Cartesian Input/Output Library

1.4.4

作成： Doxygen 1.8.4

Wed Jan 8 2014 15:12:35

Contents

Chapter 1

階層索引

1.1 クラス階層

この継承一覧はおおまかにはソートされていますが、完全にアルファベット順でソートされてはいません。

Chapter 2

構成索引

2.1 構成

クラス、構造体、共用体、インタフェースの説明です。

Chapter 3

ファイル索引

3.1 ファイル一覧

これはファイル一覧です。

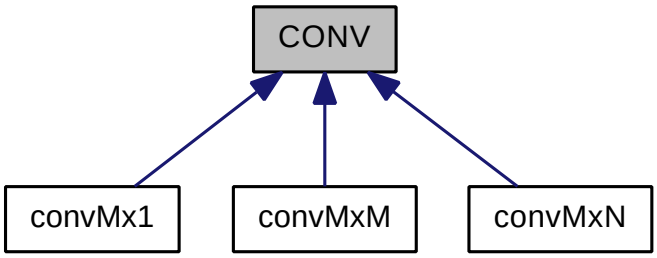
Chapter 4

クラス

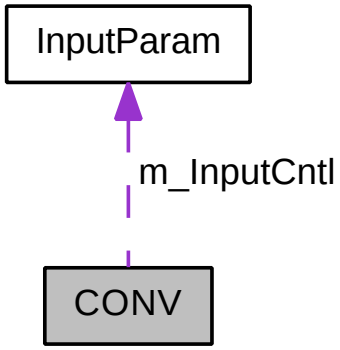
4.1 クラス CONV

`#include <conv.h>`

CONV に対する継承グラフ



CONV のコラボレーション図



構成

- struct dfi_MinMax
- struct step_rank_info

Public メソッド

- CONV ()

- ~CONV ()
- bool importCPM (cpm_ParaManager *paraMngr)
  CPM のポインタをコピーし、ランク情報を設定
- void setRankInfo ()
  ランク情報をセットする
- bool importInputParam (InputParam *InputCntl)
  InputParam のポインタをコピー
- void ReadDfiFiles ()
  dfi ファイルの読み込みと*DfiInfo* クラスデータの作成
- bool CheckConvData ()
  コンバートデータのエラーチェック
- void CheckDir (string dirstr)
  出力指定ディレクトリのチェック
- void OpenLogFile ()
  ログファイルのオープン
- void CloseLogFile ()
  ログファイルのクローズ
- void WriteTime (double *tt)
  所要時間の記述
- void MemoryRequirement (const double Memory, FILE *fp)
  メモリ使用量を表示する
- void MemoryRequirement (const double TotalMemory, const double sphMemory, const double plot3dMemory, const double thinMemory, FILE *fp)
  メモリ使用量を表示する
- virtual void VoxelInit ()
  領域分割と出力*DFI* のインスタンス
- virtual bool exec ()=0
  コーンバート処理
- double GetSliceTime (cio_DFI *dfi, int step)
  step 番号から *time* を取得
- bool convertXY (cio_Array *buf, cio_Array *&src, int headS[3], int tailS[3], int n)
  配列のコンバート
- template<class T1 , class T2 >
  bool copyArray (cio_TypeArray< T1 > *buf, cio_TypeArray< T2 > *&src, int sta[3], int end[3], int n)
  配列のコピー *(template* 関数*)*
- bool DtypeMinMax (cio_Array *src, double *min, double *max)
  データタイプ毎に *minmax* を求める
- template<class T >
  bool calcMinMax (cio_TypeArray< T > *src, double *min, double *max)
  minmax を計算
- void makeStepList (vector< step_rank_info > &StepRankList)
  step 基準のリスト生成
- void makeRankList (vector< step_rank_info > &StepRankList)
  rank 基準のリスト生成
- bool WriteIndexDfiFile (vector< dfi_MinMax * > minmaxList)
  index.dfi の出力
- bool WriteProcDfiFile (std::string proc_name, cio_Domain *out_domain, cio_MPI *out_mpi, cio_Process *out_process)
  proc.dfi の出力
- bool makeProcInfo (cio_DFI *dfi, cio_Domain *&out_domain, cio_MPI *&out_mpi, cio_Process *&out_process, int numProc)

Proc 情報の生成

- template<class T1 , class T2 >
  CONV_INLINE bool copyArray (cio_TypeArray< T1 > *buf, cio_TypeArray< T2 > *&src, int sta[3], int end[3], int n)
- template<class T >
  CONV_INLINE bool calcMinMax (cio_TypeArray< T > *src, double *min, double *max)

Static Public メソッド

- static CONV * ConvInit (InputParam *InputCntl)

  conv インスタンス
- static std::string GetFilenameExt (int file_format_type)

  出力ファイル形式から拡張子を求める

Public 変数

- cpm_ParaManager * m_paraMngr

  Cartesian Partition Manager.
- InputParam * m_InputCntl

  InputParam Class.
- int m_procGrp

  プロセスグループ番号
- int m_myRank

  自ノードのランク番号
- int m_numProc

  全ランク数
- std::string m_HostName

  ホスト名
- bool m_bgrid_interp_flag

  節点への補間フラグ
- int m_pflag
- int m_pflagv
- int m_lflag
- int m_lflagv
- vector< cio_DFI * > m_in_dfi

Protected 変数

- FILE * m_fplog

Private 変数

- unsigned m_staging

4.1.1 説明

conv.h の 47 行で定義されています。

4.1.2 コンストラクタとデストラクタ

4.1.2.1 CONV::CONV ( )

コンストラクタ

conv.C の 25 行で定義されています。

参照先 m_bgrid_interp_flag, m_in_dfi, m_lflag, m_lflagv, m_myRank, m_numProc, m_pflag, m_pflagv, m_procGrp, と m_staging.

```
26 {
27   m_procGrp = 0;
28   m_myRank  = -1;
29   m_numProc = 0;
30
31   m_pflag=0;
32   m_pflagv=0;
33   m_lflag=0;
34   m_lflagv=0;
35
36   m_bgrid_interp_flag = false;
37
38   m_in_dfi.clear();
39
40   m_staging=0;
41
42 }
```

4.1.2.2 CONV::~CONV ( )

デストラクタ

conv.C の 47 行で定義されています。

参照先 m_in_dfi.

```
48 {
49   for(int i=0; i<m_in_dfi.size(); i++ ) if( !m_in_dfi[i] ) delete m_in_dfi[i];
50
51 }
```

4.1.3 関数

4.1.3.1 template<class T > CONV_INLINE bool CONV::calcMinMax (cio_TypeArray< T > * *src*, double * *min*, double * *max*)

