ID（1～）

戻り値 要素（セル）数

使用例

```
int zone_id = 1;
int num_cells = udm_getnum_cells(model, zone_id);
```


4.4.3 構成節点（ノード）情報

要素（セル）を構成する節点（ノード）を取得します。

(CXX API)

<code>UdmSize_t UdmCell::getNumNodes() const</code> 構成ノード（節点）数を取得する. 戻り値 構成ノード（節点）数
<code>UdmNode* UdmCell::getNode(UdmSize_t node_id) const</code> 構成ノード（節点）を取得する. <code>node_id</code> 構成ノード ID（1 ～） 戻り値 構成ノード（節点）
使用例 <pre>UdmSize_t cell_size = zone->getNumCells(); for (n=1; n<=cell_size; n++) { UdmCell *cell = zone->getCell(n); node_size = cell->getNumNodes(); for (m=1; m<=node_size; m++) { UdmNode *node = cell->getNode(m); node->getCoords(x, y, z); } }</pre>

(CC API)

<code>int udm_get_cellconnectivity(</code> <code>UdmHanler_t udm_handler,</code> <code>int zone_id,</code> <code>UdmSize_t cell_id,</code> <code>UdmElementType_t* elem_type,</code> <code>UdmSize_t* node_ids,</code> <code>int* num_nodes)</code> 要素（セル）の接続情報を取得する. [in] <code>udm_handler</code> UdmModel クラスポインタ [in] <code>zone_id</code> ゾーン ID（1 ～）

[in] cell_id	要素（セル）ID（1～）
[out] elem_type	要素（セル）形状タイプ
[out] node_ids	接続節点（ノード）ID リスト
[out] num_nodes	接続節点（ノード）数
戻り値	接続節点（ノード）数

使用例

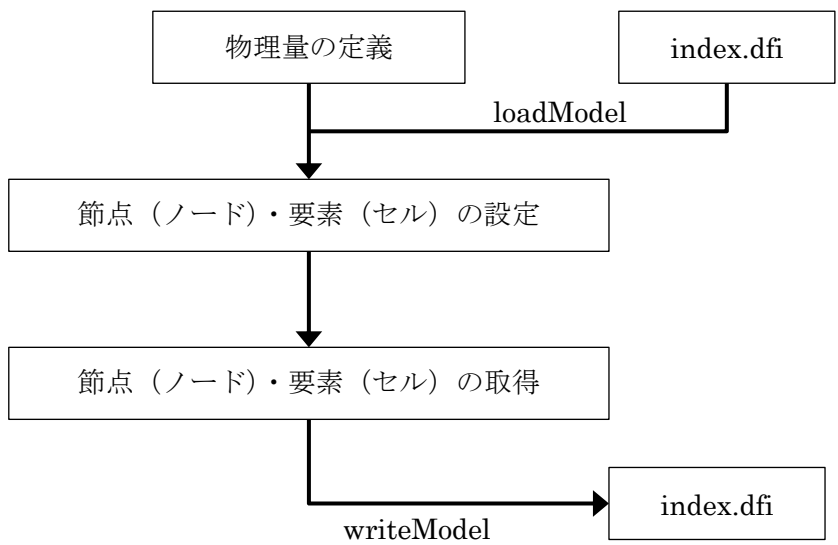
```

UdmSize_t cell_size = udm_getnum_cells(model, zone_id);
for (UdmSize_t cell_id=1; cell_id<=cell_size; cell_id++) {
    UdmElementType_t elem_type;
    UdmSize_t node_ids[8];
    unsigned int num_nodes;
    udm_get_cellconnectivity(model, zone_id, cell_id,
                             &elem_type, node_ids, &num_nodes);
}

```

4.5 節点（ノード）・要素（セル）の物理量

節点（ノード）・要素（セル）に物理量を設定するには、DFI ファイルに記述又はプログラムから DFI 定義クラス（UdmDfiConfig）に設定する必要があります。



物理量の使用の流れ

- a) loadModel(後述)により、index.dfi から物理量の定義を読み込みます。
- b) 新規・追加の物理量がある場合は、プログラムから定義を行います。
- c) 節点（ノード）・要素（セル）に物理量の値を設定します。
- d) 節点（ノード）・要素（セル）から物理量の値を取得します。
- e) 新規・追加の物理量の定義は writeModel により index.dfi に書き込まれます。

物理量の値は、物理量定義されたデータ型によって節点（ノード）・要素（セル）に設定します。

CXX API の場合、物理量の取得、設定を行う物理量の値のデータ型は、int, long long, float, double となります。

CC API の場合、整数値、実数値のデータ型は UdmInteger_t, UdmReal_t 型となります。これは、ビルド時の configure オプションによって、int, long long、又は float, double に定義されます。

configure オプション	データ型	データサイズ
実数型		
--enable-real8	double	8

(未設定：デフォルト)	float	4
整数型		
--enable-int8	long long	8
(未設定：デフォルト)	int	4

4.5.1 物理量定義

DFI 定義クラス (UdmDfiConfig) に物理量の情報を設定します。

index.dfi に物理量定義が記述されている場合は、loadModel(後述)により自動的に設定されます。

物理量を節点 (ノード)・要素 (セル) に設定する為には、DFI 定義クラス (UdmDfiConfig) への物理量定義が必要です。

(1) 物理量定義の作成

DFI 定義クラス (UdmDfiConfig) に物理量の情報を作成します。

(CXX API)

```
UdmError_t UdmFlowSolutionListConfig::setSolutionFieldInfo(
```

```
    const std::string      &solution_name,
    UdmGridLocation_t      grid_location,
    UdmDataType_t          data_type,
    UdmVectorType_t        vector_type,
    int                    nvector_size,
    bool                    constant_flag);
```

物理量情報を設定する.

物理量変数名称が存在している場合は、上書きする.

存在していない場合は、追加する.

solution_name 物理量変数名称

grid_location 物理量の定義位置

data_type データ型

vector_type ベクトル型

nvector_size N 成分数

constant_flag 物理データ固定値フラグ デフォルト=false

戻り値 エラー番号 : UDM_OK | UDM_ERROR

```
UdmError_t UdmFlowSolutionListConfig::setSolutionFieldInfo(
```

```
const std::string& solution_name,
UdmGridLocation_t grid_location,
UdmDataType_t data_type);
```

物理量情報を設定する:基本情報.

物理量変数名称が存在している場合は、上書きする.

存在していない場合は、追加する.

ベクトル型=Udm_Scalar, 初期設定値=0.0, 固定値フラグ=false にて設定する.

solution_name 物理量変数名称

grid_location 物理量の定義位置

data_type データ型

戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```
UdmModel *model = new UdmModel();
UdmDfiConfig* config = model->getDfiConfig();
UdmFlowSolutionListConfig* solutions = config->getFlowSolutionListConfig();
solutions->setSolutionFieldInfo(
    "Pressure", Udm_Vertex, Udm_RealSingle);
solutions->setSolutionFieldInfo(
    "Velocity",
    Udm_Vertex,
    Udm_RealSingle,
    Udm_Vector,
    3);
```

(CC API)

```
UdmError_t udm_config_setsolution(
    UdmHanler_t udm_handler,
    const char* solution_name,
    UdmGridLocation_t grid_location,
    UdmDataType_t data_type,
    UdmVectorType_t vector_type,
    int nvector_size,
    bool constant_flag);
```

物理量情報を設定する.

物理量変数名称が存在している場合は、上書きする。
存在していない場合は、追加する。

udm_handler	UdmModel クラスポインタ
solution_name	物理量変数名称
grid_location	物理量の定義位置
data_type	データ型
vector_type	ベクトル型
nvector_size	N 成分数
constant_flag	物理データ固定値フラグ
戻り値	エラー番号 : UDM_OK UDM_ERROR

```
UdmError_t udm_config_setscalarsolution(  
    UdmHanler_t udm_handler,  
    const char *solution_name,  
    UdmGridLocation_t grid_location,  
    UdmDataType_t data_type);
```

物理量情報を設定する:基本情報.

物理量変数名称が存在している場合は、上書きする.

存在していない場合は、追加する.

ベクトル型=Udm_Scalar, 初期設定値=0.0, 固定値フラグ=false にて設定する.

udm_handler	UdmModel クラスポインタ
solution_name	物理量変数名称
grid_location	物理量の定義位置
data_type	データ型
戻り値	エラー番号 : UDM_OK UDM_ERROR

使用例

```
UdmHanler_t model = udm_create_model();  
// 物理量定義の設定  
udm_config_setscalarsolution(  
    model,  
    "Pressure",  
    Udm_Vertex,  
    Udm_RealSingle);  
udm_config_setsolution(  
    model,
```

```

        "Velocity",
        Udm_Vertex,
        Udm_RealSingle,
        Udm_Vector,
        3,
        false);

```

(2) 物理量定義の取得

DFI 定義クラス (UdmDfiConfig) から物理量の情報を取得します。

(CXX API)

```

UdmError_t UdmFlowSolutionListConfig::getSolutionFieldInfo(
    const std::string      &solution_name,
    UdmGridLocation_t      &grid_location,
    UdmDataType_t          &data_type,
    UdmVectorType_t        &vector_type,
    int                    &nvector_size,
    bool                   &constant_flag) const;

```

物理量情報を取得する。

物理量変数名称が存在していない場合は、UDM_ERROR を返す。

[in]	solution_name	物理量変数名称
[out]	grid_location	物理量の定義位置
[out]	data_type	データ型
[out]	vector_type	ベクトル型
[out]	nvector_size	N 成分数
[out]	constant_flag	物理データ固定値フラグ
戻り値	エラー番号 : UDM_OK UDM_ERROR	

使用例

```

UdmModel *model = new UdmModel();
UdmDfiConfig* config = model->getDfiConfig();
UdmFlowSolutionListConfig* solutions = config->getFlowSolutionListConfig();
UdmGridLocation_t      grid_location;
UdmDataType_t          data_type;

```

```

UdmVectorType_t      vector_type;
int                   nvector_size;
bool                  constant_flag;
solutions->getSolutionFieldInfo(
    "Velocity",
    grid_location,
    data_type,
    vector_type,
    nvector_size,
    constant_flag);

```

(CC API)

```

UdmError_t udm_config_getsolution(
    UdmHanler_t udm_handler,
    const char* solution_name,
    UdmGridLocation_t* grid_location,
    UdmDataType_t* data_type,
    UdmVectorType_t* vector_type,
    int* nvector_size,
    bool* constant_flag);

```

物理量情報を取得する.

物理量変数名称が存在していない場合は、UDM_ERROR を返す.

[in]	udm_handler	UdmModel クラスポインタ
[in]	solution_name	物理量変数名称
[out]	grid_location	物理量の定義位置
[out]	data_type	データ型
[out]	vector_type	ベクトル型
[out]	nvector_size	N 成分数
[out]	constant_flag	物理データ固定値フラグ
戻り値	エラー番号 : UDM_OK UDM_ERROR	

使用例

```

UdmHanler_t model = udm_create_model();
// 物理量定義の取得
UdmGridLocation_t grid_location;

```



```

UdmDataType_t      data_type;
UdmVectorType_t    vector_type;
int                 nvector_size;
bool                constant_flag;
udm_config_getsolution(
                    model,
                    "Velocity",
                    &grid_location,
                    &data_type,
                    &vector_type,
                    &nvector_size,
                    &constant_flag);

```

(3) 物理量定義の存在チェック

DFI 定義クラス (UdmDfiConfig) に物理量が定義されているかチェックします。

(CXX API)

```

bool UdmFlowSolutionListConfig::existsSolutionConfig(
    const std::string &solution_name) const

```

物理量変数名称の物理量情報が存在するかチェックする.

solution_name 物理量変数名称

戻り値 true=物理量情報が存在する

使用例

```

UdmModel *model = new UdmModel();
UdmDfiConfig* config = model->getDfiConfig();
UdmFlowSolutionListConfig* solutions = config->getFlowSolutionListConfig();
if (solutions->existsSolutionConfig("Pressure")) {
    solutions->setSolutionFieldInfo(
        "Pressure", Udm_Vertex, Udm_RealSingle);
}

```

(CC API)

```

bool udm_config_existssolution(

```

<pre> UdmHanler_t udm_handler, const char *solution_name) 物理量変数名称の物理量情報が存在するかチェックする. udm_handler UdmModel クラスポインタ solution_name 物理量変数名称 戻り値 true=物理量情報が存在する </pre>
<p>使用例</p> <pre> UdmHanler_t model = udm_create_model(); if (!udm_config_exists(solution(model, "Pressure"))) { udm_config_setscalar(solution(model, "Pressure", Udm_Vertex, Udm_RealSingle); } </pre>

(4) 物理量定義の削除

DFI 定義クラス (UdmDfiConfig) から物理量定義を削除します。

(CXX API)

<pre> UdmError_t UdmFlowSolutionListConfig::removeSolutionConfig(const std::string &solution_name) 物理量変数名称の物理量情報を削除する. solution_name 物理量変数名称 戻り値 エラー番号 : UDM_OK UDM_ERROR </pre>
<p>使用例</p> <pre> UdmModel *model = new UdmModel(); UdmDfiConfig* config = model->getDfiConfig(); UdmFlowSolutionListConfig* solutions = config->getFlowSolutionListConfig(); solutions->removeSolutionFieldInfo("Pressure"); </pre>

(CC API)

```
void udm_config_removesolution(  
    UdmHanler_t udm_handler,  
    const char* solution_name)  
    物理量変数名称の物理量情報を削除する.  
    udm_handler          UdmModel クラスポインタ  
    solution_name        物理量変数名称  
    戻り値              なし
```

使用例

```
UdmHanler_t model = udm_create_model();  
udm_config_removesolution(model, "Pressure");
```

4.5.2 節点（ノード）の物理量の取得

節点（ノード）から設定された物理量の値を取得します。

(CXX API)

```
unsigned int UdmEntity::getNumSolutionValue(  
    const std::string& solution_name) const  
    物理量データのデータ値数を取得する.  
    Scalar データの場合は 1, Vector データの場合は 3, 又は定義成分数を返す.  
    solution_name        物理量データ名称  
    戻り値              データ値数
```

```
template<class VALUE_TYPE>  
UdmError_t UdmEntity::getSolutionScalar(  
    const std::string& solution_name,  
    VALUE_TYPE& value) const  
    物理量データ値を取得する.  
    [in] solution_name    物理量データ名称  
    [out] value           取得物理データ値  
    戻り値              エラー番号 : UDM_OK | UDM_ERROR
```

```
template<class VALUE_TYPE>
```

```

unsigned int UdmEntity::getSolutionVector(
    const std::string& solution_name,
    VALUE_TYPE* values) const

```

物理量データ値リストを取得する.

[in] solution_name 物理量データ名称

[out] values 取得物理データ値リスト

戻り値 取得データ数

使用例

```

UdmZone *zone = model->getZone();
for (n=1; n<=zone->getNumNodes(); n++) {
    UdmNode *node = zone->getNode(n);
    node->getSolutionScalar("Pressure", pressure);
    float motions[3] = {0.0, 0.0, 0.0};
    node->getSolutionVector("Motion", motions);
}

```

(CC API)

```

UdmError_t udm_get_nodesolution_integer(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t node_id,
    const char* solution_name,
    UdmInteger_t* value);

```

節点（ノード）の物理量データ値を取得する:integer(スカラデータ).

[in] udm_handler UdmModel クラスポインタ

[in] zone_id ゾーン ID（1～）

[in] node_id 節点（ノード）ID（1～）

[in] solution_name 物理量データ名称

[out] value 取得物理データ値:integer(スカラデータ)

戻り値 エラー番号：UDM_OK | UDM_ERROR

```

UdmError_t udm_get_nodesolution_real(
    UdmHanler_t udm_handler,
    int zone_id,

```

```

        UdmSize_t node_id,
        const char* solution_name,
        UdmReal_t* value);

```

節点（ノード）の物理量データ値を取得する:real(スカラデータ).

```

[in] udm_handler      UdmModel クラスポインタ
[in] zone_id          ゾーン ID（1～）
[in] node_id          節点（ノード）ID（1～）
[in] solution_name    物理量データ名称
[out] value           取得物理データ値:real(スカラデータ)
戻り値               エラー番号：UDM_OK | UDM_ERROR

```

```

UdmError_t udm_get_nodesolutions_integer(

```

```

        UdmHanler_t udm_handler,
        int zone_id,
        UdmSize_t node_id,
        const char* solution_name,
        UdmInteger_t *values,
        int* size);

```

節点（ノード）の物理量データ値を取得する:integer（ベクトルデータ）

```

[in] udm_handler      UdmModel クラスポインタ
[in] zone_id          ゾーン ID（1～）
[in] node_id          節点（ノード）ID（1～）
[in] solution_name    物理量データ名称
[out] values          取得物理データ値
[out] size            取得物理データ数
戻り値               エラー番号：UDM_OK | UDM_ERROR

```

```

UdmError_t udm_get_nodesolutions_real(

```

```

        UdmHanler_t udm_handler,
        int zone_id,
        UdmSize_t node_id,
        const char* solution_name,
        UdmReal_t* values,
        int* size)

```

節点（ノード）の物理量データ値を取得する:real.（ベクトルデータ）

```

[in] udm_handler      UdmModel クラスポインタ

```

[in] zone_id	ゾーン ID (1 ~)
[in] node_id	節点 (ノード) ID (1 ~)
[in] solution_name	物理量データ名称
[out] values	取得物理データ値
[out] size	取得物理データ数
戻り値	エラー番号 : UDM_OK UDM_ERROR

使用例

```
UdmHanler_t model = udm_create_model();
int zone_id = 1;
UdmSize_t num_nodes = udm_getnum_nodes(model, zone_id);
for (int n=1; n<=num_nodes; n++) {
    // 物理量の取得
    float pressure = 0.0;          // Pressure が未設定の場合は、値は返ってこない
    udm_get_nodesolution_real(model, zone_id, n, "Pressure", &pressure);
    UdmReal_t motions[3] = {0.0, 0.0, 0.0};
    int size = 0;
    udm_get_nodesolutions_real(model, zone_id, n, "Motion", motions, &size);
}
```

4.5.2 節点 (ノード) への物理量の設定

節点 (ノード) に物理量の値を設定します。

(CXX API)

```
template<class VALUE_TYPE>
UdmError_t UdmEntity::setSolutionScalar(
    const std::string& solution_name,
    VALUE_TYPE value)
    物理量データ値を設定する.
    物理量データ名称が存在しない場合は、追加する.
    solution_name    物理量データ名称
    value            物理データ値
    戻り値            エラー番号 : UDM_OK | UDM_ERROR
```

```
template<class VALUE_TYPE>
```

```
UdmError_t UdmEntity::setSolutionVector(  
    const std::string& solution_name,  
    const VALUE_TYPE* values,  
    unsigned int size = 3);
```

物理量データ値リストを設定する.

solution_name 物理量データ名称

values 物理量データ値リスト

size 物理量データ数 (default = 3)

戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```
UdmZone *zone = model->getZone();  
for (n=1; n<=zone->getNumNodes(); n++) {  
    UdmNode *node = zone->getNode(n);  
    float pressure = 1.0;  
    node->setSolutionScalar("Pressure", pressure);  
    float motions[3] = {0.0, 0.0, 0.0};  
    node->setSolutionVector("Motion", motions);  
}
```

(CC API)

```
UdmError_t udm_set_nodesolution_integer(  
    UdmHanler_t udm_handler,  
    int zone_id,  
    UdmSize_t node_id,  
    const char* solution_name,  
    UdmInteger_t value);
```

節点 (ノード) の物理量データ値を設定する:integer(スカラデータ).

udm_handler UdmModel クラスポインタ

zone_id ゾーン ID (1 ~)

node_id 節点 (ノード) ID (1 ~)

solution_name 物理量データ名称

value 物理データ値:integer(スカラデータ)

戻り値 エラー番号 : UDM_OK | UDM_ERROR

```
UdmError_t udm_set_nodesolution_real(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t node_id,
    const char* solution_name,
    UdmReal_t value);
```

節点（ノード）の物理量データ値を設定する:real(スカラデータ).

udm_handler UdmModel クラスポインタ
zone_id ゾーン ID（1～）
node_id 節点（ノード）ID（1～）
solution_name 物理量データ名称
value 物理データ値:real(スカラデータ)
戻り値 エラー番号：UDM_OK | UDM_ERROR

```
UdmError_t udm_set_nodesolutions_integer(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t node_id,
    const char* solution_name,
    const UdmInteger_t *values,
    int size);
```

節点（ノード）に物理量データ値リストを設定する:integer(ベクトル).

udm_handler UdmModel クラスポインタ
zone_id ゾーン ID（1～）
node_id 節点（ノード）ID（1～）
solution_name 物理量データ名称
values 物理量データ値リスト:integer
size 物理量データ数（default = 3）
戻り値 エラー番号：UDM_OK | UDM_ERROR

```
UdmError_t udm_set_nodesolutions_real(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t node_id,
    const char* solution_name,
```


	<pre> const UdmReal_t *values, int size); </pre> <p>節点（ノード）に物理量データ値を設定する:real(ベクトル).</p> <p>udm_handler UdmModel クラスポインタ</p> <p>zone_id ゾーン ID（1～）</p> <p>node_id 節点（ノード）ID（1～）</p> <p>solution_name 物理量データ名称</p> <p>values 物理量データ値リスト:real</p> <p>size 物理量データ数（default = 3）</p> <p>戻り値 エラー番号：UDM_OK UDM_ERROR</p>
使用例	<pre> UdmHanler_t model = udm_create_model(); int zone_id = 1; UdmSize_t num_nodes = udm_getnum_nodes(model, zone_id); for (int n=1; n<=num_nodes; n++) { float pressure = 0.0; udm_set_nodesolution_real(model, zone_id, n, "Pressure", pressure); UdmReal_t motions[3] = {0.0, 0.0, 0.0}; int size = 0; udm_get_nodesolutions_real(model, zone_id, n, "Motion", motions, 3); } </pre>

4.5.3 要素（セル）の物理量の取得

要素（セル）から設定された物理量の値を取得します。

(CXX API)

	<pre> unsigned int UdmEntity::getNumSolutionValue(const std::string& solution_name) const </pre> <p>物理量データのデータ値数を取得する.</p> <p>Scalar データの場合は 1, Vector データの場合は 3, 又は定義成分数を返す.</p> <p>solution_name 物理量データ名称</p> <p>戻り値 データ値数</p>
--	---

```
template<class VALUE_TYPE>
UdmError_t UdmEntity::getSolutionScalar(
    const std::string& solution_name,
    VALUE_TYPE& value) const
```

物理量データ値を取得する.