conv_inline.h の 102 行で定義されています。

```
105 {
106
107   if( src == NULL ) return false;
108
109   double CompVal;
110
111   //size の取得
112   const int *sz = src->getArraySizeInt();
113   //配列形状の取得
114   CIO::E_CIO_ARRAYSHAPE shape = src->getArrayShape();
115   //成分数の取得
116   int nComp = src->getNcomp();
117
118
119   if( nComp > 1 ) {
120     //nijk の処理
121     if( shape == CIO::E_CIO_NIJK ) {
122       for(int k=0; k<sz[2]; k++) {
123       for(int j=0; j<sz[1]; j++) {
124       for(int i=0; i<sz[0]; i++) {
125         CompVal=(double)0.0;
126         for(int n=0; n<nComp; n++) {
127           if( min[n] > (double)src->val(n,i,j,k) ) min[n] = (double)src->val(n,i,j,k);
```

```
128           if( max[n] < (double)src->val(n,i,j,k) ) max[n] = (double)src->val(n,i,j,k);
129           CompVal = CompVal + (double)src->val(n,i,j,k)*(double)src->val(n,i,j,k);
130         }
131         CompVal = sqrt(CompVal);
132         if( min[nComp] > CompVal ) min[nComp]=CompVal;
133         if( max[nComp] < CompVal ) max[nComp]=CompVal;
134       }}}
135
136     }
137     else if( shape == CIO::E_CIO_IJKN )
138     //ijkn の処理
139     {
140       for(int k=0; k<sz[2]; k++) {
141       for(int j=0; j<sz[1]; j++) {
142       for(int i=0; i<sz[0]; i++) {
143         CompVal=(double)0.0;
144         for(int n=0; n<nComp; n++) {
145           if( min[n] > (double)src->val(i,j,k,n) ) min[n] = (double)src->val(i,j,k,n);
146           if( max[n] < (double)src->val(i,j,k,n) ) max[n] = (double)src->val(i,j,k,n);
147           CompVal = CompVal + (double)src->val(i,j,k,n)*(double)src->val(i,j,k,n);
148         }
149         CompVal = sqrt(CompVal);
150         if( min[nComp] > CompVal ) min[nComp]=CompVal;
151         if( max[nComp] < CompVal ) max[nComp]=CompVal;
152       }}}
153     } else return false;
154
155   } else {
156     //nijk の処理
157     if( shape == CIO::E_CIO_NIJK ) {
158       for(int k=0; k<sz[2]; k++) {
159       for(int j=0; j<sz[1]; j++) {
160       for(int i=0; i<sz[0]; i++) {
161         if( min[0] > (double)src->val(0,i,j,k) ) min[0] = (double)src->val(0,i,j,k);
162         if( max[0] < (double)src->val(0,i,j,k) ) max[0] = (double)src->val(0,i,j,k);
163       }}}
164     }
165     else if( shape == CIO::E_CIO_IJKN )
166     //ijkn の処理
167     {
168       for(int k=0; k<sz[2]; k++) {
169       for(int j=0; j<sz[1]; j++) {
170       for(int i=0; i<sz[0]; i++) {
171         if( min[0] > (double)src->val(i,j,k,0) ) min[0] = (double)src->val(i,j,k,0);
172         if( max[0] < (double)src->val(i,j,k,0) ) max[0] = (double)src->val(i,j,k,0);
173       }}}
174     } else return false;
175
176   }
177   return true;
178
179 }
```

4.1.3.2 template<class T > bool CONV::calcMinMax (cio_TypeArray< T > * *src*, double * *min*, double * *max*)

minmax を計算

引数

in	*src*	minmax を求める配列データのポインタ
out	*min*	求められた最小値
out	*max*	求められた最大値

参照元 DtypeMinMax().

4.1.3.3 bool CONV::CheckConvData ( )

コンバートデータのエラーチェック

戻り値

エラーコード

conv.C の 222 行で定義されています。

参照先 E_OUTPUT_Mx1, E_OUTPUT_RANK, InputParam::Get_ConvType(), InputParam::Get_MultiFileCasting(), InputParam::Get_OutdfiNameList(), InputParam::Get_OutprocNameList(), InputParam::Get_OutputArrayShape(), InputParam::Get_OutputDataType(), InputParam::Get_OutputFormat(), InputParam::Get_OutputFormat_string(), InputParam::Get_OutputFormatType(), InputParam::Get_OutputGuideCell(), m_in_dfi, m_InputCntl, InputParam::-Set_OutprocNameList(), と InputParam::Set_OutputArrayShape().

参照元 main().

```
223 {
224
225   bool ierr=true;
226
227   //出力形式のチェック
228   if( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
229     if( m_InputCntl->Get_OutputFormat() != CIO::E_CIO_FMT_PLOT3D &&
230         m_InputCntl->Get_OutputFormat() != CIO::E_CIO_FMT_VTK ) {
231       printf("\tCan't Converter OutputFormatType.\n");
232       ierr=false;
233     }
234   }
235
236   //出力配列形状のチェック
237   //BOV 以外での出力配列形状指示は無効とし、自動的に対応する配列形状で出力
238   //なので、指定があった場合はメッセージを出力する
239   if( m_InputCntl->Get_OutputArrayShape() != CIO::E_CIO_ARRAYSHAPE_UNKNOWN ) {
240     if( m_InputCntl->Get_OutputFormat() != CIO::E_CIO_FMT_BOV ) {
241       printf("\tCan't OutputArrayShape.\n");
242       //ierr=false;
243     }
244   }
245
246   //出力配列形状のセット
247   if( m_InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_SPH ) {
248     m_InputCntl->Set_OutputArrayShape(CIO::E_CIO_NIJK);
249   }else if( m_InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_AVS ) {
250     m_InputCntl->Set_OutputArrayShape(CIO::E_CIO_NIJK);
251   }else if( m_InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_VTK ) {
252     m_InputCntl->Set_OutputArrayShape(CIO::E_CIO_NIJK);
253   }else if( m_InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_PLOT3D ) {
254     m_InputCntl->Set_OutputArrayShape(CIO::E_CIO_IJKN);
255   }else if(  m_InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_BOV &&
256              m_InputCntl->Get_OutputArrayShape() == CIO::E_CIO_ARRAYSHAPE_UNKNOWN ) {
257     m_InputCntl->Set_OutputArrayShape(CIO::E_CIO_NIJK);
258   }
259
260   //未対応のデータ型への変換チェック (sph,plot3d)
261   if( m_InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_SPH ||
262       m_InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_PLOT3D ) {
263     if( m_InputCntl->Get_OutputDataType() != CIO::E_CIO_FLOAT32 &&
264         m_InputCntl->Get_OutputDataType() != CIO::E_CIO_FLOAT64 ) {
265       printf("\tCan't Converter OutputDataType.\n");
266       ierr=false;
267     }
268   }
269
270   //未対応のデータ型への変換チェック (bov,avs)
271   if( m_InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_BOV ||
272       m_InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_AVS ) {
273     if( m_InputCntl->Get_OutputDataType() != CIO::E_CIO_INT8    &&
274         m_InputCntl->Get_OutputDataType() != CIO::E_CIO_INT16   &&
275         m_InputCntl->Get_OutputDataType() != CIO::E_CIO_INT32   &&
276         m_InputCntl->Get_OutputDataType() != CIO::E_CIO_FLOAT32 &&
277         m_InputCntl->Get_OutputDataType() != CIO::E_CIO_FLOAT64 ) {
278       printf("\tCan't Converter OutputDataType.\n");
279       ierr=false;
280     }
281   }
282
283   for( int i=0; i<m_in_dfi.size(); i++) {
284
285     //コンバート成分数のチェック
286     if( m_InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_SPH ) {
287       if( m_in_dfi[i]->GetNumComponent() > 3 ) {
288         printf("\tCan't Converter OutputFormat.\n");
289         ierr=false;
290       }
291     }
292   }
293
294   //DFI 出力のチェック、出力する DFI ファイル名のチェック
295   vector<std::string> out_dfi_name = m_InputCntl->Get_OutdfiNameList();
296   if( out_dfi_name.size() > 0 ) {
297     if( m_InputCntl->Get_OutputFormat() != CIO::E_CIO_FMT_SPH &&
298         m_InputCntl->Get_OutputFormat() != CIO::E_CIO_FMT_BOV ) {
299       printf("\tCan't outpue dfi OutputFormat. %s\n",
```

```
300             m_InputCntl->Get_OutputFormat_string().c_str());
301       ierr=false;
302     }
303   }
304   for( int i=0; i<out_dfi_name.size(); i++) {
305     std::string inPath = CIO::cioPath_DirName(out_dfi_name[i]);
306     if( inPath != "" && inPath !="./" ) {
307       printf("\tIllegal OutputDFI : %s\n",out_dfi_name[i].c_str());
308       ierr=false;
309     }
310   }
311
312   //出力する proc ファイル名のチェック
313   vector<std::string> out_proc_name = m_InputCntl->Get_OutprocNameList();
314   for( int i=0; i<out_proc_name.size(); i++) {
315     std::string inPath = CIO::cioPath_DirName(out_proc_name[i]);
316     if( inPath != "" && inPath !="./" ) {
317       printf("\tIllegal OutputProc : %s\n",out_proc_name[i].c_str());
318       ierr=false;
319     }
320   }
321
322   //出力する proc ファイル名が省略された場合、出力する dfi ファイル名から生成
323   if( out_proc_name.size() < 1 && out_dfi_name.size() > 0 ) {
324     for(int i=0; i<out_dfi_name.size(); i++) {
325       std::string proc = CIO::ExtractPathWithoutExt(out_dfi_name[i]);
326       std::string fname = proc+"_proc.dfi";
327       out_proc_name.push_back(fname);
328     }
329     if( !(m_InputCntl->Set_OutprocNameList(out_proc_name)) ) ierr=false;
330   }
331
332   //出力ガイドセル数のチェック
333   if( m_InputCntl->Get_OutputGuideCell() > 0 ) {
334     if( m_InputCntl->Get_OutputFormat() != CIO::E_CIO_FMT_BOV &&
335         m_InputCntl->Get_OutputFormat() != CIO::E_CIO_FMT_SPH ) {
336       printf("\tCan't output guide cell : %s\n",
337               m_InputCntl->Get_OutputFormat_string().c_str());
338       ierr=false;
339     }
340   }
341
342   //ファイル割振り方法のチェック
343   if( m_InputCntl->Get_MultiFileCasting() == E_OUTPUT_RANK ) {
344     if( m_InputCntl->Get_ConvType() == E_OUTPUT_Mx1 ) {
345       printf("\tCan't muiti file casting type rank\n");
346     }
347   }
348
349   return ierr;
350
351 }
```

4.1.3.4 void CONV::CheckDir (string *dirstr*)

出力指定ディレクトリのチェック

引数

in	*dirstr*	出力ディレクトリ

conv.C の 355 行で定義されています。

参照先 Exit, と Hostonly_.

参照元 main().

```
356 {
357   Hostonly_
358   {
359
360 #ifndef _WIN32
361
362     if( dirstr.size() == 0 ) {
363       //printf("\toutput current directory\n");
364       return;
365     }
366
367     DIR* dir;
368     if( !(dir = opendir(dirstr.c_str())) ) {
```

```
369       if( errno == ENOENT ) {
370         mode_t mode = S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH;
371         if ( cio_DFI::MakeDirectorySub(dirstr) != 0 )
372         {
373           printf("\tCan't generate directory(%s).\n", dirstr.c_str());
374           Exit(0);
375         }
376       }
377       else {
378         printf("Directory open error.(%s)", dirstr.c_str());
379         Exit(0);
380       }
381     }
382     else {
383       if( closedir(dir) == -1 ) {
384         printf("Directory close error.(%s)", dirstr.c_str());
385         Exit(0);
386       }
387     }
388
389 #else // for windows
390
391     if( dirstr.size() == 0 ) {
392       printf("\toutput current directory\n");
393       return;
394     }
395
396     // check to exist directory
397     if (IsDirExsist(dirstr)) {
398       // exist directory
399       return;
400     }
401
402     // make directory
403     if(!CreateDirectory(dirstr.c_str(), NULL)){
404       printf("\tCan't generate directory(%s).\n", dirstr.c_str());
405       Exit(0);
406     }
407
408 #endif  // _WIN32
409
410   }
411
412   return;
413 }
```

4.1.3.5 void CONV::CloseLogFile ( )

ログファイルのクローズ

conv.C の 448 行で定義されています。

参照先 m_fplog.

参照元 main().

```
449 {
450   fclose(m_fplog);
451 }
```

4.1.3.6 bool CONV::convertXY (cio_Array * *buf*, cio_Array *& *src*, int *headS[3]*, int *tailS[3]*, int *n*)

配列のコンバート

引数

in	*buf*	読込み用バッファ
out	*src*	読み込んだデータを格納した配列のポインタ

in	*headS*	出力領域の head インデックス
in	*tailS*	出力領域の tail インデックス
in	*n*	成分位置

conv.C の 636 行で定義されています。

参照先 copyArray().

参照元 convMx1::convMx1_out_ijkn(), convMx1::convMx1_out_nijk(), convMxN::exec(), と convMxM::mxmsolv().

```
642 {
643
644   //copy
645   int gcB          = buf->getGcInt();
646   const int *headB = buf->getHeadIndex();
647   const int *tailB = buf->getTailIndex();
648   int       gcS    = src->getGcInt();
649   int sta[3],end[3];
650   for( int i=0;i<3;i++ )
651   {
652     sta[i] = (headB[i]-gcB>=headS[i]-gcS) ? headB[i]-gcB : headS[i]-gcS;
653     end[i] = (tailB[i]+gcB<=tailS[i]+gcS) ? tailB[i]+gcB : tailS[i]+gcS;
654   }
655
656   //同じデータ型のコピー
657   if( buf->getDataType() == src->getDataType() ) {
658     // float to float
659     if( buf->getDataType() == CIO::E_CIO_FLOAT32 ) {
660       cio_TypeArray<float> *B = dynamic_cast<cio_TypeArray<float>*>(buf);
661       cio_TypeArray<float> *S = dynamic_cast<cio_TypeArray<float>*>(src);
662       copyArray(B, S, sta, end, n);
663     // double to double
664     } else {
665       cio_TypeArray<double> *B = dynamic_cast<cio_TypeArray<double>*>(buf);
666       cio_TypeArray<double> *S = dynamic_cast<cio_TypeArray<double>*>(src);
667       copyArray(B, S, sta, end, n);
668     }
669     //違う型のコピー
670   } else {
671     // float to double
672     if( buf->getDataType() == CIO::E_CIO_FLOAT32 &&
673        src->getDataType() == CIO::E_CIO_FLOAT64 ) {
674       cio_TypeArray<float>  *B = dynamic_cast<cio_TypeArray<float>*>(buf);
675       cio_TypeArray<double> *S = dynamic_cast<cio_TypeArray<double>*>(src);
676       copyArray(B, S, sta, end, n);
677       //doubel to float
678     } else {
679       cio_TypeArray<double> *B = dynamic_cast<cio_TypeArray<double>*>(buf);
680       cio_TypeArray<float>  *S = dynamic_cast<cio_TypeArray<float>*>(src);
681       copyArray(B, S, sta, end, n);
682     }
683   }
684
685   return true;
686 }
```

4.1.3.7 CONV * CONV::ConvInit (InputParam * *InputCntl*) [static]

conv インスタンス

引数

in	*InputCntl*	InputParam クラスポインタ

conv.C の 55 行で定義されています。

参照先 E_OUTPUT_Mx1, E_OUTPUT_MxM, E_OUTPUT_MxN, InputParam::Get_ConvType(), InputParam::Get_OutputFormat(), と m_bgrid_interp_flag.

参照元 main().

```
56 {
57   CONV* conv = NULL;
58
59   if(InputCntl->Get_ConvType() == E_OUTPUT_Mx1 ) {
60     conv = new convMx1();
61   } else if( InputCntl->Get_ConvType() == E_OUTPUT_MxM ) {
```

```
62     conv = new convMxM();
63   } else if( InputCntl->Get_ConvType() == E_OUTPUT_MxN ) {
64     conv = new convMxN();
65   }
66
67   //格子点補間フラグのセット
68   if( InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_PLOT3D ||
69       InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_AVS ||
70       InputCntl->Get_OutputFormat() == CIO::E_CIO_FMT_VTK ) {
71     conv->m_bgrid_interp_flag = true;
72   } else {
73     conv->m_bgrid_interp_flag = false;
74   }
75
76   return conv;
77 }
```