[in] solution_name 物理量データ名称

[out] value 取得物理データ値

戻り値 エラー番号 : UDM_OK | UDM_ERROR

```
template<class VALUE_TYPE>
unsigned int UdmEntity::getSolutionVector(
    const std::string& solution_name,
    VALUE_TYPE* values) const
```

物理量データ値リストを取得する.

[in] solution_name 物理量データ名称

[out] values 取得物理データ値リスト

戻り値 取得データ数

使用例

```
UdmZone *zone = model->getZone();
UdmElements *elements = zone->getSections()->getSection(section_id);
for (UdmSize_t cell_id=1; cell_id<=elements->getNumCells(); cell_id++) {
    UdmCell *cell = elements->getCell(cell_id);
    float temprature = 0.0;
    cell->getSolutionScalar("Temprature", &temprature );
    float velocity[3] = {0.0, 0.0, 0.0};
    cell->getSolutionVector("Velocity", velocity);
}
```

(CC API)

```
UdmError_t udm_get_cellsolution_integer(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t cell_id,
    const char* solution_name,
```

```

        UdmInteger_t * value);
要素（セル）の物理量データ値を取得する:integer(スカラデータ).
[in] udm_handler          UdmModel クラスポインタ
[in] zone_id              ゾーン ID（1～）
[in] cell_id              要素（セル）ID（1～）
[in] solution_name        物理量データ名称
[out] value               取得物理データ値:integer(スカラデータ)
戻り値                   エラー番号：UDM_OK | UDM_ERROR

```

```

UdmError_t udm_get_cellsolution_real(
        UdmHanler_t udm_handler,
        int zone_id,
        UdmSize_t cell_id,
        const char* solution_name,
        UdmReal_t* value);
要素（セル）の物理量データ値を取得する:real(スカラデータ).
[in] udm_handler          UdmModel クラスポインタ
[in] zone_id              ゾーン ID（1～）
[in] cell_id              要素（セル）ID（1～）
[in] solution_name        物理量データ名称
[out] value               取得物理データ値:real(スカラデータ)
戻り値                   エラー番号：UDM_OK | UDM_ERROR

```

```

UdmError_t udm_get_cellsolutions_integer(
        UdmHanler_t udm_handler,
        int zone_id,
        UdmSize_t cell_id,
        const char* solution_name,
        UdmInteger_t * values,
        int* size);
要素（セル）の物理量データ値を取得する:integer.
[in] udm_handler          UdmModel クラスポインタ
[in] zone_id              ゾーン ID（1～）
[in] cell_id              要素（セル）ID（1～）
[in] solution_name        物理量データ名称
[out] values              取得物理データ値

```

[out] size 取得物理データ数
 戻り値 エラー番号 : UDM_OK | UDM_ERROR

```
UdmError_t udm_get_nodesolutions_real(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t cell_id,
    const char* solution_name,
    UdmReal_t* values,
    int* size)
```

要素（セル）の物理量データ値を取得する:real.

[in] udm_handler UdmModel クラスポインタ
 [in] zone_id ゾーン ID（1～）
 [in] cell_id 要素（セル）ID（1～）
 [in] solution_name 物理量データ名称
 [out] values 取得物理データ値
 [out] size 取得物理データ数
 戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```
UdmHanler_t model = udm_create_model();
int zone_id = 1;
UdmSize_t num_cells = udm_getnum_cells(model, zone_id);
for (int n=1; n<=num_cells; n++) {
    // 物理量の取得
    float temprature = 0.0;
    udm_get_cellsolution_real(model, zone_id, n, "Templature", &temprature);
    UdmReal_t velocity[3] = {0.0, 0.0, 0.0};
    int size = 0;
    udm_get_cellsolutions_real(model, zone_id, n, "Velocity", velocity, &size);
}
```

物理量の値は、物理量定義されたデータ型によって内部的に持ちます。

CXX API の場合、物理量の取得データ型は、int, long long, float, double となりますが、定義データから取得データ型に変換して値を返します。

CC API の場合、整数値、実数値のデータ型は UdmInteger_t, UdmReal_t 型となります。
これは、ビルド時の configure オプションによって、int, long long、又は float, double に
定義されます。

4.5.4 要素（セル）への物理量の設定

要素（セル）に物理量の値を設定します。

(CXX API)

<pre>template<class VALUE_TYPE> UdmError_t UdmEntity::setSolutionScalar(const std::string& solution_name, VALUE_TYPE value) 物理量データ値を設定する. 物理量データ名称が存在しない場合は、追加する. solution_name 物理量データ名称 value 物理データ値 戻り値 エラー番号 : UDM_OK UDM_ERROR</pre> <pre>template<class VALUE_TYPE> UdmError_t UdmEntity::setSolutionVector(const std::string& solution_name, const VALUE_TYPE* values, unsigned int size = 3); 物理量データ値リストを設定する. solution_name 物理量データ名称 values 物理量データ値リスト size 物理量データ数 (default = 3) 戻り値 エラー番号 : UDM_OK UDM_ERROR</pre>	
使用例	<pre>UdmZone *zone = model->getZone(); for (n=1; n<=zone->getNumNodes(); n++) { UdmNode *node = zone->getNode(n); float templatrue = 1.0; node->setSolutionScalar("Templatrue", templatrue);</pre>

```

float velocity[3] = {0.0, 0.0, 0.0};
node->setSolutionVector("Velocity", velocity);
}

```

(CC API)

```

UdmError_t udm_set_cellsolution_integer(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t cell_id,
    const char* solution_name,
    UdmInteger_t value);

```

要素（セル）の物理量データ値を設定する:integer(スカラデータ).

udm_handler UdmModel クラスポインタ

zone_id ゾーン ID（1～）

cell_id 要素（セル）ID（1～）

solution_name 物理量データ名称

value 物理データ値:integer(スカラデータ)

戻り値 エラー番号：UDM_OK | UDM_ERROR

```

UdmError_t udm_set_nodesolution_real(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t cell_id,
    const char* solution_name,
    UdmReal_t value);

```

要素（セル）の物理量データ値を設定する:real(スカラデータ).

udm_handler UdmModel クラスポインタ

zone_id ゾーン ID（1～）

cell_id 要素（セル）ID（1～）

solution_name 物理量データ名称

value 物理データ値:real(スカラデータ)

戻り値 エラー番号：UDM_OK | UDM_ERROR

```

UdmError_t udm_set_nodesolutions_integer(
    UdmHanler_t udm_handler,

```

```

int zone_id,
UdmSize_t cell_id,
const char* solution_name,
const UdmInteger_t *values,
int size);

```

要素（セル）に物理量データ値リストを設定する:integer(ベクトル).

```

udm_handler      UdmModel クラスポインタ
zone_id          ゾーン ID（1～）
cell_id          要素（セル）ID（1～）
solution_name    物理量データ名称
values           物理量データ値リスト:integer
size             物理量データ数 (default = 3)
戻り値          エラー番号 : UDM_OK | UDM_ERROR

```

```

UdmError_t udm_set_nodesolutions_real(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t cell_id,
    const char* solution_name,
    const UdmReal_t *values,
    int size);

```

要素（セル）に物理量データ値を設定する:real(ベクトル).

```

udm_handler      UdmModel クラスポインタ
zone_id          ゾーン ID（1～）
cell_id          要素（セル）ID（1～）
solution_name    物理量データ名称
values           物理量データ値リスト:real
size             物理量データ数 (default = 3)
戻り値          エラー番号 : UDM_OK | UDM_ERROR

```

使用例

```

UdmHanler_t model = udm_create_model();
int zone_id = 1;
UdmSize_t num_cells = udm_getnum_cells(model, zone_id);
for (int n=1; n<=num_cells; n++) {
    float temprature = 0.0;

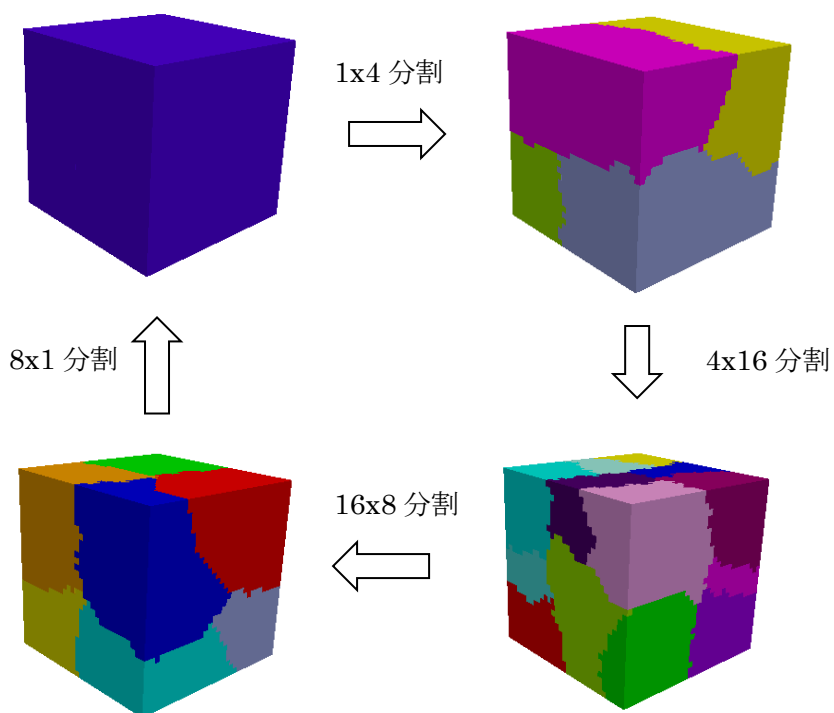
```

```
    udm_set_cellsolution_real(model, zone_id, n, "Templature", templature );  
    UdmReal_t velocity[3] = {0.0, 0.0, 0.0};  
    udm_set_cellsolutions_real(model, zone_id, n, "Velocity", velocity, 3);  
}
```


4.6 分割実行

モデルを MPI プロセス数に分割します。

分割ライブラリとして **Zoltan** ライブラリを利用しています。

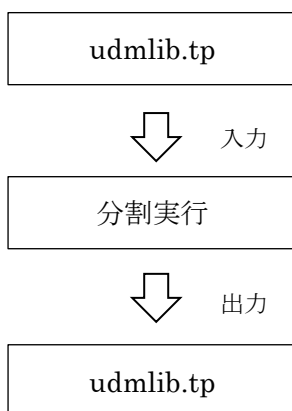


Zoltan ライブラリの分割パラメータの内、以下については **UDM** ライブラリにてサポートし、取得、設定 API を提供します。

設定項目	設定値	説明
LB_METHOD	HYPERGRAPH GRAPH	分割方法 デフォルト=HYPERGRAPH
LB_APPROACH	PARTITION REPARTITION REFINE	再分配方法 デフォルト=PARTITION

また、その他の **Zoltan** ライブラリのパラメータが存在し、取得、設定 API を提供します。
Zoltan ライブラリのパラメータについては、**Zoltan** マニュアルを参照してください。

分割パラメータは"udmlib.tp"から入出力、またはユーザプログラムから設定可能です。



4.6.1 分割実行

モデルをゾーン毎に MPI プロセス数に分割します。

(CXX API)

<pre>UdmError_t UdmModel::partitionZone(int zone_id = 1)</pre> <p>ゾーンの分割を行う。</p> <p>zone_id ゾーン ID (1 ~) : デフォルト = 1</p> <p>戻り値 エラー番号 : UDM_OK UDM_ERROR</p>
<p>使用例</p> <pre>// モデルの生成 UdmModel *model = new UdmModel(); model->loadModel("index.dfi", 0); // 分割実行 model->partitionZone(); // 分割ファイル出力 : step = 0 model->writeModel(0, 0.0);</pre>

(CC API)

<pre>UdmError_t udm_partition_zone(UdmHanler_t udm_handler,</pre>
--

<pre> int zone_id) ゾーンの分割を行う。 udm_handler UdmModel クラスポインタ zone_id ゾーン ID (1 ~) 戻り値 エラー番号 : UDM_OK UDM_ERROR </pre>
<p>使用例</p> <pre> UdmHanler_t model = udm_create_model(); udm_load_model(model, "index.dfi", 0); // 分割実行 int zone_id = 1; udm_partition_zone(model, zone_id); // 分割ファイル出力 : step = 0 udm_write_model(model, 0, 0.0); </pre>

4.6.2 分割パラメータ

Zoltan ライブラリの分割パラメータを取得、設定します。

(1) HYPERGRAPH パラメータ設定

分割方法:LB_METHOD を"HYPERGRAPH"とし、LB_APPROACH パラメータを設定します。

設定項目	設定値	説明
LB_METHOD	HYPERGRAPH	分割方法=HYPERGRAPH
LB_APPROACH	PARTITION REPARTITION REFINE	再分配方法 デフォルト=PARTITION

(CXX API)

<pre> UdmError_t UdmLoadBalance::setHyperGraphParameters(const std::string& approach) Hypergraph partitioning のパラメータの設定を行う。 approach LB_APPROACH パラメータ値 戻り値 エラー番号 : UDM_OK UDM_ERROR,etc </pre>

使用例

```
// モデルの生成
UdmModel *model = new UdmModel();
model->loadModel("index.dfi", 0);
UdmLoadBalance *partition = model->getLoadBalance();
// 分割パラメータ=ハイパーグラフ選択 : PARTITION
partition->setHyperGraphParameters("PARTITION");

// 分割実行
model->partitionZone();
```

(CC API)

```
UdmError_t udm_partition_sethypergraph(
    UdmHanler_t udm_handler,
    const char* approach)
Hypergraph partitioning のパラメータの設定を行う.
udm_handler          UdmModel クラスポインタ
approach             LB_APPROACH パラメータ値
戻り値              エラー番号 : UDM_OK | UDM_ERROR,etc
```

使用例

```
UdmHanler_t model = udm_create_model();
udm_load_model(model, "index.dfi", 0);
// 分割パラメータ=ハイパーグラフ選択 : PARTITION
udm_partition_sethypergraph(model, "PARTITION");
// 分割実行
int zone_id = 1;
udm_partition_zone(model, zone_id);
```

(2) GRAPH パラメータ設定

分割方法:LB_METHOD を"GRAPH"とし、LB_APPROACH パラメータを設定します。

設定項目	設定値	説明
LB_METHOD	GRAPH	分割方法=GRAPH
LB_APPROACH	PARTITION REPARTITION REFINE	再分配方法 デフォルト=PARTITION

(CXX API)

<p>UdmError_t UdmLoadBalance::setGraphParameters(const std::string& approach) Graph partitioning のパラメータの設定を行う. approach LB_APPROACH パラメータ値 戻り値 エラー番号 : UDM_OK UDM_ERROR,etc</p>
<p>使用例</p> <pre>// モデルの生成 UdmModel *model = new UdmModel(); model->loadModel("index.dfi", 0); UdmLoadBalance *partition = model->getLoadBalance(); // 分割パラメータ=ハイパーグラフ選択 : PARTITION partition->setGraphParameters("PARTITION"); // 分割実行 model->partitionZone();</pre>

(CC API)

<p>UdmError_t udm_partition_setgraph(UdmHanler_t udm_handler, const char* approach) Graph partitioning のパラメータの設定を行う. udm_handler UdmModel クラスポインタ approach LB_APPROACH パラメータ値 戻り値 エラー番号 : UDM_OK UDM_ERROR,etc</p>
<p>使用例</p>

```

UdmHanler_t model = udm_create_model();
udm_load_model(model, "index.dfi", 0);
// 分割パラメータ=グラフ選択 : PARTITION
udm_partition_setgraph(model, "PARTITION");
// 分割実行
int zone_id = 1;
udm_partition_zone(model, zone_id);

```

(3) パラメータ取得、設定、削除

Zoltan 分割パラメータを取得、設定します。LB_METHOD、LB_APPROACH の設定値の取得も行えます。

設定パラメータは"udmlib.tp"に出力されます。

(CXX API)

```

UdmError_t UdmLoadBalance::getParameter(
    const std::string& name,
    std::string& value) const
Zoltan 分割パラメータを取得する.
[in] name      パラメータ名
[out] value     設定値
戻り値        エラー番号 : UDM_OK | UDM_ERROR,etc

UdmError_t UdmLoadBalance::setParameter(
    const std::string& name,
    const std::string& value)
Zoltan 分割パラメータを設定する.
[in] name      パラメータ名
[in] value     設定値
戻り値        エラー番号 : UDM_OK | UDM_ERROR,etc

UdmError_t UdmLoadBalance::removeParameter(const std::string& name)
UDMLib の Zoltan パラメータを削除、又はデフォルト値とする.
[in] name      パラメータ名
戻り値        エラー番号 : UDM_OK | UDM_ERROR,etc

```

使用例

```
// モデルの生成
UdmModel *model = new UdmModel();
UdmLoadBalance *partition = model->getLoadBalance();
std::string approach;
partition->getParameters("LB_APPROACH", approach);
partition->setParameters("PHG_MULTILEVEL", "1");
```

(CC API)

```
const char* udm_partition_getparameter(
    UdmHanler_t udm_handler,
    const char* name,
    char* value)
Zoltan 分割パラメータを取得する.
[in]  udm_handler      UdmModel クラスポインタ
[in]  name             パラメータ名
[out] value            設定値
戻り値                取得設定値ポインタ : NULL の場合は取得エラー
```

```
UdmError_t udm_partition_setparameter(
    UdmHanler_t udm_handler,
    const char* name,
    const char* value)
Zoltan 分割パラメータを設定する.
[in]  udm_handler      UdmModel クラスポインタ
[in]  name             パラメータ名
[in]  value            設定値
戻り値                エラー番号 : UDM_OK | UDM_ERROR,etc
```

```
UdmError_t udm_partition_removeparameter(
    UdmHanler_t udm_handler,
    const char* name)
Zoltan 分割パラメータを削除、又はデフォルト値とする.
[in]  udm_handler      UdmModel クラスポインタ
```

[in] name	パラメータ名
戻り値	エラー番号 : UDM_OK UDM_ERROR,etc
<p>使用例</p> <pre> UdmHanler_t model = udm_create_model(); char approach[32] = {0x00}; udm_partition_getparameter(model, "LB_APPROACH", approach); udm_partition_setparameter(model, "PHG_MULTILEVEL", "1"); </pre>	

(4) Zoltan デバッグレベル

Zoltan 分割時のデバッグ出力レベルを設定します。デフォルトは"1"です。
現在のデバッグレベルは、"DEBUG_LEVEL"パラメータにて取得できます。

(CXX API)

UdmError_t UdmLoadBalance::setZoltanDebugLevel(int debug_level)	
Zoltan のデバッグレベルを設定する.	
debug_level	Zoltan のデバッグレベル
戻り値	エラー番号 : UDM_OK UDM_ERROR,etc

使用例	
// モデルの生成	
UdmModel *model = new UdmModel();	
UdmLoadBalance *partition = model->getLoadBalance();	
partition->setZoltanDebugLevel(0);	// Quiet mode

(CC API)

UdmError_t udm_partition_setdebuglevel(
UdmHanler_t udm_handler,
int debug_level)
Zoltan のデバッグレベルを設定する.
udm_handler UdmModel クラスポインタ
debug_level Zoltan のデバッグレベル
戻り値 エラー番号 : UDM_OK UDM_ERROR,etc

使用例