4.1.3.8 template<class T1 , class T2 > CONV_INLINE bool CONV::copyArray (cio_TypeArray< T1 > * *buf*, cio_TypeArray< T2 > *& *src*, int *sta[3]*, int *end[3]*, int *n*)

conv_inline.h の 31 行で定義されています。

参照先 InputParam::Get_ThinOut(), と m_InputCntl.

```
36 {
37
38   //配列形状
39   CIO::E_CIO_ARRAYSHAPE buf_shape = buf->getArrayShape();
40
41   CIO::E_CIO_ARRAYSHAPE src_shape = src->getArrayShape();
42
43   //間引き数の取得
44   int thin_count = m_InputCntl->Get_ThinOut();
45
46   //IJKN&IJKN
47   if( buf_shape == CIO::E_CIO_IJKN && src_shape == CIO::E_CIO_IJKN )
48   {
49     for( int k=sta[2];k<=end[2];k++ ){
50     if( k%thin_count != 0 ) continue;
51     for( int j=sta[1];j<=end[1];j++ ){
52     if( j%thin_count != 0 ) continue;
53     for( int i=sta[0];i<=end[0];i++ ){
54       if( i%thin_count != 0 ) continue;
55       src->hval(i/thin_count,j/thin_count,k/thin_count,n) = (T2)buf->hval(i,j,k,n);
56     }}}
57   }
58   else if( buf_shape == CIO::E_CIO_NIJK && src_shape == CIO::E_CIO_NIJK )
59   //NIJK&NIJK
60   {
61     for( int k=sta[2];k<=end[2];k++ ){
62     if( k%thin_count != 0 ) continue;
63     for( int j=sta[1];j<=end[1];j++ ){
64     if( j%thin_count != 0 ) continue;
65     for( int i=sta[0];i<=end[0];i++ ){
66       if( i%thin_count != 0 ) continue;
67       src->hval(n,i/thin_count,j/thin_count,k/thin_count) = (T2)buf->hval(n,i,j,k);
68     }}}
69   }
70   else if( buf_shape == CIO::E_CIO_IJKN && src_shape == CIO::E_CIO_NIJK )
71   //IJNK&NIJK
72   {
73     for( int k=sta[2];k<=end[2];k++ ){
74     if( k%thin_count != 0 ) continue;
75     for( int j=sta[1];j<=end[1];j++ ){
76     if( j%thin_count != 0 ) continue;
77     for( int i=sta[0];i<=end[0];i++ ){
78       if( i%thin_count != 0 ) continue;
79       src->hval(n,i/thin_count,j/thin_count,k/thin_count) = (T2)buf->hval(i,j,k,n);
80     }}}
81   }
82   else if( buf_shape == CIO::E_CIO_NIJK && src_shape == CIO::E_CIO_IJKN )
83   //NIJK&IJKN
84   {
85     for( int k=sta[2];k<=end[2];k++ ){
86     if( k%thin_count != 0 ) continue;
87     for( int j=sta[1];j<=end[1];j++ ){
88     if( j%thin_count != 0 ) continue;
89     for( int i=sta[0];i<=end[0];i++ ){
90       if( i%thin_count != 0 ) continue;
91       src->hval(i/thin_count,j/thin_count,k/thin_count,n) = (T2)buf->hval(n,i,j,k);
92     }}}
```

```
93   }
94
95   return true;
96 }
```

4.1.3.9 template<class T1 , class T2 > bool CONV::copyArray (cio_TypeArray< T1 > * *buf*, cio_TypeArray< T2 > *& *src*, int *sta[3]*, int *end[3]*, int *n*)

配列のコピー (template 関数)

引数

in	*buf*	コピー元の配列
out	*src*	コピー先の配列
in	*sta*	コピーのスタート位置
in	*end*	コピーのエンド位置
in	*n*	成分位置

参照元 convertXY().

4.1.3.10 bool CONV::DtypeMinMax (cio_Array * *src*, double * *min*, double * *max*)

データタイプ毎に minmax を求める

引数

in	*src*	minmax を求める配列データのポインタ
out	*min*	求められた最小値
out	*max*	求められた最大値

conv.C の 818 行で定義されています。

参照先 calcMinMax().

参照元 convMx1::convMx1_out_ijkn(), convMx1::convMx1_out_nijk(), convMxN::exec(), と convMxM::mxmsolv().

```
821 {
822   CIO::E_CIO_DTYPE d_type = src->getDataType();
823   if( d_type == CIO::E_CIO_UINT8 ) {
824     cio_TypeArray<unsigned char> *data = dynamic_cast<cio_TypeArray<unsigned char>*>(src);
825     if( !calcMinMax(data,min,max) ) return false;
826   } else if( d_type == CIO::E_CIO_INT8 ) {
827     cio_TypeArray<char> *data = dynamic_cast<cio_TypeArray<char>*>(src);
828     if( !calcMinMax(data,min,max) ) return false;
829   } else if( d_type == CIO::E_CIO_UINT16 ) {
830     cio_TypeArray<unsigned short> *data = dynamic_cast<cio_TypeArray<unsigned short>*>(src);
831     if( !calcMinMax(data,min,max) ) return false;
832   } else if( d_type == CIO::E_CIO_INT16 ) {
833     cio_TypeArray<short> *data = dynamic_cast<cio_TypeArray<short>*>(src);
834     if( !calcMinMax(data,min,max) ) return false;
835   } else if( d_type == CIO::E_CIO_UINT32 ) {
836     cio_TypeArray<unsigned int> *data = dynamic_cast<cio_TypeArray<unsigned int>*>(src);
837     if( !calcMinMax(data,min,max) ) return false;
838   } else if( d_type == CIO::E_CIO_INT32 ) {
839     cio_TypeArray<int> *data = dynamic_cast<cio_TypeArray<int>*>(src);
840     if( !calcMinMax(data,min,max) ) return false;
841   } else if( d_type == CIO::E_CIO_UINT64 ) {
842     cio_TypeArray<unsigned long> *data = dynamic_cast<cio_TypeArray<unsigned long>*>(src);
843     if( !calcMinMax(data,min,max) ) return false;
844   } else if( d_type == CIO::E_CIO_INT64 ) {
845     cio_TypeArray<long> *data = dynamic_cast<cio_TypeArray<long>*>(src);
846     if( !calcMinMax(data,min,max) ) return false;
847   } else if( d_type == CIO::E_CIO_FLOAT32 ) {
848     cio_TypeArray<float> *data = dynamic_cast<cio_TypeArray<float>*>(src);
849     if( !calcMinMax(data,min,max) ) return false;
850   } else if( d_type == CIO::E_CIO_FLOAT64 ) {
851     cio_TypeArray<double> *data = dynamic_cast<cio_TypeArray<double>*>(src);
852     if( !calcMinMax(data,min,max) ) return false;
853   }
854
855   return true;
856
857 }
```

4.1.3.11 virtual bool CONV::exec ( ) [pure virtual]

コーンバート処理

convMxN, convMx1, と convMxMで実装されています。

参照元 main().

4.1.3.12 std::string CONV::GetFilenameExt (int *file_format_type*) [static]

出力ファイル形式から拡張子を求める

引数

in	*file_format_type*	

戻り値

拡張子

conv.C の 690 行で定義されています。

```
691 {
692
693   if     ( file_format_type == CIO::E_CIO_FMT_SPH ) return D_CIO_EXT_SPH;
694   else if( file_format_type == CIO::E_CIO_FMT_BOV ) return D_CIO_EXT_BOV;
695   else if( file_format_type == CIO::E_CIO_FMT_AVS ) return D_CIO_EXT_SPH;
696   else if( file_format_type == CIO::E_CIO_FMT_PLOT3D ) return D_CIO_EXT_FUNC;
697   else if( file_format_type == CIO::E_CIO_FMT_VTK ) return D_CIO_EXT_VTK;
698
699   return "";
700 }
```

4.1.3.13 double CONV::GetSliceTime (cio_DFI * *dfi*, int *step*)

step 番号から time を取得

引数

in	*dfi*	dfi のポインター
in	*step*	step 番号

戻り値

time

conv.C の 622 行で定義されています。

```
623 {
624
625   const cio_TimeSlice* Tslice = dfi->GetcioTimeSlice();
626   for(int i=0; i<Tslice->SliceList.size(); i++) {
627     if( Tslice->SliceList[i].step == step ) return Tslice->SliceList[i].time;
628   }
629
630   return 0.0;
631
632 }
```

4.1.3.14 bool CONV::importCPM (cpm_ParaManager * *paraMngr*) [inline]

CPM のポインタをコピーし、ランク情報を設定

引数

in	*paraMngr*	cpm_ParaManager クラス

戻り値

エラーコード

conv.h の 131 行で定義されています。

参照元 main().

```
132   {
133     if ( !paraMngr ) return false;
134     m_paraMngr = paraMngr;
135     setRankInfo();
136     return true;
137   }
```

4.1.3.15 bool CONV::importInputParam (InputParam * *InputCntl*) [inline]

InputParam のポインタをコピー

引数

in	*InputCntl*	InputParam クラスポインタ

戻り値

エラーコード

conv.h の 155 行で定義されています。

参照元 main().

```
156   {
157     if( !InputCntl ) return false;
158     m_InputCntl = InputCntl;
159     return true;
160   }
```

4.1.3.16 bool CONV::makeProcInfo (cio_DFI * *dfi*, cio_Domain *& *out_domain*, cio_MPI *& *out_mpi*, cio_Process *& *out_process*, int *numProc*)

Proc 情報の生成

conv.C の 971 行で定義されています。

参照先 InputParam::Get_ThinOut(), と m_InputCntl.