```
UdmHanler_t model = udm_create_model();  
udm_partition_setdebuglevel(model, 0);    // Quiet mode
```

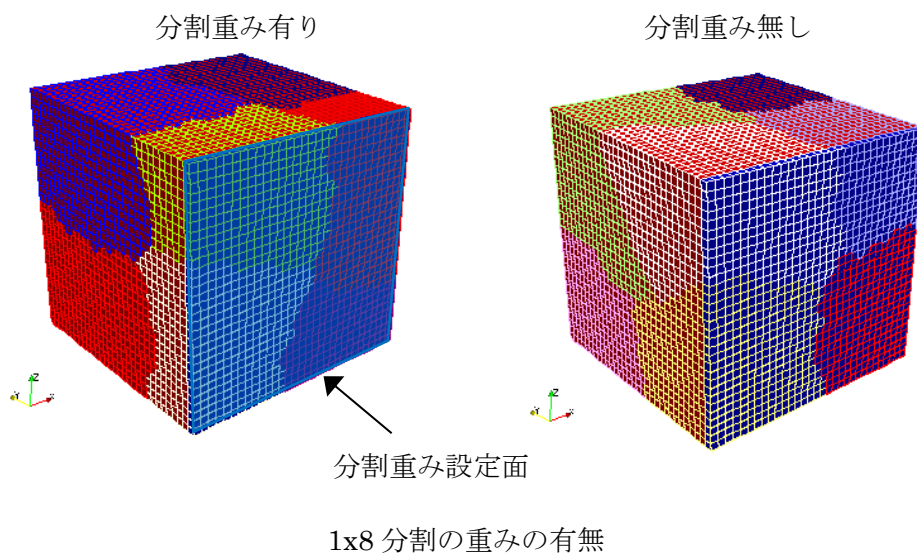
4.6.3 分割重みの取得、設定、クリア

要素（セル）に分割重みを設定します。

Zoltan 分割は重みが均一になるように分割を行います。

分割後も設定した重みは保持されます。

重みはゾーン単位でクリア可能です。



(CXX API)

```
float UdmEntity::getPartitionWeight() const
```

分割重みを取得する.

戻り値 分割重み

```
void UdmEntity::setPartitionWeight(float weight)
```

分割重みを設定する.

weight 分割重み

```
void UdmZone::clearPartitionWeight()
```

分割重みをクリアする.
分割重みフラグをクリアする.
要素（セル）に設定している重み値をクリアする.

使用例

```
UdmModel *model = new UdmModel();
UdmZone *zone = NULL;
// index.dfi 読込、CGNS 読込
model->loadModel(dfi_name);
// ゾーンの取得
UdmZone *zone = model->getZone();
// 重み設定
int cell_size = zone->getNumCells();
for (int n=1; n<=cell_size; n++) {
    UdmCell *cell = zone->getCell(n);
    int node_size = cell->getNumNodes();
    float weight = 1.0;
    for (int m=1; m<=node_size; m++) {
        // 節点（ノード）座標の取得
        UdmNode *node = cell->getNode(m);
        float x = 0, y = 0, z = 0;
        node->getCoords(x, y, z);
        // -J 面に重み設定を行う
        if (y == 0.0) {
            weight = 10.0;           // 重み設定面
            break;
        }
    }
    // 要素（セル）に重み設定
    cell->setPartitionWeight(weight);
}
// 分割実行
model->partitionZone();
// 重みクリア
zone->clearPartitionWeight();
```

(CC API)

```
void udm_get_partitionweight(  
    UdmHanler_t udm_handler,  
    int zone_id,  
    UdmSize_t cell_id,  
    float* weight)  
分割重みを取得する.  
[in] udm_handler      UdmModel クラスポインタ  
[in] zone_id          ゾーン ID ( 1 ~ )  
[in] cell_id          要素 (セル) ID ( 1 ~ )  
[out] weight          分割重み
```

```
void udm_set_partitionweight(  
    UdmHanler_t udm_handler,  
    int zone_id,  
    UdmSize_t cell_id,  
    const float weight)  
要素 (セル) に分割重みを設定する.  
udm_handler          UdmModel クラスポインタ  
zone_id              ゾーン ID ( 1 ~ )  
cell_id              要素 (セル) ID ( 1 ~ )  
weight              分割重み
```

```
void udm_clear_partitionweight(  
    UdmHanler_t udm_handler,  
    int zone_id)  
分割重みをクリアする.  
分割重みフラグをクリアする.  
要素 (セル) に設定している重み値をクリアする.  
[in] udm_handler      UdmModel クラスポインタ  
[in] zone_id          ゾーン ID ( 1 ~ )
```

使用例

```
// モデルの生成  
UdmHanler_t model = udm_create_model();
```

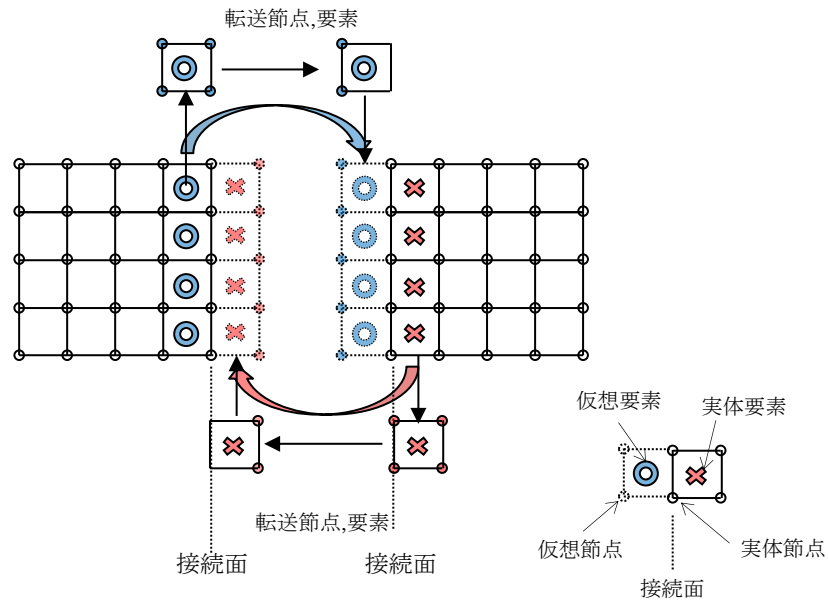
```

// index.dfi 読込、CGNS 読込
udm_load_model(model, argv[1], 0);
int zone_id = 1;          // 先頭ゾーンとする.
// 重み設定
int cell_size = udm_getnum_cells(model, zone_id);
for (int n=1; n<=cell_size; n++) {
    float weight = 1.0;
    UdmElementType_t elem_type;
    UdmSize_t node_ids[8];
    unsigned int num_nodes = 0;
    udm_get_cellconnectivity(
        model, zone_id, n,
        &elem_type, node_ids, &num_nodes);
    for (int m=0; m<num_nodes; m++) {
        // 節点 (ノード) 座標の取得
        UdmReal_t x = 0, y = 0, z = 0;
        udm_get_gridcoordinates(model, zone_id, node_ids[m], &x, &y, &z);
        // -J 面に重み設定を行う
        if (y == 0.0) {
            weight = 10.0;          // 重み設定面
            break;
        }
    }
    // 要素 (セル) に重み設定
    udm_set_partitionweight(model, zone_id, n, weight);
}
// 分割実行
udm_partition_zone(model, zone_id);
// 重みクリア
udm_clear_partitionweight(model, zone_id);

```

4.7 仮要素（セル）の転送

接続面の隣接接点（ノード）、要素（セル）から構成される仮想節点（ノード）、要素（セル）に実体節点（ノード）、要素（セル）の物理量を転送します。



- 接続面の節点（ノード）の実体要素（セル）を取得します。
- 実体要素（セル）、構成接点（ノード）及び物理量を接続先 MPI ランクに送信します。
- 受信した要素（セル）を仮要素（セル）に追加、上書します。受信した接点（ノード）の仮想節点（ノード）を追加、上書します。

(CXX API)

```
UdmError_t UdmModel::transferVirtualCells()
```

仮想セルの転送を行う。

戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```
// モデルの生成
UdmModel *model = new UdmModel();
model->loadModel(dfname);
// 仮想セルの転送
model->transferVirtualCells();
```

(CC API)

`UdmError_t udm_transfer_virtualcells(UdmHanler_t udm_handler)`

仮想セルの転送を行う.

`udm_handler` `UdmModel` クラスポインタ

戻り値 エラー番号 : `UDM_OK` | `UDM_ERROR`

使用例

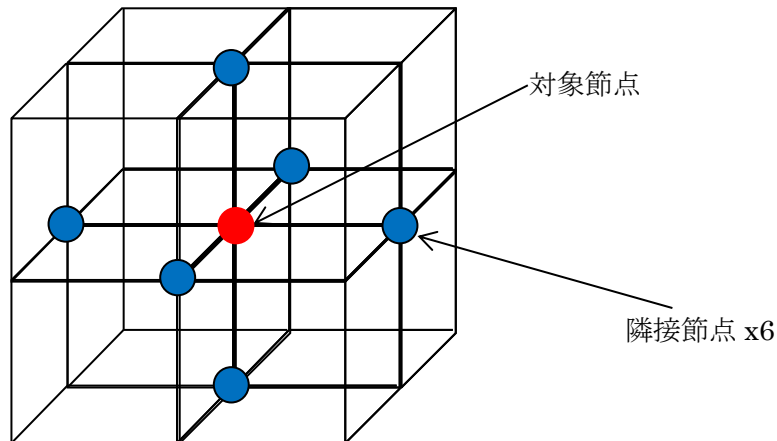
```
// モデルの生成
UdmHanler_t model = udm_create_model();
udm_load_model(model, dfname, 0);
// 仮想セルの転送
udm_transfer_virtualcells(model);
```

4.8 隣接節点（ノード）・要素（セル）

接点（ノード）、要素（セル）の隣接接点（ノード）、要素（セル）を取得します。

4.8.1 隣接節点（ノード）の取得

取得対象節点の隣接接点（ノード）を取得します。



(CXX API)

```
UdmSize_t UdmNode::getNumNeighborNodes() const
```

節点（ノード）の隣接節点（ノード）数を取得する.

戻り値 隣接節点（ノード）数

```
UdmNode* UdmNode::getNeighborNode(int neighbor_id) const
```

節点（ノード）の隣接節点（ノード）を取得する.

neighbor_id 隣接 ID : 1～getNumNeighborNodes()

戻り値 隣接節点（ノード）

使用例

```
// モデルの生成
UdmModel *model = new UdmModel();
model->loadModel(dfname);
UdmZone *zone = model->getZone();
for (int n=1; n<=zone->getNumNodes(); n++) {
    UdmNode *node = zone->getNode(n);
    float sum = 0.0;
```

```

int num_neighbor = node->getNumNeighborNodes();
for (int i=1; i<=num_neighbor; i++) {
    UdmNode *neighbor_node = node->getNeighborNode(i);
    tmp = 0.0;
    neighbor_node->getSolutionScalar("templature", tmp);
    sum += tmp;
}
}

```

(CC API)

```

int udm_getnum_neighbornodes(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t node_id)

```

節点（ノード）の隣接節点（ノード）数を取得する.

udm_handler UdmModel クラスポインタ

zone_id ゾーン ID（1～）

node_id 節点（ノード）ID（1～）

戻り値 隣接節点（ノード）数

```

int udm_get_neighbornodes(
    UdmHanler_t udm_handler,
    int zone_id,
    UdmSize_t node_id,
    UdmSize_t* neighbor_nodeids,
    UdmRealityType_t *neighbor_types,
    int* num_neighbors)

```

節点（ノード）の隣接節点（ノード）を取得する.

[in] udm_handler UdmModel クラスポインタ

[in] zone_id ゾーン ID（1～）

[in] node_id 節点（ノード）ID（1～）

[out] neighbor_nodeids 隣接節点（ノード）ID リスト

[out] neighbor_types 隣接節点（ノード）タイプリスト

[out] num_neighbors 隣接節点（ノード）数

戻り値 隣接節点（ノード）数

使用例

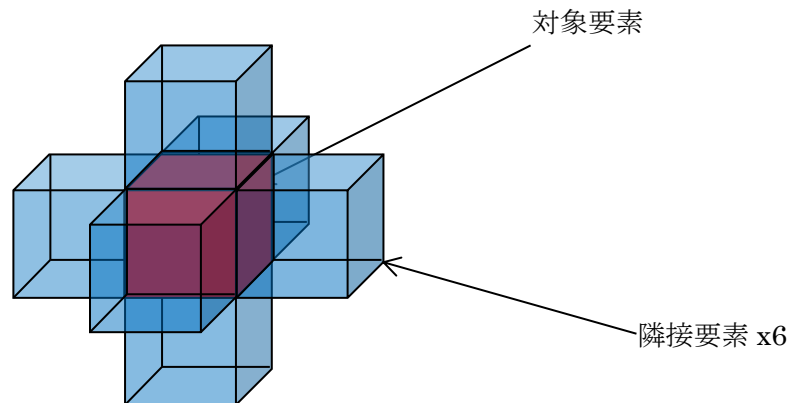
```
// モデルの生成
UdmHanler_t model = udm_create_model();
udm_load_model(model, dfname, 0);
int zone_id = 1;
UdmSize_t num_nodes = udm_getnum_nodes(model, zone_id);
for (int n=1; n<=num_nodes; n++) {
    // 接続節点（ノード）から物理量の計算
    float sum = 0.0;
    int num_neighbor = udm_getnum_neighbornodes(model, zone_id, n);
    UdmSize_t neighbor_nodeids[6];
    UdmRealityType_t neighbor_types [6];
    udm_get_neighbornodes(model, zone_id, n,
                          neighbor_nodeids, neighbor_types, &num_neighbor);
    for (int i=0; i<num_neighbor; i++) {
        tmp = 0.0;
        udm_get_nodesolution_real(model, zone_id, neighbor_nodeids[i],
                                  "templature", &tmp);
        sum += tmp;
    }
}
}
```

CXX API の場合、UdmNode::getNeighborNode() の隣接 ID は 1 ～ UdmNode::getNumNeighborNodes() の範囲の整数となります。

CC API の場合、udm_get_neighbornodes() の隣接節点（ノード）ID はゾーン内の節点（ノード）ID となります。

4.8.2 隣接要素（セル）の取得

取得対象要素（セル）の隣接要素（セル）を取得します。



(CXX API)

```
UdmSize_t UdmCell::getNumNeighborCells() const
```

要素 (セル) の隣接要素 (セル) 数を取得する.

戻り値 隣接要素 (セル) 数

```
UdmCell* UdmCell::getNeighborCell(int neighbor_id) const
```

要素 (セル) の隣接要素 (セル) を取得する.

neighbor_id 隣接 ID : 1 ~ getNumNeighborCells()

戻り値 隣接要素 (セル)

使用例

```
// モデルの生成
UdmModel *model = new UdmModel();
model->loadModel(dfname);
UdmZone *zone = model->getZone();
for (int n=1; n<=zone->getNumCells(); n++) {
    UdmCell *cell = zone->getCell(n);
    // 隣接要素 (セル)
    UdmSize_t num_neighbor = cell->getNumNeighborCells();
    for (i=1; i<=num_neighbor; i++) {
        UdmCell *neighbor_cell = cell->getNeighborCell(i);
        float neighbor_t0 = 0.0;
        neighbor_cell->getSolutionScalar("templatue", neighbor_t0);
    }
}
```

```
}
```

(CC API)

```
int udm_getnum_neighborcells(  
    UdmHanler_t udm_handler,  
    int zone_id,  
    UdmSize_t cell_id)  
要素 (セル) の隣接要素 (セル) 数を取得する.  
udm_handler      UdmModel クラスポインタ  
zone_id          ゾーン ID (1 ~)  
cell_id          要素 (セル) ID (1 ~)  
戻り値           隣接要素 (セル) 数
```

```
int udm_get_neighborcells(  
    UdmHanler_t udm_handler,  
    int zone_id,  
    UdmSize_t cell_id,  
    UdmSize_t* neighbor_cellids,  
    UdmRealityType_t *neighbor_types,  
    int* num_neighbors)  
要素 (セル) の隣接要素 (セル) を取得する.  
[in] udm_handler      UdmModel クラスポインタ  
[in] zone_id          ゾーン ID (1 ~)  
[in] cell_id          要素 (セル) ID (1 ~)  
[out] neighbor_cellids 隣接要素 (セル) ID リスト  
[out] neighbor_types   隣接要素 (セル) タイプリスト  
[out] num_neighbors    隣接要素 (セル) 数  
戻り値           隣接要素 (セル) 数
```

使用例

```
// モデルの生成  
UdmHanler_t model = udm_create_model();  
udm_load_model(model, dfname, 0);  
int zone_id = 1;  
UdmSize_t num_cells = udm_getnum_cells(model, zone_id);
```

```

for (int n=1; n<=num_cells; n++) {
    // 隣接要素 (セル)
    UdmSize_t num_neighbor = udm_getnum_neighborcells(model, zone_id, n);
    UdmSize_t neighbor_cellids[6];
    UdmRealityType_t neighbor_types[6];
    udm_get_neighborcells(model, zone_id, n,
                          neighbor_cellids, neighbor_types, &size);
    for (int i=0; i<num_neighbor; i++) {
        UdmReal_t neighbor_t0 = 0.0;
        udm_get_cellsolution_real(model, zone_id,
                                  neighbor_cellids[i], "templature", &neighbor_t0);
    }
}

```

CXX API の場合、UdmCell::getNeighborCell(); の隣接 ID は 1 ～ UdmCell::getNumNeighborCells()の範囲の整数となります。

CC API の場合、udm_get_neighborcells();の隣接要素 (セル) ID はゾーン内の要素 (セル) ID となります。

4.9 ユーザ定義情報

ソルバを実行する上で節点（ノード）、要素（セル）に設定する物理量（実数又は整数）以外に様々なパラメータが存在します。

そのパラメータを UDM ライブラリでは、出力 CGNS ファイルにユーザ定義情報として出力することができます。

ユーザ定義情報は多次元の配列を定義することができます。また任意のユーザ定義データ名により複数のユーザ定義情報を持つことができ、それぞれにデータ型を定義することができます。

ユーザ定義データに定義できるデータ型は以下です。

UdmDataType_t	C 言語データ型	データサイズ
Udm_Integer	int	4byte
Udm_LongInteger	long long	8byte
Udm_RealSingle	float	4byte
Udm_RealDouble	double	8byte

4.9.1 ユーザ定義情報の取得

ユーザ定義情報の取得を行います。

(CXX API)

```
UdmError_t UdmUserDefinedDatas::getUserDataInfo(  
    const std::string& array_name,  
    UdmDataType_t& data_type,  
    int& dimension,  
    UdmSize_t* dim_sizes) const
```

ユーザ定義データ情報を取得する。

[in] array_name	ユーザ定義データ名
[out] data_type	データ型
[out] dimension	データ次元数
[out] dim_sizes	データ次元毎のデータ数
戻り値	エラー番号 : UDM_OK UDM_ERROR

```
UdmError_t UdmUserDefinedDatas::getUserDataArray(  
    const std::string& array_name,  
    UdmDataType_t data_type,  
    void* data) const;
```

ユーザ定義データを取得する.