参照元 convMxM::exec(), と convMx1::exec().

```
976 {
977
978   //間引き数の取得
979   int thin_count = m_InputCntl->Get_ThinOut();
980
981   //MPI 情報の生成
982   out_mpi = new cio_MPI(numProc,0);
983
984   //Domain 情報の生成
985   cio_Domain* dfi_domain = (cio_Domain *)dfi->GetcioDomain();
986   double Gorigin[3];
987   double Gregion[3];
988   int Gvoxel[3];
989   int Gdiv[3];
```

```
990   for(int i=0; i<3; i++) {
991     Gorigin[i] = dfi_domain->GlobalOrigin[i];
992     Gregion[i] = dfi_domain->GlobalRegion[i];
993     Gvoxel[i]  = dfi_domain->GlobalVoxel[i];
994     Gdiv[i]    = dfi_domain->GlobalDivision[i];
995   }
996   //間引きありのときボクセルサイズを更新
997   if( thin_count > 1 ) {
998     for(int i=0; i<3; i++) {
999       if( Gvoxel[i]%thin_count != 0 ) Gvoxel[i]=Gvoxel[i]/thin_count+1;
1000       else                            Gvoxel[i]=Gvoxel[i]/thin_count;
1001     }
1002   }
1003   //numProc が 1 のとき（Mx1 のとき）GlobalDivision を 1 にする
1004   if( numProc == 1 ) for(int i=0; i<3; i++) Gdiv[i]=1;
1005
1006   //out_domain の生成
1007   out_domain = new cio_Domain(Gorigin,Gregion,Gvoxel,Gdiv);
1008
1009   //Process 情報の生成
1010   const cio_Process* dfi_Process = dfi->GetcioProcess();
1011   out_process = new cio_Process();
1012   cio_Rank rank;
1013   if( numProc == dfi_Process->RankList.size() ) {
1014     for(int i=0; i<numProc; i++) {
1015       rank.RankID = dfi_Process->RankList[i].RankID;
1016       for(int j=0; j<3; j++) {
1017         rank.VoxelSize[j]=dfi_Process->RankList[i].VoxelSize[j];
1018         rank.HeadIndex[j]=dfi_Process->RankList[i].HeadIndex[j];
1019         rank.TailIndex[j]=dfi_Process->RankList[i].TailIndex[j];
1020       }
1021       if( thin_count > 1 ) {
1022         for(int j=0; j<3; j++) {
1023           if( rank.VoxelSize[j]%thin_count != 0 ) rank.VoxelSize[j]=rank.VoxelSize[j]/thin_count+1;
1024           else                                    rank.VoxelSize[j]=rank.VoxelSize[j];
1025           if( rank.HeadIndex[j]%thin_count != 0 ) rank.HeadIndex[j]=rank.HeadIndex[j]/thin_count+1;
1026           else                                    rank.HeadIndex[j]=rank.HeadIndex[j];
1027           rank.TailIndex[j]=rank.HeadIndex[j]+rank.VoxelSize[j]-1;
1028         }
1029       }
1030       out_process->RankList.push_back(rank);
1031     }
1032   } else if( numProc == 1 ) {
1033     rank.RankID=0;
1034     for(int i=0; i<3; i++) {
1035       rank.VoxelSize[i]=Gvoxel[i];
1036       rank.HeadIndex[i]=1;
1037       rank.TailIndex[i]=rank.HeadIndex[i]+Gvoxel[i]-1;
1038     }
1039     out_process->RankList.push_back(rank);
1040   }
1041
1042   return true;
1043 }
```

4.1.3.17 void CONV::makeRankList (vector< step_rank_info > & *StepRankList*)

rank 基準のリスト生成

引数

out	*StepRankList*	step 基準のリスト

conv.C の 761 行で定義されています。

参照先 CONV::step_rank_info::dfi, m_in_dfi, m_myRank, m_numProc, CONV::step_rank_info::rankEnd, と CONV::step_rank_info::rankStart.

参照元 convMxM::exec().

```
762 {
763
764   //総 rank 数を求める
765   int Total_rank = 0;
766   for(int i=0; i<m_in_dfi.size(); i++){
767     const cio_Process* DFI_Process = m_in_dfi[i]->GetcioProcess();
768     Total_rank+=DFI_Process->RankList.size();
769   }
770
771   //自ランクで担当する rank 数を求める
```

```
772   int nRank = Total_rank/m_numProc;
773   if( Total_rank%m_numProc != 0 ) {
774     for(int i=0; i<Total_rank%m_numProc; i++) {
775       if( m_myRank == i ) nRank++;
776     }
777   }
778
779   //自ランクが担当するランクのスタートとエンドを求める
780   int sta,end;
781   sta = m_myRank * nRank;
782   if( Total_rank%m_numProc != 0 ) {
783     if( m_myRank >= Total_rank%m_numProc ) sta = sta+Total_rank%m_numProc;
784   }
785   end = sta+nRank-1;
786
787   //処理 rank リストの生成
788   int cnt=0;
789   for(int i=0; i<m_in_dfi.size(); i++) {
790     step_rank_info info;
791     info.rankStart = -1;
792     const cio_Process* DFI_Process = m_in_dfi[i]->GetcioProcess();
793     for(int j=0; j<DFI_Process->RankList.size(); j++) {
794       if( sta > cnt ) { cnt++; continue; }
795       if( info.rankStart == -1 ) {
796         info.dfi = m_in_dfi[i];
797         info.rankStart = j;
798       }
799       info.rankEnd = j;
800       cnt++;
801       if( end < cnt ) break;
802     }
803     if( info.rankStart > -1 ) StepRankList.push_back(info);
804     if( end < cnt ) break;
805   }
806
807   //stepStart,stepEnd のセット
808   for(int i=0; i<StepRankList.size(); i++) {
809     const cio_TimeSlice* TSlice = StepRankList[i].dfi->GetcioTimeSlice();
810     StepRankList[i].stepStart=0;
811     StepRankList[i].stepEnd=TSlice->SliceList.size()-1;
812   }
813
814 }
```

4.1.3.18 void CONV::makeStepList (vector< step_rank_info > & *StepRankList*)

step 基準のリスト生成

引数

out	*StepRankList*	step 基準のリスト

conv.C の 704 行で定義されています。

参照先 CONV::step_rank_info::dfi, m_in_dfi, m_myRank, m_numProc, CONV::step_rank_info::stepEnd, と CONV::step_rank_info::stepStart.

参照元 convMxM::exec(), と convMx1::exec().

```
705 {
706
707   //総ステップ数を求める
708   int Total_step = 0;
709   for(int i=0; i<m_in_dfi.size(); i++){
710     const cio_TimeSlice* TSlice = m_in_dfi[i]->GetcioTimeSlice();
711     Total_step+=TSlice->SliceList.size();
712   }
713
714   //自ランクで担当するステップ数を求める
715   int nStep = Total_step/m_numProc;
716   if( Total_step%m_numProc != 0 ) {
717     for(int i=0; i<Total_step%m_numProc; i++) {
718       if( m_myRank == i ) nStep++;
719     }
720   }
721
722   //自ランクが担当するステップのスタートとエンドを求める
723   int sta,end;
724   sta = m_myRank * nStep;
725   if( Total_step%m_numProc != 0 ) {
```

```
726     if( m_myRank >= Total_step%m_numProc ) sta = sta+Total_step%m_numProc;
727   }
728   end = sta+nStep-1;
729
730   //処理ステップリストの生成
731   int cnt=0;
732   for(int i=0; i<m_in_dfi.size(); i++){
733     step_rank_info info;
734     info.stepStart = -1;
735     const cio_TimeSlice* TSlice = m_in_dfi[i]->GetcioTimeSlice();
736     for( int j=0; j<TSlice->SliceList.size(); j++) {
737
738       if( sta > cnt ) { cnt++; continue; }
739       if( info.stepStart == -1 ) {
740         info.dfi=m_in_dfi[i];
741         info.stepStart=j;
742       }
743       info.stepEnd = j;
744       cnt++;
745       if( end < cnt ) break;
746     }
747     if( info.stepStart > -1 ) StepRankList.push_back(info);
748     if( end < cnt ) break;
749   }
750
751   //rantStart,rankEnd のセット
752   for(int i=0; i<StepRankList.size(); i++) {
753     const cio_Process* DFI_Process = StepRankList[i].dfi->GetcioProcess();
754     StepRankList[i].rankStart=0;
755     StepRankList[i].rankEnd=DFI_Process->RankList.size()-1;
756   }
757 }
```

4.1.3.19 void CONV::MemoryRequirement (const double *Memory*, FILE * *fp*)

メモリ使用量を表示する

引数

in	*Memory*	メモリ量
in	*fp*	ファイルポインタ

conv.C の 466 行で定義されています。

参照元 convMx1::exec().

```
467 {
468   const double mem = Memory;
469   const double KB = 1024.0;
470   const double MB = 1024.0*KB;
471   const double GB = 1024.0*MB;
472   const double TB = 1024.0*GB;
473   const double PB = 1024.0*TB;
474   const double factor = 1.05; // estimate 5% for addtional
475
476   // Global memory
477   fprintf (fp," MemorySize = ");
478   if ( mem > PB ) {
479     fprintf (fp,"%6.2f (PB)\n", mem / PB *factor);
480   }
481   else if ( mem > TB ) {
482     fprintf (fp,"%6.2f (TB)\n", mem / TB *factor);
483   }
484   else if ( mem > GB ) {
485     fprintf (fp,"%6.2f (GB)\n", mem / GB *factor);
486   }
487   else if ( mem > MB ) {
488     fprintf (fp,"%6.2f (MB)\n", mem / MB *factor);
489   }
490   else if ( mem > KB ) {
491     fprintf (fp,"%6.2f (KB)\n", mem / KB *factor);
492   }
493   else if ( mem <= KB ){
494     fprintf (fp,"%6.2f (B)\n", mem *factor);
495   }
496   else {
497     fprintf (fp,"Caution! Memory required : %d (Byte)", (int)(mem *factor) );
498   }
499
500   fflush(fp);
501 }
```

4.1.3.20 void CONV::MemoryRequirement (const double *TotalMemory*, const double *sphMemory*, const double *plot3dMemory*, const double *thinMemory*, FILE * *fp*)

メモリ使用量を表示する

引数

in	*TotalMemory*	トータルメモリ使用量最大値
in	*sphMemory*	sph ファイル読み込みのための wk メモリ使用量最大値
in	*plot3dMemory*	plot3d ファイル書き込みのためのメモリ使用量最大値
in	*thinMemory*	間引きオプションのためのメモリ使用量最大値
in	*fp*	ファイルポインタ

conv.C の 505 行で定義されています。

```
506 {
507   double mem;
508   const double KB = 1024.0;
509   const double MB = 1024.0*KB;
510   const double GB = 1024.0*MB;
511   const double TB = 1024.0*GB;
512   const double PB = 1024.0*TB;
513   const double factor = 1.05; // estimate 5% for addtional
514
515   fprintf (fp,"*** Required MemorySize ***");
516   fprintf (fp,"\n");
517
518   mem = sphMemory;
519   fprintf (fp,"  read SPH MemorySize = ");
520   if ( mem > PB ) {
521     fprintf (fp,"%6.2f (PB)", mem / PB *factor);
522   }
523   else if ( mem > TB ) {
524     fprintf (fp,"%6.2f (TB)", mem / TB *factor);
525   }
526   else if ( mem > GB ) {
527     fprintf (fp,"%6.2f (GB)", mem / GB *factor);
528   }
529   else if ( mem > MB ) {
530     fprintf (fp,"%6.2f (MB)", mem / MB *factor);
531   }
532   else if ( mem > KB ) {
533     fprintf (fp,"%6.2f (KB)", mem / KB *factor);
534   }
535   else if ( mem <= KB ){
536     fprintf (fp,"%6.2f (B)", mem *factor);
537   }
538   else {
539     fprintf (fp,"Caution! Memory required : %d (Byte)", (int)(mem *factor) );
540   }
541
542   mem = plot3dMemory;
543   fprintf (fp,"  write PLOT3D MemorySize = ");
544   if ( mem > PB ) {
545     fprintf (fp,"%6.2f (PB)", mem / PB *factor);
546   }
547   else if ( mem > TB ) {
548     fprintf (fp,"%6.2f (TB)", mem / TB *factor);
549   }
550   else if ( mem > GB ) {
551     fprintf (fp,"%6.2f (GB)", mem / GB *factor);
552   }
553   else if ( mem > MB ) {
554     fprintf (fp,"%6.2f (MB)", mem / MB *factor);
555   }
556   else if ( mem > KB ) {
557     fprintf (fp,"%6.2f (KB)", mem / KB *factor);
558   }
559   else if ( mem <= KB ){
560     fprintf (fp,"%6.2f (B)", mem *factor);
561   }
562   else {
563     fprintf (fp,"Caution! Memory required : %d (Byte)", (int)(mem *factor) );
564   }
565
566   mem = thinMemory;
567   fprintf (fp,"  write thin out MemorySize = ");
568   if ( mem > PB ) {
569     fprintf (fp,"%6.2f (PB)", mem / PB *factor);
570   }
571   else if ( mem > TB ) {
572     fprintf (fp,"%6.2f (TB)", mem / TB *factor);
```

```
573   }
574   else if ( mem > GB ) {
575     fprintf (fp,"%6.2f (GB)", mem / GB *factor);
576   }
577   else if ( mem > MB ) {
578     fprintf (fp,"%6.2f (MB)", mem / MB *factor);
579   }
580   else if ( mem > KB ) {
581     fprintf (fp,"%6.2f (KB)", mem / KB *factor);
582   }
583   else if ( mem <= KB ){
584     fprintf (fp,"%6.2f (B)", mem *factor);
585   }
586   else {
587     fprintf (fp,"Caution! Memory required : %d (Byte)", (int)(mem *factor) );
588   }
589
590   mem = TotalMemory;
591   fprintf (fp,"  TotalMemorySize = ");
592   if ( mem > PB ) {
593     fprintf (fp,"%6.2f (PB)", mem / PB *factor);
594   }
595   else if ( mem > TB ) {
596     fprintf (fp,"%6.2f (TB)", mem / TB *factor);
597   }
598   else if ( mem > GB ) {
599     fprintf (fp,"%6.2f (GB)", mem / GB *factor);
600   }
601   else if ( mem > MB ) {
602     fprintf (fp,"%6.2f (MB)", mem / MB *factor);
603   }
604   else if ( mem > KB ) {
605     fprintf (fp,"%6.2f (KB)", mem / KB *factor);
606   }
607   else if ( mem <= KB ){
608     fprintf (fp,"%6.2f (B)", mem *factor);
609   }
610   else {
611     fprintf (fp,"Caution! Memory required : %d (Byte)", (int)(mem *factor) );
612   }
613
614   fprintf (fp,"\n");
615   fprintf (fp,"\n");
616
617   fflush(fp);
618 }
```

4.1.3.21 void CONV::OpenLogFile ( )

ログファイルのオープン

conv.C の 417 行で定義されています。

参照先 Exit, m_fplog, m_HostName, m_myRank, m_numProc, と m_procGrp.