[in] array_name ユーザ定義データ名

[in] data_type 取得データ型

[out] data ユーザ定義データ

戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```
UdmDataType_t datatype;
int dimension = 0, n;
UdmSize_t dim_sizes[3] = {0};
UdmZone *zone = model->getZone();
// ユーザ定義データ
const UdmUserDefinedDatas* user_defined = zone->getUserDefinedDatas();
user_defined->getUserDataInfo("UserMatrix", datatype, dimension, dim_sizes);
int len = 1;
for ( n=0; n<dimension; n++) {
    len *= dim_sizes[n];
}
float *mat_values = new float[len];
user_defined->getUserDataArray("UserMatrix", Udm_RealSingle, mat_values);
```

(CC API)

```
UdmError_t udm_user_getinfo(
    UdmHanler_t udm_handler,
    int zone_id,
    const char* user_name,
    UdmDataType_t* data_type,
    int* dimension,
    UdmSize_t* dim_sizes)
ユーザ定義データ情報を取得する.
[in] udm_handler           UdmModel クラスポインタ
[in] zone_id            ゾーン ID ( 1 ~ )
[in] array_name           ユーザ定義データ名
[out] data_type           データ型
```

[out] dimension データ次元数
 [out] dim_sizes データ次元毎のデータ数
 戻り値 エラー番号 : UDM_OK | UDM_ERROR

```
UdmError_t udm_user_getdata(
    UdmHanler_t udm_handler,
    int zone_id,
    const char* user_name,
    UdmDataType_t data_type,
    void* data);
```

ユーザ定義データを取得する.

[in] udm_handler UdmModel クラスポインタ
 [in] zone_id ゾーン ID (1 ~)
 [in] user_name ユーザ定義データ名
 [in] data_type 取得データ型
 [out] data ユーザ定義データ
 戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```
UdmDataType_t datatype;
int dimension = 0, n;
UdmSize_t dim_sizes[3] = {0};
int len;
UdmHanler_t model = udm_create_model();
int zone_id = 1;

// ユーザ定義データ
udm_user_getinfo(model, zone_id,
    "UserMatrix", &datatype, &dimension, dim_sizes);

len = 1;
for (n=0; n<dimension; n++) {
    len *= dim_sizes[n];
}

float *mat_values = (float*)malloc(sizeof(float)*len);
udm_user_getdata(model, zone_id, "UserMatrix", Udm_RealSingle, mat_values);
```

4.9.2 ユーザ定義情報の設定

ユーザ定義情報の取得を行います。

(CXX API)

<pre>UdmError_t UdmUserDefinedDatas::setUserData(const std::string& array_name, UdmDataType_t data_type, int dimension, UdmSize_t* dim_sizes, void* data);</pre>	
ユーザ定義データを設定する.	
array_name	ユーザ定義データ名
data_type	データ型
dimension	データ次元数
dim_sizes	データ次元毎のデータ数
data	ユーザ定義データ
戻り値	エラー番号 : UDM_OK UDM_ERROR

使用例	
<pre>UdmZone *zone = model->getZone(); // ユーザ定義データの設定 UdmUserDefinedDatas* user_defined = zone->getUserDefinedDatas(); UdmSize_t user_sizes[2] = {3, 10}; float user_real[10][3]; int value = 0; for (j=0; j<user_sizes[1]; j++) { for (i=0; i<user_sizes[0]; i++) { user_real[j][i] = 0.5*(value++); } } user_defined->setUserData("UserMatrix", Udm_RealSingle, 2, user_sizes, user_real);</pre>	

(CC API)

```
UdmError_t udm_user_setdata(  
    UdmHanler_t udm_handler,  
    int zone_id,  
    const char* user_name,  
    UdmDataType_t data_type,  
    int dimension,  
    UdmSize_t* dim_sizes,  
    void* data)  
ユーザ定義データを設定する。  
[in] udm_handler      UdmModel クラスポインタ  
[in] zone_id          ゾーン ID ( 1 ~ )  
[in] array_name       ユーザ定義データ名  
[in] data_type         データ型  
[in] dimension         データ次元数  
[in] dim_sizes         データ次元毎のデータ数  
[in] data              ユーザ定義データ  
戻り値      エラー番号 : UDM_OK | UDM_ERROR
```

使用例

```
UdmHanler_t model = udm_create_model();  
int zone_id = 1;  
// ユーザ定義データの設定  
UdmSize_t user_sizes[2] = {3, 10};  
float user_real[10][3];  
int value = 0;  
for (j=0; j<user_sizes[1]; j++) {  
    for (i=0; i<user_sizes[0]; i++) {  
        user_real[j][i] = 0.5*(value++);  
    }  
}  
udm_user_setdata(model, zone_id,  
    "UserMatrix",  
    Udm_RealSingle,
```

```
2,
user_sizes,
user_real);
```

4.9.3 ユーザ定義情報の削除

ユーザ定義情報の削除を行います。

(CXX API)

```
UdmError_t UdmUserDefinedDatas::removeUserData(
    const std::string &array_name);
```

ユーザ定義データを削除する.

array_name ユーザ定義データ名

戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```
UdmZone *zone = model->getZone();
// ユーザ定義データの設定
UdmUserDefinedDatas* user_defined = zone->getUserDefinedDatas();
user_defined->removeUserData("UserMatrix");
```

(CC API)

```
UdmError_t udm_user_remove(
    UdmHanler_t udm_handler,
    int zone_id,
    const char* user_name);
```

ユーザ定義データを削除する.

[in] udm_handler UdmModel クラスポインタ

[in] zone_id ゾーン ID (1 ~)

[in] user_name ユーザ定義データ名

戻り値 エラー番号 : UDM_OK | UDM_ERROR

使用例