参照元 main().

```
418 {
419   //log file open
420   string prefix="log_comb_id";
421   int len = prefix.size()+10;//+6+4
422   char* tmp = new char[len];
423   memset(tmp, 0, sizeof(char)*len);
424   sprintf(tmp, "%s_%06d.%s", prefix.c_str(), m_myRank, "txt");
425
426   std::string logname(tmp);
427   if ( tmp ) delete [] tmp;
428
429   if ( !(m_fplog = fopen(logname.c_str(), "w")) ){
430     printf("\tFile Open Error : '%s'\n",logname.c_str());
431     Exit(0);
432   }
433   fprintf(m_fplog,"####################\n",logname.c_str());
434   fprintf(m_fplog,"### log_comb.txt ###\n",logname.c_str());
435   fprintf(m_fplog,"####################\n",logname.c_str());
436   fprintf(m_fplog,"\n");
437
438   fprintf(m_fplog,"procGrp  = %d\n", m_procGrp);
439   fprintf(m_fplog,"myRank   = %d\n", m_myRank);
440   fprintf(m_fplog,"numProc  = %d\n", m_numProc);
441   fprintf(m_fplog,"HostName = %s\n", m_HostName.c_str());
442   fprintf(m_fplog,"\n");
```

```
443
444 }
```

4.1.3.22 void CONV::ReadDfiFiles ( )

dfi ファイルの読み込みとDfiInfo クラスデータの作成

conv.C の 81 行で定義されています。

参照先 InputParam::Get_IndfiNameList(), LOG_OUTV_, m_fplog, m_in_dfi, m_InputCntl, と STD_OUTV_.