```
UdmHanler_t model = udm_create_model();
```

```
int zone_id = 1;  
udm_user_remove(model, zone_id, "UserMatrix");
```

4.10 データ型定義、列挙型

UDM ライブラリで使用するデータ型定義、列挙体を説明します。

4.10.1 データ型

データ型の定義は `udm_define.h` にて宣言されています。

(1) UdmSize_t データ型

節点（ノード）、要素（セル）の ID、サイズのデータ型です。

UDM ライブラリのビルド時の `configure` オプションによってデータ型は異なります。

configure オプション	UdmSize_t データ型
なし（デフォルト）	unsigned int
<code>--enable-size8</code>	size_t（64 ビット環境=8 バイト整数）

(2) UdmReal_t データ型

CC API の物理量のデータ型です。

UDM ライブラリのビルド時の `configure` オプションによってデータ型は異なります。

configure オプション	UdmReal_t データ型
なし（デフォルト）	float（4byte）
<code>--enable-real8</code>	double（8byte）

(3) UdmInteger_t データ型

CC API の物理量のデータ型です。

UDM ライブラリのビルド時の `configure` オプションによってデータ型は異なります。

configure オプション	UdmInteger_t データ型
なし（デフォルト）	int（4byte）
<code>--enable-int8</code>	long long（8byte）

4.10.2 列挙型

列挙型の定義は `udmlib.h` にて宣言されています。

(1) UdmEnable_t 列挙型

ENABLE タイプを定義します。

UdmEnable_t	値	説明
Udm_EnableUnknown	0	不明
udm_disable	1	Disable
udm_enable	2	Enable

(2) UdmDataType_t 列挙型

データ型を定義します。

UdmDataType_t	値	説明
Udm_DataTypeUnknown,	0	不明
Udm_Integer	1	4byte 整数
Udm_LongInteger	2	8byte 整数
Udm_RealSingle	3	単精度実数
Udm_RealDouble	4	倍精度実数
Udm_String	5	文字列
Udm_Boolean	6	真偽値
Udm_Numeric	7	整数

(3) UdmFileCompositionType_t;列挙型

CGNS ファイル構成タイプを定義します。

UdmFileCompositionType_t	値	説明
Udm_FileCompositionTypeUnknown,	0	不明
Udm_IncludeGrid	1	CGNS:GridCoordinates と FlowSolution 同一構成
Udm_ExcludeGrid	2	CGNS:GridCoordinates と FlowSolution 別構成
Udm_AppendStep	3	時系列追加出力
Udm_EachStep	4	時系列毎出力
Udm_GridConstant	5	座標固定値出力

Udm_GridTimeSlice	6	座標値時系列出力
-------------------	---	----------

(4) UdmZoneType_t 列挙型

CGNS ゾーンタイプを定義します。

UdmZoneType_t	値	説明
Udm_ZoneTypeUnknown	0	不明
Udm_Structured	1	構造格子
Udm_Unstructured	2	非構造格子

(5) UdmGridLocation_t 列挙型

CGNS ゾーンタイプを定義します。

UdmGridLocation_t	値	説明
Udm_GridLocationUnknown	0	不明
Udm_Vertex	1	節点（ノード）
Udm_CellCenter	2	要素（セル）センター

(6) UdmVectorType_t 列挙型

ベクトルデータタイプを定義します。

UdmVectorType_t	値	説明
Udm_VectorTypeUnknown	0	不明
Udm_Scalar	1	スカラーデータ型
Udm_Vector	3	ベクトルデータ型
Udm_Nvector	9	N 成分データ型

(7) UdmElementType_t 列挙型

要素形状タイプを定義します。

UdmElementType_t	値	説明
Udm_ElementTypeUnknown	0	不明
Udm_NODE	1	ノード（節点）
Udm_BAR_2	2	BAR 要素
Udm_TRI_3	3	Shell:三角形要素

Udm_QUAD_4	4	Shell:四角形要素
Udm_TETRA_4	5	Solid:四面体要素
Udm_PYRA_5	6	Solid:ピラミッド要素
Udm_PENTA_6	7	Solid:五面体要素
Udm_HEXA_8	8	Solid:六面体要素
Udm_MIXED	9	混合要素

(8) UdmRealityType_t 列挙型

節点（ノード）、要素（セル）の仮想タイプを定義します。

UdmRealityType_t	値	説明
Udm_RealityTypeUnknown	0	不明
Udm_Virtual	1	仮想節点（ノード）、要素（セル）
Udm_Actual	2	実体節点（ノード）、要素（セル）

(9) UdmRealityType_t 列挙型

節点（ノード）、要素（セル）の仮想タイプを定義します。

UdmRealityType_t	値	説明
Udm_RealityTypeUnknown	0	不明
Udm_Virtual	1	仮想節点（ノード）、要素（セル）
Udm_Actual	2	実体節点（ノード）、要素（セル）

(10) UdmSimulationType_t 列挙型

時系列シミュレーションタイプを定義します。

UdmSimulationType_t	値	説明
Udm_SimulationTypeUnknown	0	不明
Udm_TimeAccurate	1	タイムステップ
Udm_NonTimeAccurate	2	ワンステップ

(11) UdmMemArrayType_t 列挙型

メモリ配列タイプを定義します。

UdmMemArrayType_t	値	説明
-------------------	---	----

Udm_MemArrayTypeUnknown	0	不明
Udm_MemSequentialArray	1	スカラメモリ配列（成分配列） { {X0,X1,X2,X3}, {Y0,Y1,Y2,Y3}, {Z0,Z1,Z2,Z3} }
Udm_MemIndexesArray	2	ベクトルメモリ配列（ノード配列） { {X0,Y0,Z0}, {X1,Y1,Z1}, {X2,Y2,Z2}, {X3,Y3,Z3} }

(12) UdmCellClass_t 列挙型

要素クラスタイプを定義します。

UdmCellClass_t	値	説明
Udm_CellClassUnknown	0	不明
Udm_CellClass	1	要素（セル）クラス
Udm_ComponentClass	2	部品要素（セル）クラス

5. UDM ライブラリ API 一覧

以下に UDM ライブラリが提供する API 一覧を示します。

(1) UDM モデルの構成要素

機能	API	
UDM モデルの生成	CXX	UdmModel::UdmModel
	CC	udm_create_model
UDM モデルの破棄	CXX	UdmModel::~~UdmModel
	CC	udm_delete_model
UDM ゾーンの生成	CXX	UdmModel::createZone
	CC	udm_create_zone
UDM ゾーン数の取得	CXX	UdmModel::getNumZones
	CC	udm_getnum_zones
UDM ゾーン の取得	CXX	UdmModel::getZone
節点（ノード）座標クラス の取得	CXX	UdmZone::getGridCoordinates
UDM セクション管理クラス の取得	CXX	UdmZone::getSections
UDM セクションの生成	CXX	UdmSections::createSection
	CC	udm_create_section
セクション（要素構成）数の取得	CXX	UdmSections::getNumSections
セクションの取得	CXX	UdmSections::getSection

(2) DFI ファイルの入出力

機能	API	
DFI ファイルの読込	CXX	UdmModel::loadModel
	CC	udm_load_model
DFI ファイルの書込	CXX	UdmModel::writeModel
	CC	udm_write_model
出力ディレクトリの取得	CXX	UdmDfiConfig::getOutputPath
	CC	udm_config_getoutputpath
出力ディレクトリの設定	CXX	UdmDfiConfig::setOutputPath

機能	API	
	CC	udm_config_setoutputpath
フィールドデータディレクトリの取得	CXX	UdmFileInfoConfig::getDirectoryPath
	CC	udm_config_getfelddirectory
フィールドデータディレクトリ の設定	CXX	UdmFileInfoConfig::setDirectoryPath
	CC	udm_config_setfelddirectory
時刻ディレクトリの取得	CXX	UdmFileInfoConfig::isTimeSliceDirectory
	CC	udm_config_istimeslicedirectory
時刻ディレクトリ の設定	CXX	UdmFileInfoConfig::setTimeSliceDirectory
	CC	udm_config_settimeslicedirectory
ベースファイル名の取得	CXX	UdmFileInfoConfig::getPrefix
	CC	udm_config_getfileprefix
ベースファイル名 の設定	CXX	UdmFileInfoConfig::setPrefix
	CC	udm_config_setfileprefix
ファイル命名書式の取得	CXX	UdmFileInfoConfig::getFieldfilenameFormat
ファイル命名書式 の設定	CXX	UdmFileInfoConfig::setFieldfilenameFormat
CGNS ファイル構成タイプの 設定	CXX	UdmFileInfoConfig::setFileCompositionType
	CC	udm_config_setfilecomposition
CGNS ファイル構成タイプの 設定チェック	CXX	UdmFileInfoConfig::existsFileCompositionType
	CC	udm_config_existsfilecomposition
単位系の単位の取得	CXX	UdmUnitListConfig::getUnit
単位系の基準値の取得	CXX	UdmUnitListConfig::getReference
単位系の差分値の取得	CXX	UdmUnitListConfig::getDifference
単位系の設定	CXX	UdmUnitListConfig::setUnitConfig
	CC	udm_config_setunit udm_config_setunitwithdiff
単位系の存在チェック	CXX	UdmUnitListConfig::existsUnitConfig
	CC	udm_config_existsunit
単位系の削除	CXX	UdmUnitListConfig::removeUnitConfig
	CC	udm_config_removeunit
単位系の取得	CC	udm_config_getunit(
CGNS ファイルの読込	CXX	UdmModel::readCgns
	CC	udm_read_cgns

(3) 節点（ノード）

機能	API	
節点（ノード）の生成	CXX	UdmGridCoordinates::insertGridCoordinates
	CC	udm_insert_gridcoordinates
節点（ノード）数の取得	CXX	UdmZone::getNumNodes UdmGridCoordinates::getNumNodes
	CC	udm_getnum_nodes
節点（ノード）の取得	CXX	UdmZone::getNode UdmGridCoordinates::getNodeById
節点（ノード）の座標値の取得	CXX	UdmNode::getCoords
	CC	udm_get_gridcoordinates
節点（ノード）の座標値の設定	CXX	UdmNode::setCoords
	CC	udm_set_gridcoordinates
内部境界情報を節点（ノード）の追加	CXX	UdmGridCoordinates::insertRankConnectivity
	CC	udm_insert_rankconnectivity

(4) 要素（セル）

機能	API	
要素（セル）の生成	CXX	UdmElements::insertCellConnectivity
	CC	udm_insert_cellconnectivity
要素（セル）数の取得	CXX	UdmZone::getNumCells UdmElements::getNumCells
	CC	udm_getnum_cells
要素（セル）の取得	CXX	UdmZone::getCell UdmElements::getCell
構成節点（ノード）数の取得	CXX	UdmCell::getNumNodes
構成ノード（節点）の取得	CXX	UdmCell::getNode
	CC	udm_get_cellconnectivity

(5) 節点（ノード）・要素（セル）の物理量

機能	API	
物理量定義の設定	CXX	UdmFlowSolutionListConfig::

機能	API	
		setSolutionFieldInfo
	CC	udm_config_setsolution udm_config_setscalarsolution
物理量定義の取得	CXX	UdmFlowSolutionListConfig::getSolutionFieldInfo
	CC	udm_config_getsolution
物理量定義の存在チェック	CXX	UdmFlowSolutionListConfig::existsSolutionConfig
	CC	udm_config_existssolution
物理量定義の削除	CXX	UdmFlowSolutionListConfig::removeSolutionConfig
	CC	udm_config_removesolution
節点（ノード）、要素（セル）の物理量数の取得	CXX	UdmEntity::getNumSolutionValue
節点（ノード）、要素（セル）のスカラ物理量データ値の取得	CXX	UdmEntity::getSolutionScalar
節点（ノード）、要素（セル）のベクトル物理量データ値の取得	CXX	UdmEntity::getSolutionVector
節点（ノード）、要素（セル）のスカラ物理量データ値の設定	CXX	UdmEntity::setSolutionScalar
節点（ノード）、要素（セル）のベクトル物理量データ値の設定	CXX	UdmEntity::setSolutionVector
節点（ノード）の物理量データ値の取得:integer(スカラデータ)	CC	udm_get_nodesolution_integer
節点（ノード）の物理量データ値の取得:real(スカラデータ)	CC	udm_get_nodesolution_real
節点（ノード）の物理量データ値の取得:integer. (ベクトルデータ)	CC	udm_get_nodesolutions_integer
節点（ノード）の物理量データ値の取得:real. (ベクトルデータ)	CC	udm_get_nodesolutions_real
節点（ノード）に物理量データ	CC	udm_set_nodesolution_integer

機能	API	
値の設定:integer(スカラデータ)		
節点（ノード）に物理量データ値の設定:real(スカラデータ).	CC	udm_set_nodesolution_real
節点（ノード）に物理量データ値の設定する:integer(ベクトル)	CC	udm_set_nodesolutions_integer
節点（ノード）に物理量データ値の設定:real(ベクトル).	CC	udm_set_nodesolutions_real
要素（セル）の物理量データ値の取得:integer(スカラデータ)	CC	udm_get_cellsolution_integer
要素（セル）の物理量データ値の取得:real(スカラデータ)	CC	udm_get_cellsolution_real
要素（セル）の物理量データ値の取得:integer. (ベクトルデータ)	CC	udm_get_cellsolutions_integer
要素（セル）の物理量データ値の取得:real. (ベクトルデータ)	CC	udm_get_cellsolutions_real
要素（セル）に物理量データ値の設定:integer(スカラデータ)	CC	udm_set_cellsolution_integer
要素（セル）に物理量データ値の設定:real(スカラデータ).	CC	udm_set_cellsolution_real
要素（セル）に物理量データ値の設定する:integer(ベクトル)	CC	udm_set_cellsolutions_integer
要素（セル）に物理量データ値の設定:real(ベクトル).	CC	udm_set_cellsolutions_real

(6) 分割

機能	API	
ゾーンの分割	CXX	UdmModel::partitionZone
	CC	udm_partition_zone
Hypergraph partitioning のパラメータの設定	CXX	UdmLoadBalance::setHyperGraphParameters

機能	API	
	CC	udm_partition_sethypergraph
Graph partitioning のパラメータの設定	CXX	UdmLoadBalance::setGraphParameters
	CC	udm_partition_setgraph
Zoltan 分割パラメータの取得	CXX	UdmLoadBalance::getParameter
	CC	udm_partition_getparameter
Zoltan 分割パラメータの設定	CXX	UdmLoadBalance::setParameter
	CC	udm_partition_setparameter
Zoltan パラメータの削除	CXX	UdmLoadBalance::removeParameter
	CC	udm_partition_removeparameter
Zoltan デバッグレベルの設定	CXX	UdmLoadBalance::setZoltanDebugLevel
	CC	udm_partition_setdebuglevel
分割重みの取得	CXX	UdmEntity::getPartitionWeight
	CC	udm_get_partitionweight
分割重みの設定	CXX	UdmEntity::setPartitionWeight
	CC	udm_set_partitionweight
分割重みのクリア	CXX	UdmZone::clearPartitionWeight
	CC	udm_clear_partitionweight

(7) 仮想セルの転送

機能	API	
仮想セルの転送	CXX	UdmModel::transferVirtualCells
	CC	udm_transfer_virtualcells

(8) 隣接節点（ノード）・要素（セル）

機能	API	
節点（ノード）の隣接節点（ノード）数の取得	CXX	UdmNode::getNumNeighborNodes
	CC	udm_getnum_neighbornodes
節点（ノード）の隣接節点（ノード）の取得	CXX	UdmNode::getNeighborNode
	CC	udm_get_neighbornodes
要素（セル）の隣接要素（セル）数の取得	CXX	UdmCell::getNumNeighborCells
	CC	udm_getnum_neighborcells
要素（セル）の隣接要素（セル）	CXX	UdmCell::getNeighborCell

機能	API	
の取得	CC	udm_get_neighborcells

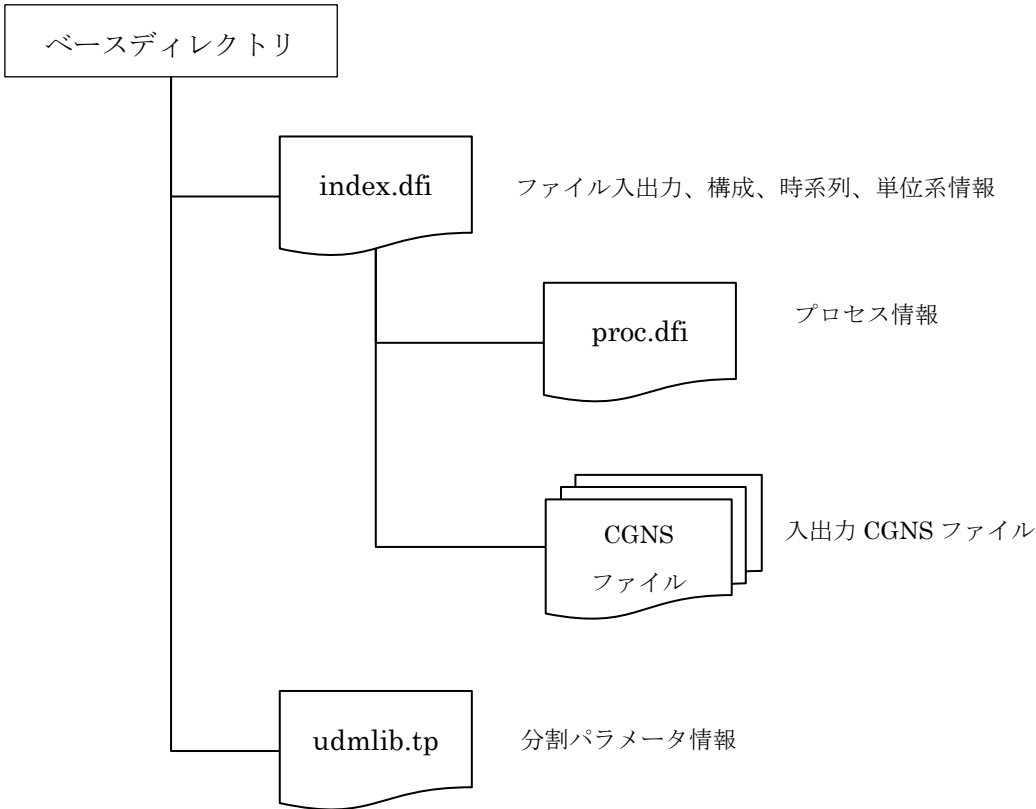
(9) ユーザ定義情報

機能	API	
ユーザ定義データ情報の取得	CXX	UdmUserDefinedDatas::getUserDataInfo
	CC	udm_user_getinfo
ユーザ定義データの取得	CXX	UdmUserDefinedDatas::getUserDataArray
	CC	udm_user_getdata
ユーザ定義データの設定	CXX	UdmUserDefinedDatas::setUserData
	CC	udm_user_setdata
ユーザ定義データの削除	CXX	UdmUserDefinedDatas::removeUserData
	CC	udm_user_remove

6. ファイル仕様

UDM ライブラリにて使用するファイルの使用について説明します。

UDM ライブラリのファイル構成は以下のようになります。



UDM ライブラリファイル構成

(注) ベースディレクトリは、読み込みを行った index.dfi のディレクトリとなります。

ファイル名	説明
index.dfi (任意)	ファイル入出力等の基本情報
proc.dfi (任意)	プロセス情報 ファイル名は index.dfi に記述
udmlib.tp (固定)	分割パラメータ
CGNS ファイル (任意)	入出力 CGNS ファイル CGNS ファイル名、構成は index.dfi に定義

6.1 入出力設定ファイル : index.dfi

入出力設定ファイル(index.dfi)はファイル情報(FileInfo), ファイルパス情報(FilePath), 単位系(Unit), 時系列データ(TimeSlice) の4つのブロックで構成されています。

ブロック	ラベル	値	オプション (default)	項目名
FileInfo	ファイル情報の設定項目を記述します。			
	DFIType	"Uns"	Option default="Uns"	DFI 種別 "Uns"固定
	DirectoryPath	相対パス 絶対パス	Option default="."/	出力フィールドデ ータディレクトリ
	TimeSliceDirectory	on off	Option default="off"	時刻ディレクトリ 作成オプション
	Prefix	ベースファイ ル名	Request	ベースファイル名
	FileFormat	"cgns"	Option default="cgns"	ファイルフォーマ ット
	FieldFilenameFormat	step_rank rank_step	Option default="step_rank"	ファイル命名書式
	FileCompositionType	IncludeGrid ExcludeGrid AppendStep EachStep GridConstant GridTimeSlice	Option default= @list("IncludeGrid", "EachStep")	CGNS ファイル構 成タイプ
FilePath	ファイルパス情報の設定項目を記述します。			
	Process	proc.dfi ファイ ル名	proc.dfi	proc.dfi ファイル名
UnitList	単位系情報の設定項目を記述します。			
[単位系名称]	単位系の名称を定義します。			単位系名称
	Unit	単位 (文字列)	Request	単位
	Reference	基準値 (実数)	Option	基準値
	Difference	差分値 (実数)	Option	差分値
TimeSlice	時系列情報の設定項目を記述します。			
	Slice{@}	ステップ番号データ		ステップ情報

ブロック	ラベル	値	オプション (default)	項目名
	Step	出力ステップ 番号（整数）	Request	出力ステップ番号
	Time	出力時間 （実数）	Request	出力時間
	AverageStep	平均ステップ 番号（整数）	Option	平均ステップ番号
	AverageTime	平均時間 （実数）	Option	平均時間
FlowSolution	物理量の設定項目を記述します。			
[物理変数名]	Feild(DataArray_t)：物理データリストの設定を記述します。			
	GridLocation	Vertex CellCenter	Option default="Vertex"	物理量を適用する 位置を記述します。 "Vertex"= ノード （頂点） "CellCenter"= 要素 中央
	DataType	Integer LongInteger RealSingle RealDouble	Option default="RealSingle"	データ型
	VectorType	Scalar Vector Nvector	Option	データ数 Scalar= スカラ型 (1D) Vector=ベクトル型 (3D) Nvector=N 成分型 (N)
	NvectorSize	整数	Option	データ成分数 VectorType=Nvect or の時のみ有効
	Constant	true false	Option default=false	固定物理量フラグ

6.1.1 FileInfo：ファイル情報の設定項目

ファイルの入出力、構成を定義します。

ブロック	ラベル	値	オプション (default)	項目名
FileInfo	ファイル情報の設定項目を記述します。			
	DFIType	"Uns"	Option default="Uns"	DFI 種別 "Uns"固定
	DirectoryPath	相対パス 絶対パス	Option default="."/	出力フィールドデータディレクトリ
	TimeSliceDirectory	on off	Option default="off"	時刻ディレクトリ 作成オプション
	Prefix	ベースファイル名	Request	ベースファイル名
	FileFormat	"cgns"	Option default="cgns"	ファイルフォーマット
	FieldFilenameFormat	step_rank rank_step	Option default="step_rank"	ファイル命名書式
	FileCompositionType	IncludeGrid ExcludeGrid AppendStep EachStep GridConstant GridTimeSlice	Option default= @list("IncludeGrid", "EachStep")	CGNS ファイル構成タイプ

(1) DFIType：DFI 種別

"Uns"のみサポートします。

(2) DirectoryPath：フィールドデータディレクトリ

フィールドデータディレクトリの相対、絶対パスを記述します。

(3) TimeSliceDirectory：時刻ディレクトリ作成オプション

時刻、ステップ反復出力のディレクトリの作成の有無、書式を記述します。

"off"：時刻、ステップ反復出力のディレクトリを作成しません。

"on"：時刻、ステップ反復出力のディレクトリを作成します。

作成ディレクトリは 10 桁のステップ数のディレクトリとなります。

例："00000000100" = 100 ステップの出力ディレクトリ

(4) **Prefix** : ベースファイル名

入出力ファイルのベースファイル名となります。

(5) **FileFormat** : ファイルフォーマット

ファイル形式を記述します。

UDMlib では"cgns"のみサポートします。

(6) **FieldFilenameFormat** : ファイル命名書式

ファイルの命名書式を記述します。

"step_rank" :

並列実行 = [Prefix]_[ステップ番号:10 桁]_id[RankID:6 桁].[ext]

逐次実行 = [Prefix]_[ステップ番号:10 桁].[ext]

"rank_step" :

並列実行 = [Prefix]_id[RankID:6 桁]_[ステップ番号:10 桁].[ext]

逐次実行 = [Prefix]_[ステップ番号:10 桁].[ext]

(7) **FileCompositionType** : CGNS ファイル構成タイプ

入出力ファイルの構成タイプを記述します。複数記述が可能です。

CGNS:GridCoordinates と CGNS:FlowSolution の出力方法	
IncludeGrid (デフォルト)	CGNS:GridCoordinates と CGNS:FlowSolution を 1 つのファイルに出力します。 ExcludeGrid と同時に使用することはできません。
ExcludeGrid	CGNS:GridCoordinates と CGNS:FlowSolution を別ファイルに出力します。 IncludeGrid と同時に使用することはできません。
CGNS:FlowSolution のステップ出力方法	
AppendStep	CGNS:FlowSolution をステップ毎に追加して 1 つのファイルに出力します。 EachStep と同時に使用することはできません。

EachStep (デフォルト)	CGNS:FlowSolution をステップ毎に別ファイルにします。 AppendStep と同時に使用することはできません。
CGNS:GridCoordinates の時系列出力方法	
GridConstant (デフォルト)	CGNS:GridCoordinates は最初の 1 回のみ出力を行います。 ノード座標が時系列毎に変化するしない、又は変位データは物理量として出力を行い CGNS:GridCoordinates の座標データは変化しない場合に使用します。 GridTimeSlice と同時に使用することはできません。
GridTimeSlice	CGNS:GridCoordinates を時系列毎に出力を行う。 ノード座標が時系列毎に変化する場合に使用します。 出力前に CGNS:GridCoordinates の座標データを更新する必要があります。 GridConstant と同時に使用することはできません。

FileCompositionType (CGNS ファイル構成タイプ) の組合せにより以下の動作となります。



(1)			(C) Constant 物理量データ	
IncludeGrid ExcludeGrid	AppendStep EachStep	GridConstant GridTimeSlice	出力ファイル	出力動作
IncludeGrid	AppendStep	GridConstant	CGNS File : 1	1 つの CGNS ファイルに 1 つの CGNS:GridCoordinates と 時系列の CGNS:FlowSolution を出 力します。 デフォルト出力設定です。

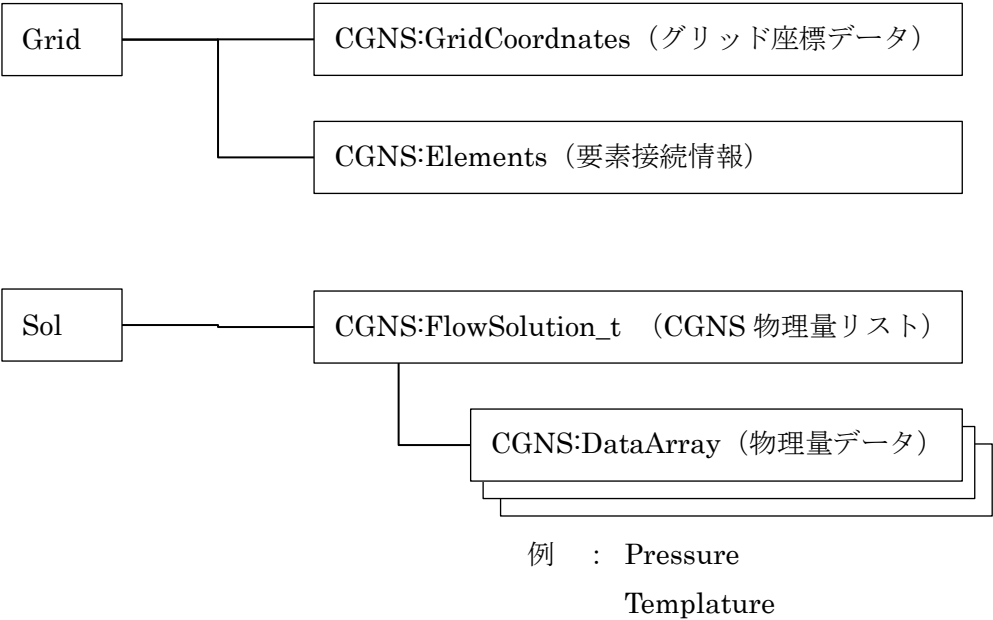
IncludeGrid ExcludeGrid	AppendStep EachStep	GridConstant GridTimeSlice	出力ファイル	出力動作
		<div> <div>Link</div> <div>出力なし</div> </div> <div> <div>1座標データ</div> <div>Grid</div> <div>Sol1(C) Sol2 Sol3</div> <div>時系列物理データ</div> </div>		
		GridTimeSlice	CGNS File : 1	1つのCGNSファイルに 時系列の CGNS:GridCoordinatesと CGNS:FlowSolutionを出力しま す。
		<div> <div>Link</div> <div>出力なし</div> </div> <div> <div>時系列座標データ</div> <div>Grid1 Grid2 Grid3</div> <div>Sol1(C) Sol2 Sol3</div> <div>時系列物理データ</div> </div>		

IncludeGrid ExcludeGrid	AppendStep EachStep	GridConstant GridTimeSlice	出力ファイル	出力動作
IncludeGrid	EachStep	無効	Step File : * Link File : 1	ステップ毎の CGNS ファイルに CGNS:GridCoordinates と 1 ステップの CGNS:FlowSolution を出力します。 また、すべてのステップの CGNS ファイルへのリンクファイルを作成します。
<div> <div>グリッドデータリンク x 3 時系列物理データリンク x 3</div> <pre> graph LR Link[Link] --- Grid1[Grid1 ----- Sol1(C)] Link --- Grid2[Grid2 ----- Sol2] Link --- Grid3[Grid3 ----- Sol3] </pre> </div>				

IncludeGrid ExcludeGrid	AppendStep EachStep	GridConstant GridTimeSlice	出力ファイル	出力動作
ExcludeGrid	AppendStep	GridConstant	Grid File : 1 Solution File : 1 Link File : 1	CGNS:GridCoordinates は別ファイルとして出力します。 1つのCGNSファイルに全ステップのCGNS:FlowSolutionを出力します。 CGNS:GridCoordinates とCGNS:FlowSolution をリンクしたリンクファイルを作成します。
		<div style="text-align: center;"> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;">Link</div> <div style="margin-top: 10px;"> グリッドデータリンク x 1 時系列物理データリンク x 3 </div> <div style="display: flex; justify-content: center; align-items: center; margin-top: 10px;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Grid</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Sol1(C)</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Sol2</div> <div style="border: 1px solid black; padding: 5px;">Sol3</div> </div> </div>		
		GridTimeSlice	Grid File : 1 Solution File : 1 Link File : 1	CGNS:GridCoordinates は別ファイルとして出力します。 1つのCGNSファイルに全ステップのCGNS:FlowSolutionを出力します。 CGNS:GridCoordinates ファイルにCGNS:FlowSolutionのリンクを追加します。
		<div style="text-align: center;"> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;">Link</div> <div style="margin-top: 10px;"> グリッドデータリンク x 3 時系列物理データリンク x 3 </div> <div style="display: flex; justify-content: center; align-items: center; margin-top: 10px;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Grid1</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Grid2</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Grid3</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Sol1(C)</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Sol2</div> <div style="border: 1px solid black; padding: 5px;">Sol3</div> </div> </div>		

IncludeGrid	AppendStep	GridConstant	出力ファイル	出力動作
ExcludeGrid	EachStep	GridTimeSlice		
ExcludeGrid	EachStep	GridConstant	Grid File : 1 Solution File : * Link File : 1	CGNS:GridCoordinates, CGNS:FlowSolution は別ファイル として出力します。 時系列毎の CGNS:FlowSolution を 出力します。
<div> <div>Link</div> <div> グリッドデータリンク x 1 時系列物理データリンク x 3 </div> <div> <div>Grid</div> <div> <div>Sol1(C)</div> <div>Sol2</div> <div>Sol3</div> </div> </div> </div>				
		GridTimeSlice	Grid File : * Solution File : * Link File : 1	CGNS:GridCoordinates, CGNS:FlowSolution は別ファイル として出力します。 時 系 列 毎 の CGNS:GridCoordinates, CGNS:FlowSolution を出力しま す。
<div> <div>Link</div> <div> グリッドデータリンク x 3 時系列物理データリンク x 3 </div> <div> <div>Grid1</div> <div>Grid2</div> <div>Grid3</div> <div> <div>Sol1(C)</div> <div>Sol2</div> <div>Sol3</div> </div> </div> </div>				

上記の Grid, Sol はそれぞれ座標データ、物理量データを示すものですが、以下の CGNS ノードを出力します。



6.1.2 FilePath：ファイルパス情報

ファイルパス情報を定義します。

ブロック	ラベル	値	オプション (default)	項目名
FilePath	ファイルパス情報の設定項目を記述します。			
	Process	proc.dfi ファイル名	proc.dfi	proc.dfi ファイル名

(1) Process：proc.dfi ファイル名

proc.dfi ファイル名を相対パス、絶対パスにて記述します。

6.1.3 UnitList：単位系情報

単位系の情報を定義します。

UDMLib では、単位系情報：UnitList の設定項目を返す API を提供するのみとします。

ブロック	ラベル	値	オプション (default)	項目名
------	-----	---	--------------------	-----

ブロック	ラベル	値	オプション (default)	項目名
UnitList	単位系情報の設定項目を記述します。			単位系情報
[単位系名称]		Length Velocity Pressure Temperature	Request	単位系名称
	Unit	単位（文字列）	Request	単位
	Reference	基準値（実数）	Option	基準値
	Difference	差分値（実数）	Option	差分値

(1) [単位系名称]

単位系の名称を記述します。

(例) Length
Velocity
Pressure
Temperature

(2) Unit : 単位

単位系の単位（文字列）を記述します。

(3) Reference : 基準値

単位の基準値（実数）を記述します。

(4) Difference : 差分値

単位の差分値（実数）を記述します。

6.1.4 TimeSlice : 時系列情報

CGNS ファイルの出力時に時系列情報を TimeSlice ブロックに出力します。

ブロック	ラベル	値	オプション (default)	項目名
TimeSlice	時系列情報の設定項目を記述します。			時系列情報
Slice{@}	ステップ番号データ			

ブロック	ラベル	値	オプション (default)	項目名
	Step	出力ステップ番号 (整数)	Request	出力ステップ番号
	Time	出力時間 (実数)	Request	出力時間
	AverageStep	平均ステップ番号 (整数)	Option	平均ステップ番号
	AverageTime	平均時間 (実数)	Option	平均時間

(1) Slice : ステップ番号データ

1 ステップ毎に 1Slice を出力します。

(2) Step : 出力ステップ番号

出力ステップ番号 (整数) を出力します。

(3) Time : 出力ステップ時間

出力ステップ時間 (実数) を出力します。

(4) AverageStep : 平均ステップ番号

平均ステップ番号 (整数) を出力します。

CGNS 出力時に平均ステップ番号、時間を指定した場合のみ出力します。

(5) AverageTime : 平均時間

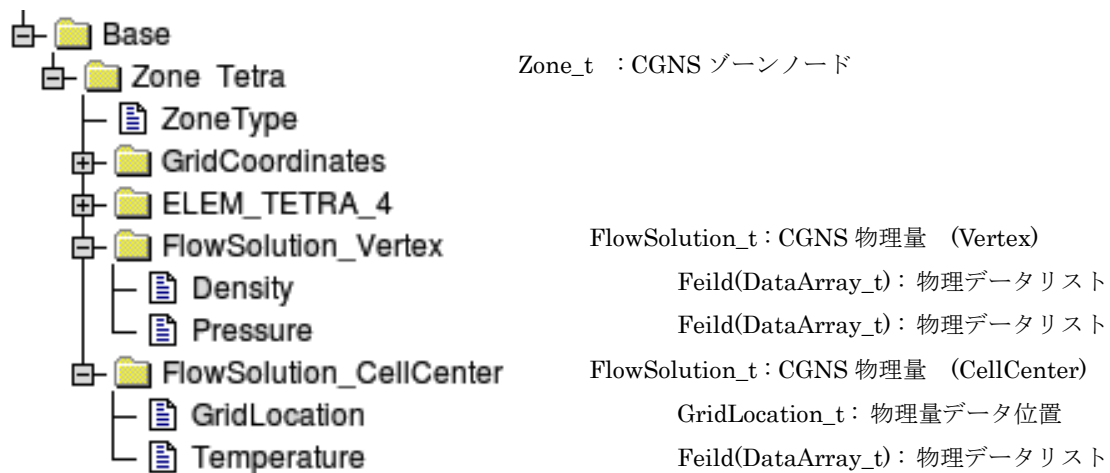
平均ステップ時間 (実数) を出力します。

CGNS 出力時に平均ステップ番号、時間を指定した場合のみ出力します。

6.1.5 FlowSolution : 物理量定義情報

物理量を定義します。

CGNS:FlowSolution_t (CGNS 物理量) のデータ構造は以下の様になっています。



index.dfi の物理変数情報：FlowSolution の設定項目により CGNS:FlowSolution_t の入出力を行います。

FlowSolution に記述されていない物理量は、節点（ノード）、要素（セル）に物理量値を設定することはできません。また、CGNS ファイルへの出力は行いません。

以下、FlowSolution の設定項目の説明を行います。

ブロック	ラベル	値	オプション (default)	項目名
FlowSolution	物理量の設定項目を記述します。			
[物理変数名]	Feild(DataArray_t)：物理データリストの設定を記述します。			
	GridLocation	"Vertex" or "CellCenter"	Option default="Vertex"	物理量を適用する位置 を記述します。 "Vertex"=ノード（頂点） "CellCenter"=要素中央
	DataType	Integer LongInteger RealSingle RealDouble	Option default="RealSingle"	データ型
	VectorType	Scalar Vector Nvector	Option	データ数 Scalar=スカラ型 (1D) Vector=ベクトル型(3D) Nvector=N 成分型(N)
	NvectorSize	整数	Option	データ成分数

ブロック	ラベル	値	オプション (default)	項目名
				VectorType=Nvector の 時のみ有効
	Constant	true false	Option default=false	固定物理量フラグ

(1) FlowSolution

CGNS ファイルから入出力を行う物理量変数を定義します。

この FlowSolution に記述された物理量変数、及びユーザプログラムからの FlowSolution の設定された物理量変数に対して、CGNS ファイルから入出力を行います。

入力 CGNS ファイルに設定された物理量変数が存在しない場合はエラーとします。

(2) [物理変数名]

個別の物理量データ ("Density", "Pressure", "Temperature"等) の物理変数名称を記述します。

重複名称は記述できません。

(注) 物理変数名称は最大 32 文字以内とします。

(3) GridLocation : 物理量定義位置

"Vertex"又は"CellCenter"の物理量を適用する位置を記述します。

Vertex : 頂点 (ノード)

CellCenter : 要素 (セル) 中央

CGNS では、CGNS:FlowSolution_t (CGNS 物理量) 単位で記述します。

UDMLib では、GridLocation に対応した CGNS:FlowSolution_t を命名規約により作成します。

(4) DataType : データ型

物理量データのデータ型を記述します。

Integer : 4byte 整数型

LongInteger : 8byte 整数型

RealSingle : 単精度浮動小数点 (4byte)

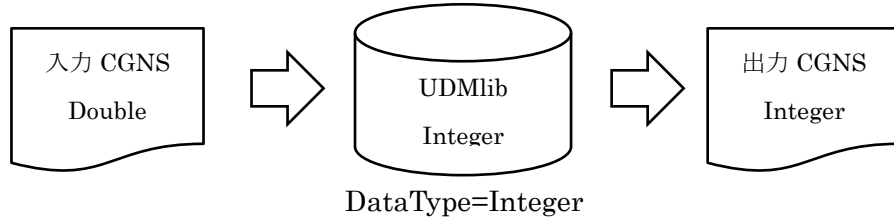
RealDouble : 倍精度浮動小数点 (8byte)

データ型は UDMLib の内部のデータ型となります。

入力 CGNS ファイルのデータ型から UDMLib 内部データ型(DataType)に変換します。

出力 CGNS ファイルは UDMLib 内部データ型(DataType)として出力します。

(例) 入力 CGNS : データ型 double から UDMLib : データ型 Integer に変換



DataType はオプション設定項目であり、記述がない場合は入力 CGNS ファイルのデータ型を UDMLib のデータ型とします。

(5) VectorType : ベクトル型

物理量データのベクトル型を記述します。

Scalar : スカラデータ型 (データ数=1) : デフォルト

Vector : ベクトル型 (データ数=3)

Nvector : N 成分型 (データ数=[NvectorSize])

CGNS:Feild(DataArray_t) : 物理データリストには、スカラデータしか記述できません。

以下の命名規則により CGNS:Feild(DataArray_t)から入出力を行います。

VectorType : ベクトル型	CGNS 命名規則
Scalar	[物理変数名]をそのまま使用します。
Vector	[物理変数名]の末尾に X,Y,Z を付加します。 (例) [物理変数名] = Pressure CGNS:Feild = PressureX, PressureY, PressureZ
Nvector	[物理変数名]の末尾に成分数(N)を 1 ~ 付加します。 (例) [物理変数名] = Pressure, 5 成分 CGNS:Feild = Pressure1, ~ Pressure5

(6) NvectorSize : 成分数

物理量データの N 成分型の成分数を記述します。

VectorType=Nvector のみ有効です。

(7) Constant : 固定値

物理量データが固定値であることを示します。

UDMlib では、Constant に対応した CGNS:FlowSolution_t を命名規約により作成します。

固定値物理量データは時系列データとしては出力せず、最初のステップのみ出力します。

UDMlib では、FlowSolution の設定項目の GridLocation, Constant に対応した命名規則による CGNS:FlowSolution_t を作成します。

GridLocation	Constant	CGNS:FlowSolution_t 名称	CGNS:FlowSolution_t 名称（時系列リンク名）
Vertex	false	"UdmSol_Node"	"UdmSol_Node_[%10d]"
CellCenter		"UdmSol_Cell"	"UdmSol_Cell_[%10d]"
Vertex	true	"UdmSol_Node_Const"	"UdmSol_Node_Const"
CellCenter		"UdmSol_Cell_Const"	"UdmSol_Cell_Const"

CGNS リンクファイルの時系列リンク名称は末尾にステップ数を 0 埋め 10 桁で付加した CGNS:FlowSolution_t 名称となります。

Constant 物理量データは、時系列物理量データではありませんのでステップ数の付加はありません。

6.2 プロセス情報ファイル：proc.dfi

CGNS ファイルの出力時にプロセス情報を出力します。

計算領域全体の情報と各プロセスの領域情報を出力します。

ブロック	ラベル	値	オプション (default)	項目名
Domain	ドメイン情報の設定項目を記述します。			ドメイン情報
	CellDimension	1 2 3	default=3	非構造格子モデルの次元数 (=1D, 2D, 3D) を記述します。
	VertexSize	整数		全体の節点 (ノード) 数を記述します。
	CellSize	整数		全体の要素 (セル) 数を記述します。
MPI	並列実行情報の設定項目を記述します。			並列情報
	NumberOfRank	整数	Request	並列実行数を記述します。
	NumberOfGroup	整数	default=1	並列実行グループ数
Process	プロセスの領域情報の設定項目を記述します。			プロセス情報
Rank{@}	プロセス別の領域情報を記述します。			プロセス別情報
	ID	整数		ランク番号
	VertexSize	整数		プロセスの領域の節点 (ノード) 数を記述します。
	CellSize	整数		プロセスの領域の要素 (セル) 数を記述します。

6.2.1 Domain：ドメイン情報

計算空間全体の領域情報を記述します。

ブロック	ラベル	値	オプション (default)	項目名
Domain	ドメイン情報の設定項目を記述します。			ドメイン情報
	CellDimension	1 2 3	default=3	非構造格子モデルの次元数 (=1D, 2D, 3D) を記述します。

ブロック	ラベル	値	オプション (default)	項目名
	VertexSize	整数		全体の節点（ノード）数を記述します。
	CellSize	整数		全体の要素（セル）数を記述します。

(1) CellDimension：次元数

非構造格子モデルの次元数（=1D, 2D, 3D）を整数で記述します。

(2) VertexSize：全体節点（ノード）数

すべてのプロセスの節点（ノード）数の合計を記述します。

共通節点（ノード）は重複加算しないモデル全体の節点（ノード）数となります。

(3) CellSize：全体要素（セル）数

すべてのプロセスの要素（セル）数の合計を記述します。

6.2.2 MPI：並列情報

並列実行の情報を記述します。

ブロック	ラベル	値	オプション (default)	項目名
MPI	並列実行情報の設定項目を記述します。			並列情報
	NumberOfRank	整数	Request	並列実行数を記述します。
	NumberOfGroup	整数	default=1	並列実行グループ数

(1) NumberOfRank：並列実行数

MPI プロセス数を記述します。

(2) NumberOfGroup：並列実行グループ数

MPI グループのプロセス数を記述します。

6.2.2 Process：プロセス情報

プロセスの領域情報を記述します。

ブロック	ラベル	値	オプション (default)	項目名
Process	プロセスの領域情報の設定項目を記述します。			プロセス情報
Rank{@}	プロセス別の領域情報を記述します。			プロセス別情報
	ID	整数		ランク番号
	VertexSize	整数		プロセスの領域の節点（ノード） 数を記述します。
	CellSize	整数		プロセスの領域の要素（セル）数 を記述します。

(1) ID：ランク番号

MPI ランク番号を記述します。

(2) VertexSize：節点（ノード）数

プロセス領域の節点（ノード）数を記述します。

仮想節点（ノード）は含まれません。

(3) CellSize：要素（セル）数

プロセス領域の要素（セル）数数を記述します。

仮要素（セル）数は含まれません。

6.3 分割パラメータファイル : udmlib.tp

Zoltan 分割パラメータを定義します。

分割実行後、実行時に設定したパラメータを udmlib.tp に出力します。

ブロック	ラベル	値	説明
UDMLib	UDMLib の設定項目を記述します。		
partition	分割情報、分割パラメータを記述します。		
	LB_METHOD	HYPERGRAPH GRAPH	分割方法 default=HYPERGRAPH
	GRAPH_PACKAGE	PHG	グラフ分割の分割パッケージ (LB_METHOD=GRAPH のみ有効) default=PHG
	LB_APPROACH	REPARTITION PARTITION REFINE	再分配方法 default = REPARTITION
	DEBUG_LEVEL	0~10 default=1	デバッグ出力レベル
	[Zoltan 分割パラメータ]	[設定値]	任意

(1) LB_METHOD : 分割方法

Zoltan 分割の分割方法を記述します。

"HYPERGRAPH"又は"GRAPH"を記述可能です。

(2) GRAPH_PACKAGE : 分割パッケージ

LB_METHOD=GRAPH のみ有効で、"PHG"のみサポートします。

(3) LB_APPROACH : 再分配方法

Zoltan 分割の再分配方法を記述します。

(4) DEBUG_LEVEL : デバッグ出力レベル

Zoltan 分割時の Zoltan ライブラリが出力するデバッグメッセージを制御します。

DEBUG_LEVEL=0 の場合、デバッグメッセージは出力しません。

(5) [Zoltan 分割パラメータ]

Zoltan ライブラリには、LB_METHOD（分割方法）に応じた様々なパラメータが存在します。

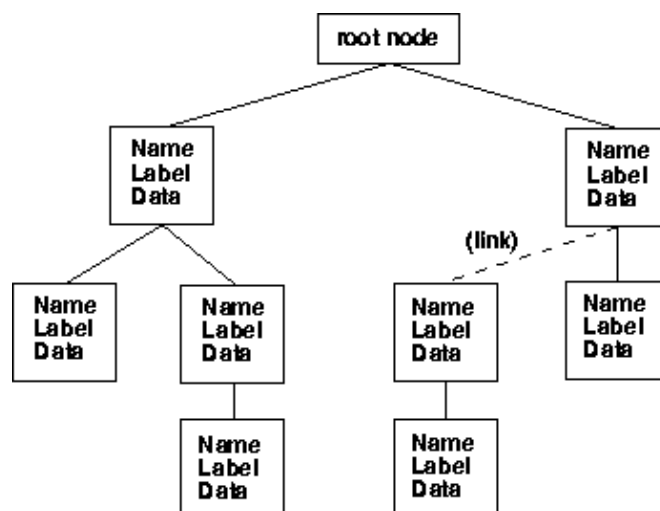
そのパラメータを任意に設定できます。

パラメータについては、Zoltan ライブラリのマニュアルを参照してください。

6.4 CGNS ファイル

CGNS ファイルは、1 つまたは複数の CGNS ファイルから構成される CGNS データベースからなります。

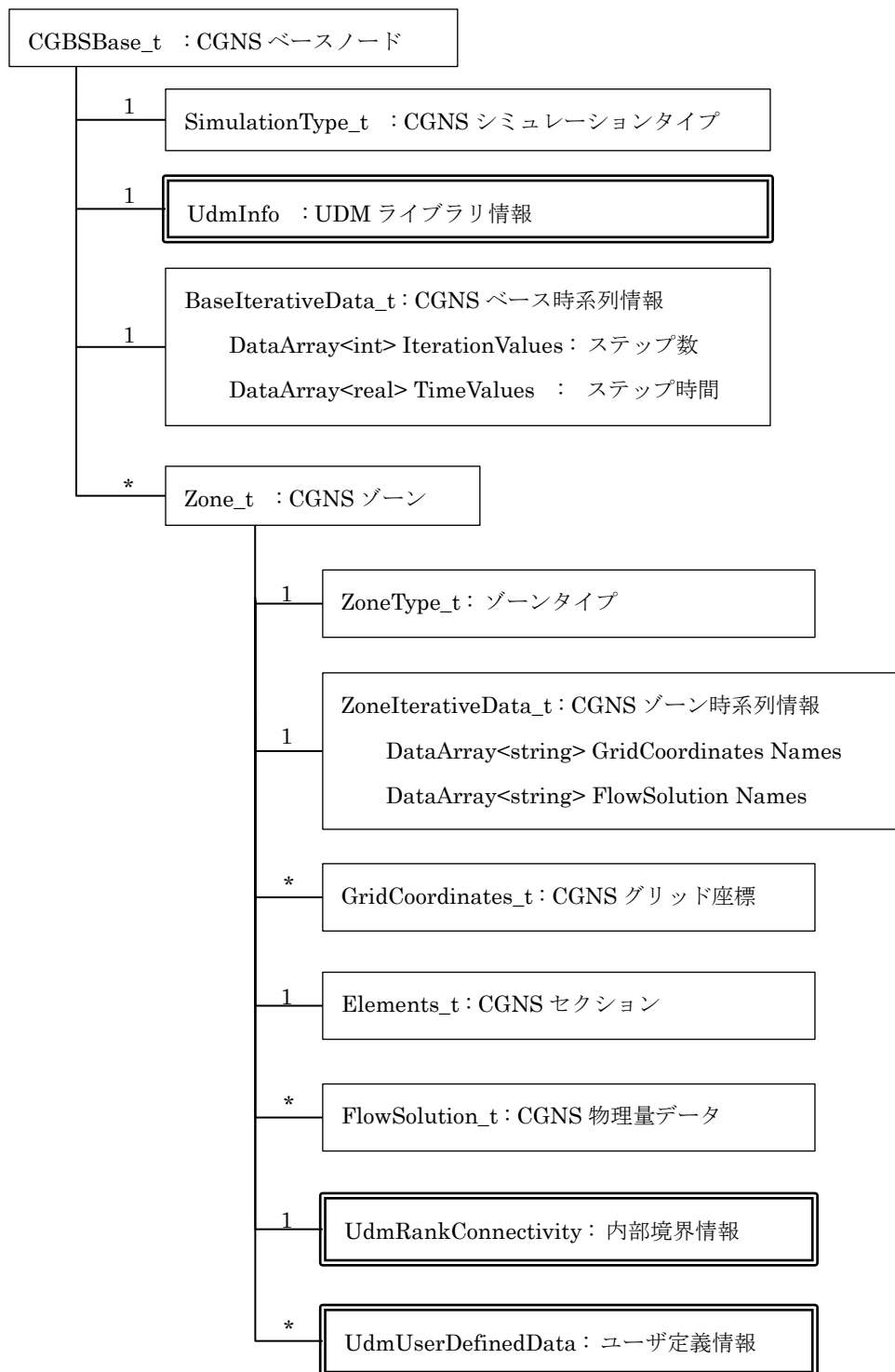
最上位の CGNS ノードは「CGNS ルートノード」です。CGNS ルートノード配下の各 CGNS ノードは、名前とラベルの両方によって定義され、情報またはデータを含んでも含まなくてもよい。各 CGNS ノードは、1 つまたは複数の「子」CGNS ノードへの「親」とすることができる。CGNS ノードは、別の CGNS ファイルを子 CGNS ノードとしてリンクを持つことができます。リンクはツリー上に存在しているようにアクセスを可能です。CGNS のツリー状の構造の例を以下に示します。




ユーザーが容易に特定の情報にアクセスできるように CGNS ファイルの構造は、Standard Interface Data Structures (SIDS)にて規定されています。

CGNS ファイルはバイナリファイルであり、ユーザが内容を表示することはできません。しかし、ユーティリティの CGNSview によって、CGNS ファイルの内容を見ることができます。

UDM ライブラリでは、CGNS の SIDS に規定されている CGNS ノードの一部のみサポートしています。また、UDM ライブラリ用の拡張の CGNS ノードを持ちます。



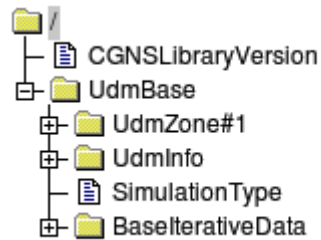
 : UDM ライブラリ拡張 CGNS ノード

6.4.1 CGBSBase_t : CGNS ベースノード

CGNS データベースのルートノード配下に 1 つのみ存在する CGNS ノードです。

UDM ライブラリでは、CGNS ベースノード名は"UdmBase"となります。

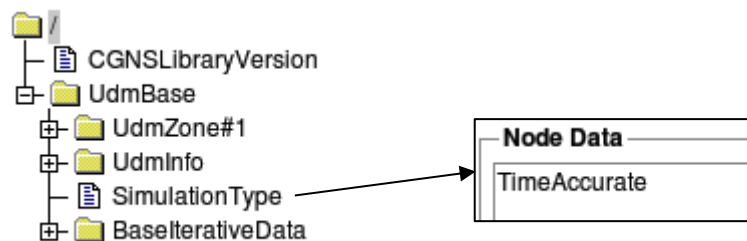
(出力例)



6.4.2 SimulationType_t : CGNS シミュレーションタイプ

シミュレーションタイプを記述します。"TimeSlice"と"NonTimeSlice"が記述可能ですが、UDM ライブラリでは、"TimeSlice"固定です。

(出力例)



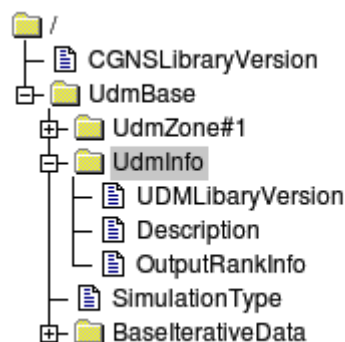
6.4.3 UdmInfo : UDM ライブラリ情報

UDM ライブラリの拡張 CGNS ノードです。

以下の子 CGNS ノードに UDM ライブラリ、出力モデルの基本情報を記述します。

子 CGNS ノード名	データ型	説明
UDMLibraryVersion	char[32]	UDM ライブラリのバージョン
Description	char[154]	UDM ライブラリのライセンス情報
OutputRankInfo	int[2]	出力 MPI ランク情報 データ = { [MPI プロセス数] [MPI ランク番号] }

(出力例)

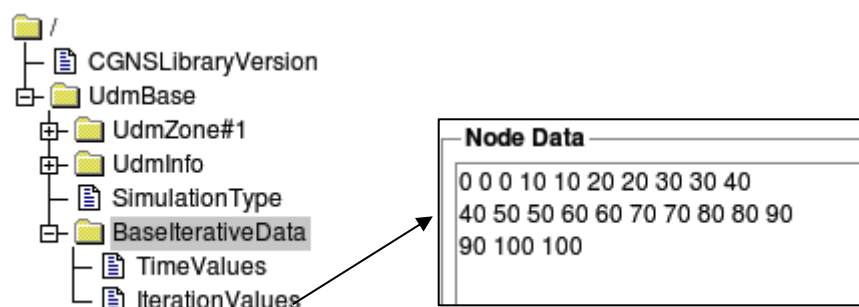


6.4.4 BaselIterativeData_t : CGNS ベース時系列情報

出力モデルの時系列分のステップ番号、ステップ時間を以下の子 CGNS ノードに出力します。

子 CGNS ノード名	データ型	説明
TimeValues	float[]	ステップ時間
IterationValues	int[]	ステップ番号

(出力例)



6.4.5 Zone_t : CGNS ゾーン

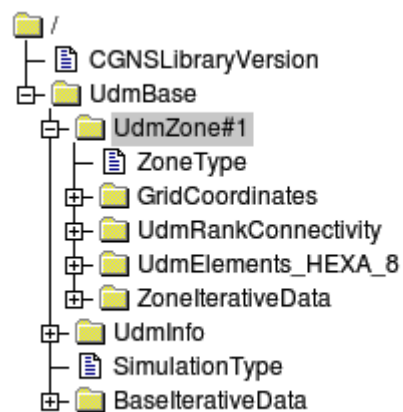
CGNS ゾーンを記述します。

UDM ライブラリでは、CGNS ゾーンノード名は"UdmZone#1"となります。

CGNS ゾーンノードのデータ属性として、節点（ノード）数と要素（セル）数を持ちます。

データ属性	データ型	説明
節点（ノード）数	long	GridCoordinates_t : CGNS グリッド出力の節点（ノード）数と一致します。
要素（セル）数	long	Elements_t : CGNS セクション出力の要素（セル）数と一致します。

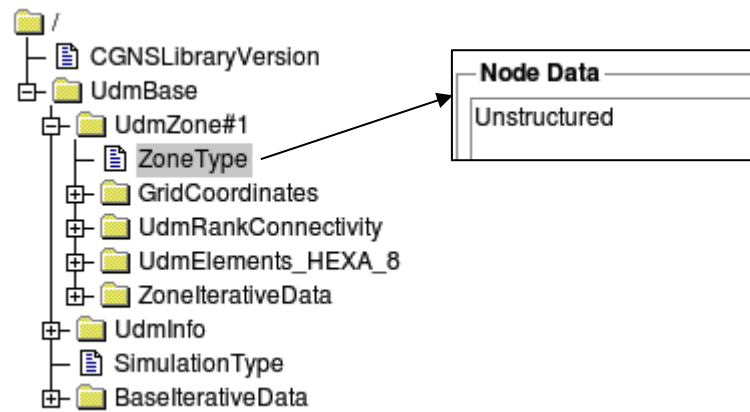
(出力例)



6.4.6 ZoneType_t: ゾーンタイプ

ゾーンタイプを記述します。"Unstructured"と"Structured"が記述可能ですが、UDM ライブラリでは、"Unstructured"のみサポートします。

(出力例)



6.4.7 ZoneliterativeData_t: CGNS ゾーン時系列情報

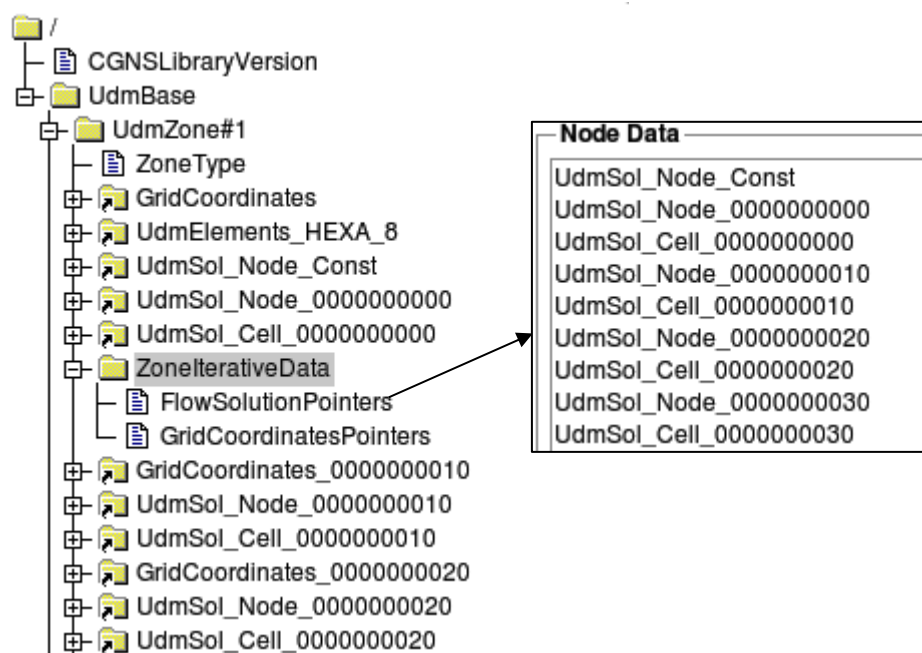
時系列出力の GridCoordinates_t (CGNS グリッド)、FlowSolution_t (CGNS 物理量) の CGNS ノード名を以下の子 CGNS ノードに出力します。

子 CGNS ノード名	データ型	説明
FlowSolutionPointers	char[32][]	FlowSolution_t (CGNS 物理量) の CGNS ノード名を時系列分出力します。
GridCoordinatesPointers	char[32][]	GridCoordinates_t (CGNS グリッド) の CGNS ノード名を時系列分出力します。

出力時系列数は BaseIterativeData_t (CGNS ベース時系列情報) の出力数と同じでなければなりません。

同一インデックス位置の CGNS ベース時系列情報が出力時のステップ番号、時間となります。

(出力例)



6.4.8 GridCoordinates_t : CGNS グリッド座標

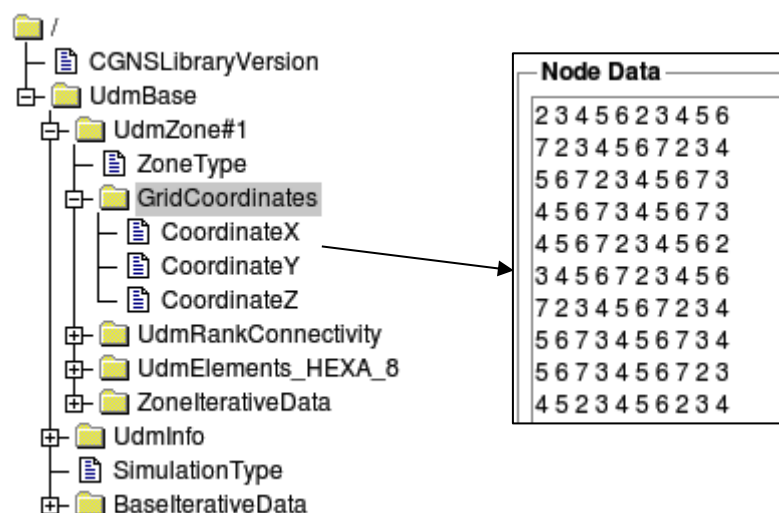
節点（ノード）のグリッド座標を出力します。

モデルの節点（ノード）座標が時系列により変化する場合は、複数出力することが可能です。

グリッド座標は X,Y,Z の 3 次元で表現し、以下の子 CGNS ノードに出力します。

子 CGNS ノード名	データ型	説明
CoordinateX	float[]	グリッド X 座標
CoordinateY	float[]	グリッド Y 座標
CoordinateZ	float[]	グリッド Z 座標

(出力例)



6.4.9 Elements_t: CGNS セクション

要素（セル）の構成節点（ノード）を出力します。

構成節点（ノード）は GridCoordinates_t（CGNS グリッド座標）のインデックス番号（1～）を記述します。

UDM ライブラリでは、CGNS セクションノード名は"UdmElements_[要素形状名]"となります。

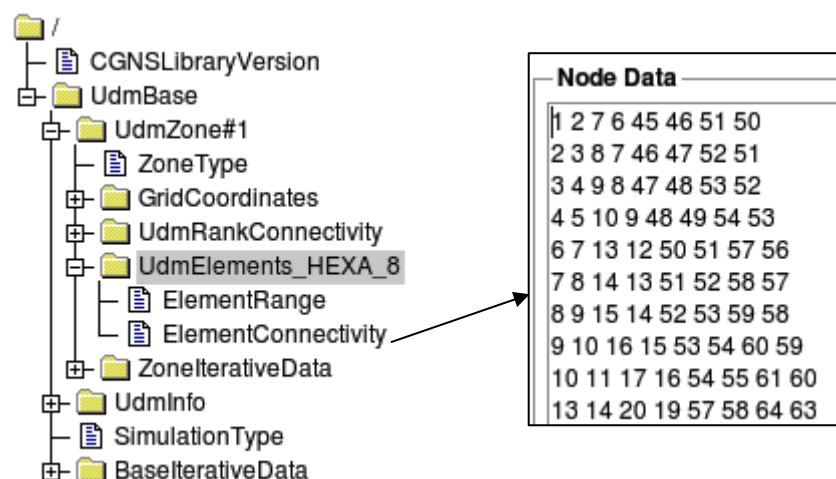
CGNS セクションノードのデータ属性として、要素形状番号を持ちます。

データ属性	データ型	説明
要素形状番号	int	要素（セル）の形状番号 TETRA_4 =10, PYRA_5 =12, PENTA_6 =14, HEXA_8 =17, MIXED =20

構成節点（ノード）を定義する為に以下の子 CGNS ノードを持ちます。

子 CGNS ノード名	データ型	説明
ElementRange	long[2]	要素（セル）のインデックス範囲（1 以上）
ElementConnectivity	long[]	構成節点（ノード）インデックス（1 以上）

（出力例）



6.4.10 FlowSolution_t : CGNS 物理量データ

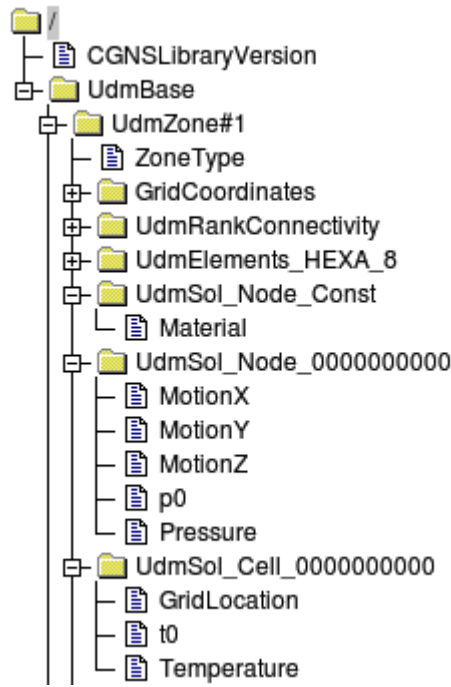
モデルの節点（ノード）、要素（セル）に設定された物理量を出力します。

FlowSolution_t（CGNS 物理量データ）は節点（ノード）、要素（セル）毎、時系列毎に出力します。

物理量の名前の子 CGNS ノードに物理量データを出力します。

子 CGNS ノード名	データ型	説明
[物理量の名前]	任意	物理量データ
GridLocation	char[10]	物理量の定義位置 "CellCenter" : 要素（セル） "Vertex" : 節点（ノード） デフォルト

（出力例）



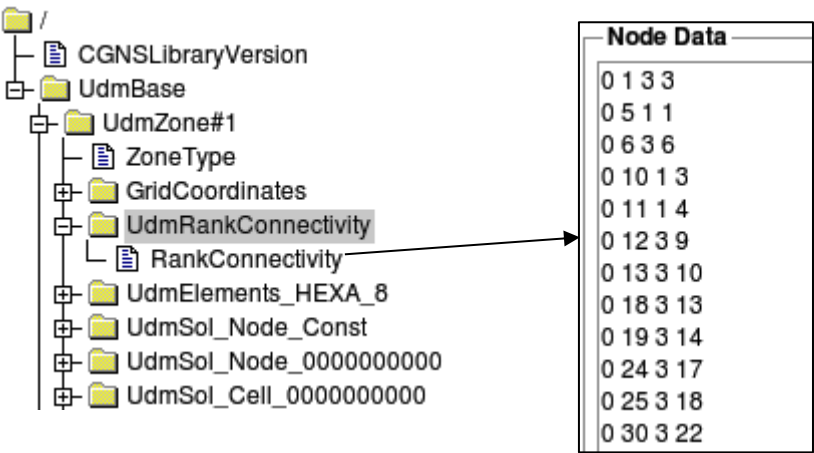
6.4.11 UdmRankConnectivity：内部境界情報

他の MPI ランクと接続している節点（ノード）の情報を出力します。

子 CGNS ノード（RankConnectivity）に接続情報を出力します。

子 CGNS ノード名	データ型	説明
RankConnectivity	int[4]	接続情報

(出力例)



接続情報は 1 つの接続節点（ノード）に対して 4 つの整数からなります。

接続情報[0]	自グローバル ID	自 MPI ランク番号
接続情報[1]		自ローカル番号
接続情報[2]	接続先グローバ	接続先 MPI ランク番号
接続情報[3]	ル ID	接続先ローカル番号

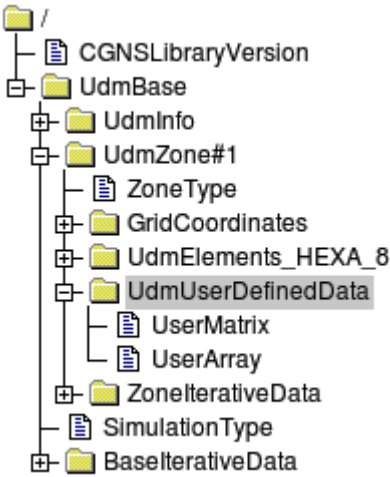
6.4.12 UdmUserDefinedData : ユーザ定義情報

ユーザ定義データを出力します。

ユーザ定義の名前の子 CGNS ノードにユーザデータを出力します。

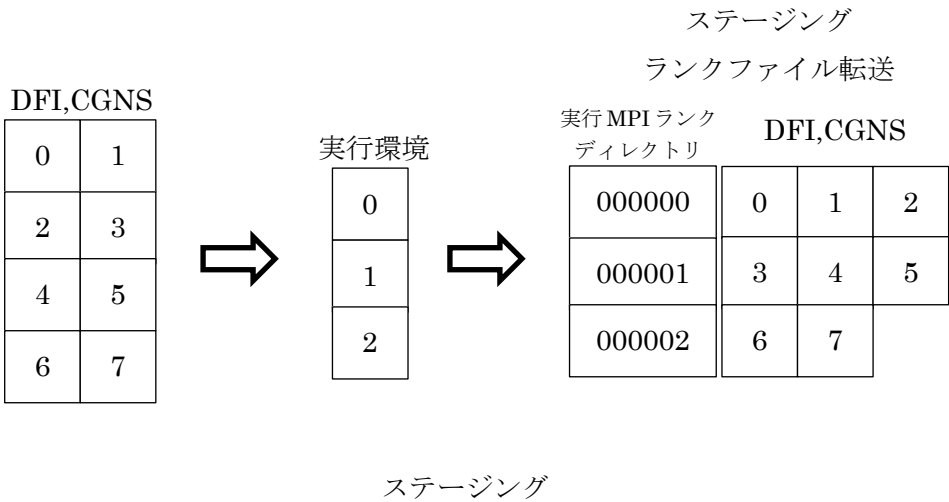
子 CGNS ノード名	データ型	説明
[ユーザ定義の名前]	任意	ユーザ定義データ

(出力例)



7. ステージングツール

ステージングツール `udm-frm`(UDM ライブラリ `File RankMapper`) は、大規模並列計算機でUDM ライブラリを使用する上で、各計算ノード(MPI ランク) 毎に必要なファイルを、ランク番号で命名したディレクトリにコピーするステージング対応用のプログラムです。



7.1 ステージングツールのビルド

`udm-frm` はログインノードで動作し、ステージング機構のあるマシンでの利用を対象としています。したがって、ステージング機構を持たない計算機環境には必要ありません。

`udm-frm` は、UDM ライブラリのビルド、インストールにより `$prefix/bin/udm-frm` の実行モジュールが作成されます。(「パッケージのビルド」参照)

しかし、フロントエンド端末の様な MPI ライブラリの無い環境では"`--disable-mpi`"オプションでビルドします。

```
$ ./configure --prefix=/home/user/udmlib_front ¥
               --with-cgns=/home/user/cgnslib ¥
               --with-tp=/home/user/textparser ¥
               --disable-mpi
$ make
$ make install
```

(注) CGNS ライブラリ、TextParser ライブラリは必須となります。

7.1.1 京フロントエンドでのステージングツールのビルド

京フロントエンドでのステージングツールのビルド方法について説明します。

必要なライブラリは以下となります。

ライブラリ名	機能説明
CGNS ライブラリ	数値流体力学(CFD)用の階層型データベース
TextParser	TextPaser 形式の ASCII ファイルの入出力
UDM ライブラリ	非構造格子モデルの MxN 分割ライブラリ

ステージングツール機能のみのビルドであり、MPI 環境はありませんので、Zontan ライブラリは必要ありません。

インストール先`--prefix`についてはユーザ環境に合わせて適時変更してください。

また、コンパイラは `gcc`, `g++` を使用するものとして、オプション設定は行いません。

他のコンパイラを使用する場合は、オプションの追加を行ってください。

(CGNS ライブラリ)

```
$ tar xzvf cgnslib_3.2.1.tar.gz
$ cd cgnslib_3.2.1/src
$ ./configure --prefix=/volumeXX/k9999/udm_project/local_nonempi/cgnslib_3.2.1
$ make
$ make install
```

(TextParser ライブラリ)

```
$ unzip TextParser-master.zip
$ cd TextParser-master
$ ./configure --prefix=/volumeXX/k9999/udm_project/local_nonempi/TextParser
$ make
$ make install
```

(UDM ライブラリ)

```
$ tar xzvf UDMLib-X.X.X.tar.gz
$ cd UDMLib-X.X.X
$ ../configure --prefix=/volumeXX/k9999/udm_project/local_nonempi/udmlib_X.X.X  ¥
               --with-cgns=/volumeXX/k9999/udm_project/local_nonempi/cgnslib_3.2.1  ¥
               --with-tp=/volumeXX/k9999/local_nonempi/TextParser  ¥
```

`--disable-mpi`

`$ make`

`$ make install`

"`--disable-mpi`"オプションを付加してください。

7.2 使用方法

udm-frm はコマンドを実行して使用します。

```
$ udm-frm --input INDEX_DFI --np N  OPTIONS.
```

以下の引数を指定します

引数	オプション	説明
-i, --input INDEX_DFI	必須	入力 index.dfi ファイル
-n, --np NP	必須	振分プロセス数
-o, --output [=OUTPUT_PATH]	省略可	出力ディレクトリ デフォルト = ./
-u, --with-udmlib [=UDMLIBTP_FILE]	省略可	udmlib.tp ファイル
-s, --step [=STEP_NO]	省略可	ファイルコピーステップ番号
-v --view	省略可	ファイルコピー表示
--version	省略可	バージョン情報表示
-h --help	省略可	ヘルプ出力

(1) -i, --input INDEX_DFI : 必須

振分対象の領域分割情報が記述された index.dfi を指定します。

index.dfi に記述された構成にて CGNS ファイルが配置されている必要があります。

(2) -n, --np NP : 必須

ステージングを行う実行並列数を指定します。

(3) -o, --output [=OUTPUT_PATH]

振り分け結果のコピー先のディレクトリ名を指定します。

OUTPUT_PATH にディレクトリ名を指定します。

省略した場合はカレントディレクトリが出力先となります。

(例 1) --output=hoge

カレントディレクトリに hoge/ディレクトリが生成され、そのディレクトリ配下に各ランク用の 000000/,000001/,... ディレクトリが生成されます。

(例 2) 省略時

カレントディレクトリに各ランク用の 000000/,000001/,... ディレクトリが生成されます。

(4) -u, --with-udmlib [=UDMLIBTP_FILE]

分割パラメータ情報である"udmlib.tp"をコピーします。

UDMLIBTP_FILE に"udmlib.tp"のファイルパスを指定してください。

(5) `-s, --step [=STEP_NO]`

`udm-frm` は、すべての時系列情報、ファイルを振り分けます。

しかし、リスタート等の1つの時系列のみ必要な場合、**STEP_NO** を指定してください。**STEP_NO** に指定された時系列情報、ファイルのみ振り分けます

(6) `-v --view`

振り分け状況（進捗状況）を表示します。

(7) `--version`

`udm-frm` のバージョンを表示します。

(8) `-h --help`

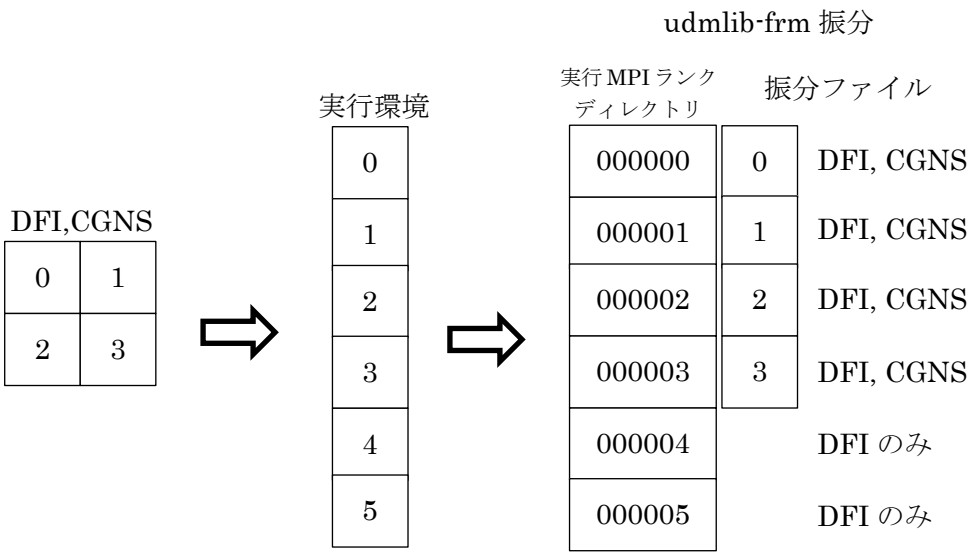
ヘルプ表示を行います。

7.3 使用例

udm-frm の使用例を以下に説明します。

7.3.1 4 分割 CGNS ファイルを 6 分割で実行

4 分割された CGNS ファイルを 6 プロセスで実行する場合を例に説明します。



4->6 分割実行ステージング

振分対象の DFI、CGNS ファイルは 4 分割されたファイルで以下の構成とします。

```
input_32-m4
|-- cgns_hexa_0000000000_id000000.cgns
|-- cgns_hexa_0000000000_id000001.cgns
|-- cgns_hexa_0000000000_id000002.cgns
|-- cgns_hexa_0000000000_id000003.cgns
|-- cgns_hexa_id000000.cgns
|-- cgns_hexa_id000001.cgns
|-- cgns_hexa_id000002.cgns
|-- cgns_hexa_id000003.cgns
|-- index.dfi
|-- proc.dfi
`-- udmlib.tp
```

この DFI、CGNS ファイルを 6MPI プロセスにて実行するする為に以下のコマンドにて **udm-frm** にてステージングを行います。

実行コマンド

```
$ /home/user/udmlib_front/bin/udm-frm ¥
--input=./input_32-m4/index.dfi ¥
--np=6 ¥
--output=./output_32-m6
```

a) /home/user/udmlib_front/bin/udm-frm

UDM ライブラリのインストール先の bin/udm-frm を実行します。

b) --input=./input_32-m4/index.dfi

振分対象の 4 分割 index.dfi を指定します。

c) --np=6

6 プロセスに振り分けを行います。

d) --output=./output_32-m6

カレントディレクトリ配下の"output_32-m6"に出力します。

実行後の振分結果は以下となります。

```
output_32-m6
|-- 000000
|  |-- cgns_hexa_0000000000_id000000.cgns
|  |-- cgns_hexa_id000000.cgns
|  |-- index.dfi
|  `-- proc.dfi
|-- 000001
|  |-- cgns_hexa_0000000000_id000001.cgns
|  |-- cgns_hexa_id000001.cgns
|  |-- index.dfi
|  `-- proc.dfi
|-- 000002
|  |-- cgns_hexa_0000000000_id000002.cgns
|  |-- cgns_hexa_id000002.cgns
|  |-- index.dfi
|  `-- proc.dfi
```



```

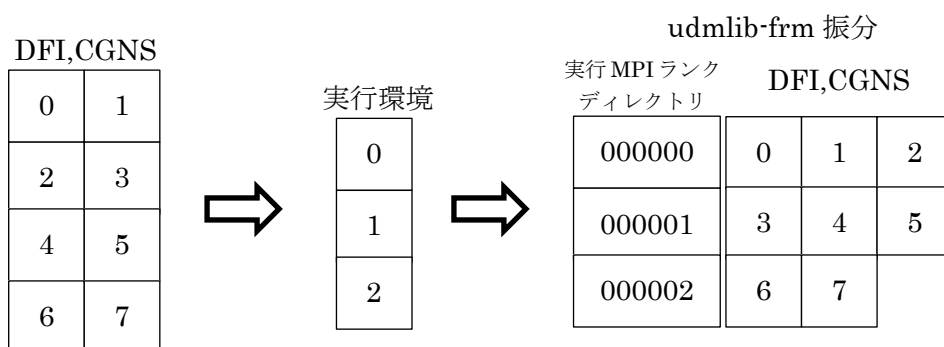
|-- 000003
|   |-- cgns_hexa_0000000000_id000003.cgns
|   |-- cgns_hexa_id000003.cgns
|   |-- index.dfi
|   `-- proc.dfi
|-- 000004
|   |-- index.dfi
|   `-- proc.dfi
`-- 000005
    |-- index.dfi
    `-- proc.dfi

```

- a) 振分プロセス番号の 6 桁のディレクトリが作成されます。
- b) 各ディレクトリに並列実行時に読み込む DFI ファイル、CGNS ファイルがコピーされます。
- c) 4,5 ランク番号は読み込むべき CGNS ファイルが存在しませんので、DFI ファイルのみとなります。

7.3.2 8 分割 CGNS ファイルを 3 分割で実行

8 分割、100 ステップ実行された CGNS ファイルを 3 プロセスで 50 ステップから実行する場合を例に説明します。



8->3 分割実行ステージング

振分対象の DFI、CGNS ファイルは 8 分割されたファイルで以下の構成とします。

100 ステップ分の CGNS ファイルが存在します。

```
input_step100-m8
|-- cgns_hexa_0000000000_id000000.cgns
|-- cgns_hexa_0000000000_id000001.cgns
|-- cgns_hexa_0000000000_id000002.cgns
|-- cgns_hexa_0000000000_id000003.cgns
|-- cgns_hexa_0000000000_id000004.cgns
|-- cgns_hexa_0000000000_id000005.cgns
|-- cgns_hexa_0000000000_id000006.cgns
|-- cgns_hexa_0000000000_id000007.cgns
|-- cgns_hexa_0000000010_id000000.cgns
|-- cgns_hexa_0000000010_id000001.cgns
|-- cgns_hexa_0000000010_id000002.cgns
|-- cgns_hexa_0000000010_id000003.cgns
|-- cgns_hexa_0000000010_id000004.cgns
|-- cgns_hexa_0000000010_id000005.cgns
|-- cgns_hexa_0000000010_id000006.cgns
|-- cgns_hexa_0000000010_id000007.cgns
(省略)
|-- cgns_hexa_0000000100_id000000.cgns
|-- cgns_hexa_0000000100_id000001.cgns
|-- cgns_hexa_0000000100_id000002.cgns
|-- cgns_hexa_0000000100_id000003.cgns
|-- cgns_hexa_0000000100_id000004.cgns
|-- cgns_hexa_0000000100_id000005.cgns
|-- cgns_hexa_0000000100_id000006.cgns
|-- cgns_hexa_0000000100_id000007.cgns
|-- cgns_hexa_id000000.cgns
|-- cgns_hexa_id000001.cgns
|-- cgns_hexa_id000002.cgns
|-- cgns_hexa_id000003.cgns
|-- cgns_hexa_id000004.cgns
|-- cgns_hexa_id000005.cgns
|-- cgns_hexa_id000006.cgns
|-- cgns_hexa_id000007.cgns
|-- index.dfi
```

```
| -- proc.dfi
|-- udmlib.tp
```

この DFI、CGNS ファイルを 3 プロセスにて実行する為に以下のコマンドにて **udm-frm** にてステージングを行います。

実行コマンド

```
$ /home/user/udmlib_front/bin/udm-frm ¥
--input=./input_step100-m8/index.dfi ¥
--np=3 ¥
--output=./output_step50-m3 ¥
--step=50 ¥
--with-udmlib=./input_step100-m8/udmlib.tp
```

a) /home/user/udmlib_front/bin/udm-frm

UDM ライブラリのインストール先の bin/udm-frm を実行します。

b) --input=./input_step100-m8/index.dfi

振分対象の 8 分割 index.dfi を指定します。

c) --np=3

3 プロセスに振り分けを行います。

d) --output=./output_step50-m3

カレントディレクトリ配下の "output_step50-m3" に出力します。

e) --step=50

時系列ステップ番号 50 のみのデータを振り分けします。

f) --with-udmlib=./input_step100-m8/udmlib.tp

udmlib.tp を振り分けます。

実行後の振分結果は以下となります。

```
output_step50-m3
|-- 000000
|  |-- cgns_hexa_0000000050_id000000.cgns
|  |-- cgns_hexa_0000000050_id000001.cgns
|  |-- cgns_hexa_0000000050_id000002.cgns
|  |-- cgns_hexa_id000000.cgns
|  |-- cgns_hexa_id000001.cgns
```

```

| |-- cgns_hexa_id000002.cgns
| |-- index.dfi
| |-- proc.dfi
| |-- udmlib.tp
|-- 000001
| |-- cgns_hexa_0000000050_id000003.cgns
| |-- cgns_hexa_0000000050_id000004.cgns
| |-- cgns_hexa_0000000050_id000005.cgns
| |-- cgns_hexa_id000003.cgns
| |-- cgns_hexa_id000004.cgns
| |-- cgns_hexa_id000005.cgns
| |-- index.dfi
| |-- proc.dfi
| |-- udmlib.tp
|-- 000002
| |-- cgns_hexa_0000000050_id000006.cgns
| |-- cgns_hexa_0000000050_id000007.cgns
| |-- cgns_hexa_id000006.cgns
| |-- cgns_hexa_id000007.cgns
| |-- index.dfi
| |-- proc.dfi
|-- udmlib.tp

```

- a) 振分プロセス番号の 6 桁のディレクトリが作成されます。
- b) 各ディレクトリに並列実行時に読み込む DFI ファイル、CGNS ファイルを振り分けします。
- c) ステップ番号 50 のみの CGNS ファイルを振り分けします。
- d) udm.tp もコピーします。

付録 1 : CGNS ライブラリのオプションビルド

CGNS ライブラリの機能として、CGNS ファイルの Viewer 等のツール、HDF5 のサポート付きでビルドする方法について説明します。

- (1) Tcl/Tk ライブラリのインストール
- (2) SZIP ライブラリのインストール
- (3) HDF5 ライブラリのインストール
- (4) cgnstools, HDF5 サポートの CGNS ライブラリのインストール

CGNS ファイルの Viewer 等のツールには以下のライブラリが必要となります。

Tcl/Tk ライブラリ

ホームページ : <http://www.tcl.tk/>

ダウンロードファイル

tcl8.6.1-src.tar.gz

tk8.6.1-src.tar.gz

(注) **tcl8.6.1** を使用してください。tcl8.6.2 は CGNS ライブラリはサポートしていません。: 2014/11/01 現在

また、CGNS ライブラリはデフォルトでは ADF データベースのみのサポートですが、HDF5 データベースもオプションでサポートします。(UDM ライブラリでは、ADF のみのサポートです。)

Downloads URL : <http://www.hdfgroup.org/HDF5/release/obtain5.html>

ダウンロードファイル

hdf5-1.8.13.tar.gz : HDF5 ソースファイル

szip-2.1.tar.gz : SZIP ソースファイル

zlib も必要となります。必要に応じてダウンロード(<http://www.zlib.net/>)してください。

Tcl/Tk ライブラリをソースファイルからインストールする方法について説明します。実行環境に rpm/yum, apt-get によりインストールされている場合は必要ありません。

```
$ mkdir /usr/local/tcl8.6.1
$ mkdir /usr/local/tcl8.6.1/src
$ cd /usr/local/tcl8.6.1/src (注)
$ wget http://sourceforge.net/projects/tcl/files/Tcl/8.6.1/tcl8.6.1-src.tar.gz/download
$ wget http://sourceforge.net/projects/tcl/files/Tcl/8.6.1/tk8.6.1-src.tar.gz/download
$ tar xzvf tcl8.6.1-src.tar.gz
```

```
$ tar xzvf tk8.6.1-src.tar.gz
(tcl8.6.1 のビルド)
$ cd tcl8.6.1/unix/
$ ./configure --prefix=/usr/local/tcl8.6.1 --enable-64bit
$ make
$ make install
(tk8.6.1 のビルド)
$ cd ../../tk8.6.1/unix/
$ ./configure --prefix=/usr/local/tcl8.6.1 ¥
    --with-x ¥
    --with-tcl=../../tcl8.6.1/unix ¥
    --enable-64bit
$ make
$ make install
```

Tcl/Tk ライブラリのビルド

(注) CGNS ライブラリが Tcl/Tk の[インストールパス/unix]を検索する為に Tcl/Tk のソースファイルをインストール先に配置する。

HDF5 ライブラリのビルド方法について以下に説明します。ZLIB ライブラリはインストール済みとし説明は割愛します。

```
$ tar xzvf szip-2.1.tar.gz
$ ./configure --prefix=/usr/local/szip-2.1
$ make
$ sudo make install
```

SZIP ライブラリのビルド

```
$ ./configure --prefix=/usr/local/hdf5-1.8.13 ¥
    --with-zlib ¥
    --with-szlib=/usr/local/szip-2.1 ¥
    --enable-cxx
$ make
$ sudo make install
```

HDF5 ライブラリのビルド

Tcl/Tk, SZIP, HDF5 ライブラリのインストール後、CGNS ライブラリのビルドを行います。

```
$ tar xzvf cgnslib_3.2.1.tar.gz
$ cd cgnslib_3.2.1/src
$ ./configure --prefix=/usr/local/cgnslib_3.2.1_hdf5 ¥
    --enable-cgnstools ¥
    --with-tcl=/usr/local/tcl8.6.1/src/tcl8.6.1 ¥
    --with-tk=/usr/local/tcl8.6.1/src/tk8.6.1 ¥
    --enable-64bit ¥
    --with-hdf5=/usr/local/hdf5-1.8.13 ¥
    --with-zlib ¥
    --with-szip="/usr/local/szip-2.1/lib/libsz.a -ldl" ¥      (注)
    --with-x
$ make
$ sudo make install
```

CGNS ライブラリのビルド

(注) "-ldl"を付加する必要があるが、LDFLAGS、LIBS に設定しても反映されないなので、--with-szip に付加します。

CGNS ライブラリのインストールにより cgnstools が使用可能となります。

/usr/local/cgnslib_3.2.1_hdf5/bin/cgnsview

CGNS ファイルのデータベースツリーを表示します。

/usr/local/cgnslib_3.2.1_hdf5/bin/cgnsplot

CGNS ファイルのモデルを表示します。

その他 cgnstools については、CGNS マニュアルを参照してください。

付録 2 : Zoltan パラメータによる分割の違い

Zoltan の分割パラメータには PACKAGE、LB_APPROACH に以下の値を設定可能です。

Zoltan パラメータ	設定値
PACKAGE	"HYPERGRAPH"
	"GRAPH"
LB_APPROACH	"PARTITION"
	"REPARTITION"

Zoltan の分割パラメータの違いによる分割 CGNS を以下に示します。

分割元モデル : 8x8x8

分割数 : 4

分割プログラムは"exapmles/cxx/partition"にて行います。オプション等の説明については「利用例 : Zoltan による分割 (partition)」を参照してください。

実行コマンド (exapmles/partition/partition は以下となります。

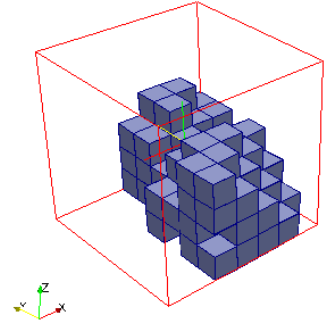
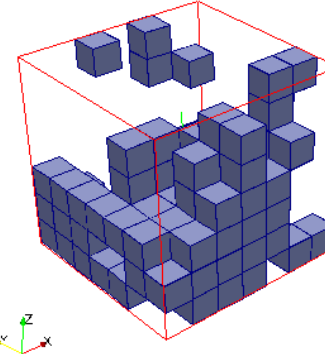
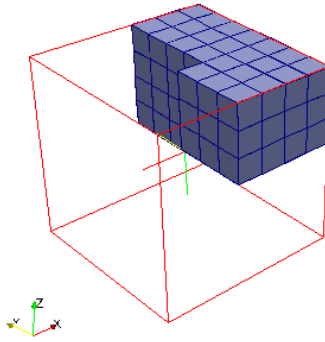
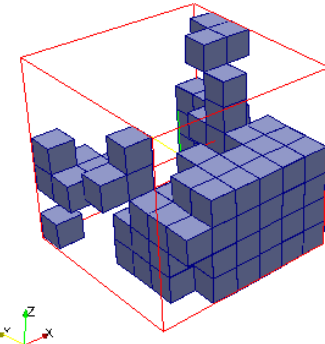
```
# PACKAGE=HYPERGRAPH LB_APPROACH=PARTITION
$ mpiexec -np 4 ./partition input_8/index.dfi ¥
--output=./output_8-hyper-partition ¥
--enable_partition

# PACKAGE=HYPERGRAPH LB_APPROACH=REPARTITION
$ mpiexec -np 4 ./partition input_8/index.dfi ¥
--output=./output_8-hyper-repartition ¥
--enable_repartition

# PACKAGE=GRAPH LB_APPROACH=PARTITION
$ mpiexec -np 4 ./partition input_8/index.dfi ¥
--output=./output_8-graph-partition ¥
--enable_graph --enable_partition

# PACKAGE=GRAPH LB_APPROACH=REPARTITION
$ mpiexec -np 4 ./partition input_8/index.dfi ¥
--output=./output_8-graph-repartition ¥
--enable_graph --enable_repartition
```

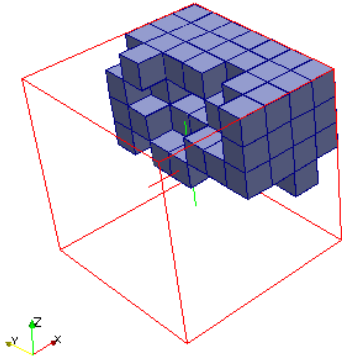
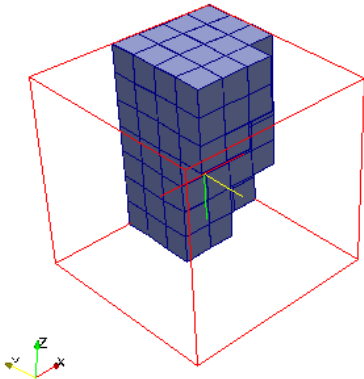

1 つの 8x8x8 モデルを 4 分割した時の cgns_hexa_8_id000000.cgns（ランク 0）のモデル表示は以下となります。

Zoltan パラメータ	cgns_hexa_8_id000000.cgns（ランク 0）分割結果
PACKAGE=HYPERGRAPH LB_APPROACH=PARTITION	
PACKAGE=HYPERGRAPH LB_APPROACH=REPARTITION	
PACKAGE=GRAPH LB_APPROACH=PARTITION	
PACKAGE=GRAPH LB_APPROACH=REPARTITION	

1 つの 8x8x8 モデルを 4 分割：ランク 0

また、4 分割された 8x8x8 モデルを 4 分割した時の cgns_hexa_8_id000000.cgns（ランク 0）のモデル表示は以下となります。

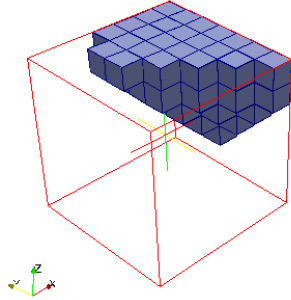
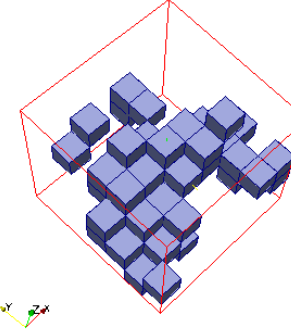
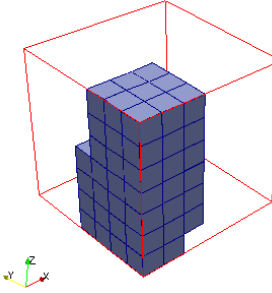
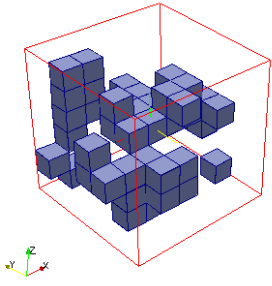
分割元モデル : 8x8x8 : output_8-graph-partition
前記の"GRAPH", "PARTITION"分割モデル
分割数 : 4

Zoltan パラメータ	cgns_hexa_8_id000000.cgns（ランク 0）分割結果
PACKAGE=HYPERGRAPH LB_APPROACH=PARTITION	
PACKAGE=HYPERGRAPH LB_APPROACH=REPARTITION	Zoltan No changes (分割なし)
PACKAGE=GRAPH LB_APPROACH=PARTITION	
PACKAGE=GRAPH LB_APPROACH=REPARTITION	Zoltan No changes (分割なし)

4 分割 8x8x8 モデルを 4 分割 : ランク 0

同様に、4 分割された 8x8x8 モデルを 5 分割した時の cgns_hexa_8_id000004.cgns（ランク 4）のモデル表示は以下となります。

分割元モデル : 8x8x8 : output_8-graph-partition
前記の"GRAPH", "PARTITION"分割モデル
分割数 : 5

Zoltan パラメータ	cgns_hexa_8_id000004.cgns（ランク 4）分割結果
PACKAGE=HYPERGRAPH LB_APPROACH=PARTITION	
PACKAGE=HYPERGRAPH LB_APPROACH=REPARTITION	
PACKAGE=GRAPH LB_APPROACH=PARTITION	
PACKAGE=GRAPH LB_APPROACH=REPARTITION	

4 分割 8x8x8 モデルを 5 分割 : ランク 4