参照元 main().

```
82 {
83   int ic=0;
84   vector<string>::const_iterator it;
85
86   vector<std::string>in_dfi_name = m_InputCntl->Get_IndfiNameList();
87
88   // dfi ファイルの読込み
89   ic=0;
90   int tempg[3];
91   int tempd[3];
92   CIO::E_CIO_ERRORCODE ret = CIO::E_CIO_SUCCESS;
93   for (it = in_dfi_name.begin(); it != in_dfi_name.end(); it++) {
94     string fname=(*it).c_str();
95     cio_DFI* dfi_in = cio_DFI::ReadInit(MPI_COMM_WORLD,
96                                         fname,
97                                         tempg,
98                                         tempd,
99                                         ret);
100    if( dfi_in == NULL ) exit(0);
101    if( ret != CIO::E_CIO_SUCCESS && ret != CIO::E_CIO_ERROR_INVALID_DIVNUM ) exit(0);
102    m_in_dfi.push_back(dfi_in);
103    ic++;
104  }
105
106
107  LOG_OUTV_ {
108    fprintf(m_fplog,"\n");
109    fprintf(m_fplog,"*** dfi file info ***\n");
110    fprintf(m_fplog,"\n");
111    //for(int i=0;i<ndfi;i++){
112    for(int i=0;i<m_in_dfi.size();i++){
113      const cio_FileInfo *DFI_Info = m_in_dfi[i]->GetcioFileInfo();
114      fprintf(m_fplog,"\tDFI_Info->DirectoryPath            = %s\n",DFI_Info->DirectoryPath.c_str());
115      fprintf(m_fplog,"\tDFI_Info->TimeSliceDirFlag         = %d\n",DFI_Info->TimeSliceDirFlag);
116      fprintf(m_fplog,"\tDFI_Info->Prefix                   = %s\n",DFI_Info->Prefix.c_str());
117      fprintf(m_fplog,"\tDFI_Info->FileFormat               = %d\n",DFI_Info->FileFormat);
118      fprintf(m_fplog,"\tDFI_Info->GuideCell                = %d\n",DFI_Info->GuideCell);
119      fprintf(m_fplog,"\tDFI_Info->DataType                 = %d\n",DFI_Info->DataType);
120      fprintf(m_fplog,"\tDFI_Info->Endian                   = %d\n",DFI_Info->Endian);
121      fprintf(m_fplog,"\tDFI_Info->ArrayShape               = %d\n",DFI_Info->ArrayShape);
122      fprintf(m_fplog,"\tDFI_Info->Component                = %d\n",DFI_Info->Component);
123
124      const cio_MPI *DFI_MPI = m_in_dfi[i]->GetcioMPI();
125      fprintf(m_fplog,"\tDFI_MPI->NumberOfRank              = %d\n",DFI_MPI->NumberOfRank);
126      fprintf(m_fplog,"\tDFI_MPI->NumberOfGroup             = %d\n",DFI_MPI->NumberOfGroup);
127
128      const cio_Domain *DFI_Domain = m_in_dfi[i]->GetcioDomain();
129      fprintf(m_fplog,"\tDFI_Domain->GlobalVoxel[0]         = %d\n",DFI_Domain->GlobalVoxel[0]);
130      fprintf(m_fplog,"\tDFI_Domain->GlobalVoxel[1]         = %d\n",DFI_Domain->GlobalVoxel[1]);
131      fprintf(m_fplog,"\tDFI_Domain->GlobalVoxel[2]         = %d\n",DFI_Domain->GlobalVoxel[2]);
132      fprintf(m_fplog,"\tDFI_Domain->GlobalDivision[0]      = %d\n",DFI_Domain->GlobalDivision[0]);
133      fprintf(m_fplog,"\tDFI_Domain->GlobalDivision[1]      = %d\n",DFI_Domain->GlobalDivision[1]);
134      fprintf(m_fplog,"\tDFI_Domain->GlobalDivision[2]      = %d\n",DFI_Domain->GlobalDivision[2]);
135
136      const cio_Process *DFI_Process = m_in_dfi[i]->GetcioProcess();
137      fprintf(m_fplog,"\n");
138      fprintf(m_fplog,"\tDFI_Process->RankList.size()       = %d\n",DFI_Process->RankList.size());
139      for(int j=0; j< DFI_Process->RankList.size(); j++ ) {
140        fprintf(m_fplog,"\t  DFI_Process->RankList[%d].RankID       = %d\n",j,DFI_Process->RankList[j].
    RankID);
141        fprintf(m_fplog,"\t  DFI_Process->RankList[%d].HostName     = %s\n",j,DFI_Process->RankList[j].
    HostName.c_str());
142        fprintf(m_fplog,"\t  DFI_Process->RankList[%d].VoxelSize[0] = %d\n",j,DFI_Process->RankList[j].
    VoxelSize[0]);
143        fprintf(m_fplog,"\t  DFI_Process->RankList[%d].VoxelSize[1] = %d\n",j,DFI_Process->RankList[j].
    VoxelSize[1]);
144        fprintf(m_fplog,"\t  DFI_Process->RankList[%d].VoxelSize[2] = %d\n",j,DFI_Process->RankList[j].
    VoxelSize[2]);
145        fprintf(m_fplog,"\t  DFI_Process->RankList[%d].HeadIndex[0] = %d\n",j,DFI_Process->RankList[j].
```

```
    HeadIndex[0]);
146         fprintf(m_fplog,"\t  DFI_Process->RankList[%d].HeadIndex[1] = %d\n",j,DFI_Process->RankList[j].
    HeadIndex[1]);
147         fprintf(m_fplog,"\t  DFI_Process->RankList[%d].HeadIndex[2] = %d\n",j,DFI_Process->RankList[j].
    HeadIndex[2]);
148         fprintf(m_fplog,"\t  DFI_Process->RankList[%d].TailIndex[0] = %d\n",j,DFI_Process->RankList[j].
    TailIndex[0]);
149         fprintf(m_fplog,"\t  DFI_Process->RankList[%d].TailIndex[1] = %d\n",j,DFI_Process->RankList[j].
    TailIndex[1]);
150         fprintf(m_fplog,"\t  DFI_Process->RankList[%d].TailIndex[2] = %d\n",j,DFI_Process->RankList[j].
    TailIndex[2]);
151       }
152
153       fprintf(m_fplog,"\n");
154       const cio_TimeSlice* DFI_TSlice = m_in_dfi[i]->GetcioTimeSlice();
155       for(int j=0; j< DFI_TSlice->SliceList.size(); j++ ) {
156         fprintf(m_fplog,"\t  DFI_TSlice->SliceList[%d].step         = %d\n",j,DFI_TSlice->SliceList[j].step
    );
157         fprintf(m_fplog,"\t  DFI_TSlice->SliceList[%d].time         = %f\n",j,DFI_TSlice->SliceList[j].time
    );
158       }
159     }
160     fprintf(m_fplog,"\n");
161   }
162
163   STD_OUTV_ {
164     printf("\n");
165     printf("*** dfi file info ***\n");
166     printf("\n");
167     for(int i=0;i<m_in_dfi.size();i++){
168       const cio_FileInfo *DFI_Info = m_in_dfi[i]->GetcioFileInfo();
169       printf("\tDFI_Info->DirectoryPath            = %s\n",DFI_Info->DirectoryPath.c_str());
170       printf("\tDFI_Info->TimeSliceDirFlag         = %d\n",DFI_Info->TimeSliceDirFlag);
171       printf("\tDFI_Info->Prefix                   = %s\n",DFI_Info->Prefix.c_str());
172       printf("\tDFI_Info->FileFormat               = %d\n",DFI_Info->FileFormat);
173       printf("\tDFI_Info->GuideCell                = %d\n",DFI_Info->GuideCell);
174       printf("\tDFI_Info->DataType                 = %d\n",DFI_Info->DataType);
175       printf("\tDFI_Info->Endian                   = %d\n",DFI_Info->Endian);
176       printf("\tDFI_Info->ArrayShape               = %d\n",DFI_Info->ArrayShape);
177       printf("\tDFI_Info->Component                = %d\n",DFI_Info->Component);
178
179       const cio_MPI *DFI_MPI = m_in_dfi[i]->GetcioMPI();
180       printf("\tDFI_MPI->NumberOfRank              = %d\n",DFI_MPI->NumberOfRank);
181       printf("\tDFI_MPI->NumberOfGroup             = %d\n",DFI_MPI->NumberOfGroup);
182
183       const cio_Domain *DFI_Domain = m_in_dfi[i]->GetcioDomain();
184       printf("\tDFI_Domain->GlobalVoxel[0]         = %d\n",DFI_Domain->GlobalVoxel[0]);
185       printf("\tDFI_Domain->GlobalVoxel[1]         = %d\n",DFI_Domain->GlobalVoxel[1]);
186       printf("\tDFI_Domain->GlobalVoxel[2]         = %d\n",DFI_Domain->GlobalVoxel[2]);
187       printf("\tDFI_Domain->GlobalDivision[0]      = %d\n",DFI_Domain->GlobalDivision[0]);
188       printf("\tDFI_Domain->GlobalDivision[1]      = %d\n",DFI_Domain->GlobalDivision[1]);
189       printf("\tDFI_Domain->GlobalDivision[2]      = %d\n",DFI_Domain->GlobalDivision[2]);
190
191       const cio_Process *DFI_Process = m_in_dfi[i]->GetcioProcess();
192       printf("\n");
193       printf("\tDFI_Process->RankList.size()       = %d\n",(int)DFI_Process->RankList.size());
194       for(int j=0; j< DFI_Process->RankList.size(); j++ ) {
195         printf("\t  DFI_Process->RankList[%d].RankID       = %d\n",j,DFI_Process->RankList[j].RankID);
196         printf("\t  DFI_Process->RankList[%d].HostName     = %s\n",j,DFI_Process->RankList[j].HostName.
    c_str());
197         printf("\t  DFI_Process->RankList[%d].VoxelSize[0] = %d\n",j,DFI_Process->RankList[j].VoxelSize[0])
    ;
198         printf("\t  DFI_Process->RankList[%d].VoxelSize[1] = %d\n",j,DFI_Process->RankList[j].VoxelSize[1])
    ;
199         printf("\t  DFI_Process->RankList[%d].VoxelSize[2] = %d\n",j,DFI_Process->RankList[j].VoxelSize[2])
    ;
200         printf("\t  DFI_Process->RankList[%d].HeadIndex[0] = %d\n",j,DFI_Process->RankList[j].HeadIndex[0])
    ;
201         printf("\t  DFI_Process->RankList[%d].HeadIndex[1] = %d\n",j,DFI_Process->RankList[j].HeadIndex[1])
    ;
202         printf("\t  DFI_Process->RankList[%d].HeadIndex[2] = %d\n",j,DFI_Process->RankList[j].HeadIndex[2])
    ;
203         printf("\t  DFI_Process->RankList[%d].TailIndex[0] = %d\n",j,DFI_Process->RankList[j].TailIndex[0])
    ;
204         printf("\t  DFI_Process->RankList[%d].TailIndex[1] = %d\n",j,DFI_Process->RankList[j].TailIndex[1])
    ;
205         printf("\t  DFI_Process->RankList[%d].TailIndex[2] = %d\n",j,DFI_Process->RankList[j].TailIndex[2])
    ;
206       }
207
208       printf("\n");
209       const cio_TimeSlice* DFI_TSlice = m_in_dfi[i]->GetcioTimeSlice();
210       for(int j=0; j< DFI_TSlice->SliceList.size(); j++ ) {
211         printf("\t  DFI_TSlice->SliceList[%d].step         = %d\n",j,DFI_TSlice->SliceList[j].step);
212         printf("\t  DFI_TSlice->SliceList[%d].time         = %f\n",j,DFI_TSlice->SliceList[j].time);
213       }
214
```

```
215     }
216   }
217
218 }
```

4.1.3.23 void CONV::setRankInfo ( ) [inline]

ランク情報をセットする

conv.h の 142 行で定義されています。

```
143   {
144     m_procGrp = 0;
145     m_myRank  = m_paraMngr->GetMyRankID();
146     m_numProc = m_paraMngr->GetNumRank();
147     m_HostName= m_paraMngr->GetHostName();
148   }
```

4.1.3.24 virtual void CONV::VoxelInit ( ) [inline], [virtual]

領域分割と出力DFI のインスタンス

convMxNで再定義されています。

conv.h の 220 行で定義されています。

参照元 main().

```
220 { return; }
```

4.1.3.25 bool CONV::WriteIndexDfiFile (vector< dfi_MinMax * > *minmaxList*)

index.dfi の出力

引数

minmaxList	minmax のリスト

conv.C の 861 行で定義されています。

参照先 InputParam::Get_OutdfiNameList(), InputParam::Get_OutprocNameList(), InputParam::Get_OutputArrayShape(), InputParam::Get_OutputDataType(), InputParam::Get_OutputDir(), InputParam::Get_OutputFormat(), InputParam::Get_OutputGuideCell(), と m_InputCntl.

参照元 convMxM::exec(), と convMx1::exec().

```
862 {
863
864   //出力 dfi ファイル名の取得
865   vector<std::string> out_dfi_name  = m_InputCntl->Get_OutdfiNameList();
866   vector<std::string> out_proc_name = m_InputCntl->Get_OutprocNameList();
867
868   if( minmaxList.size() != out_dfi_name.size() &&
869       minmaxList.size() != out_proc_name.size() ) return false;
870
871   for(int i=0; i<minmaxList.size(); i++) {
872
873     cio_DFI* dfi = minmaxList[i]->dfi;
874
875     printf("dfiname : %s procname : %s\n",out_dfi_name[i].c_str(),out_proc_name[i].c_str());
876
877     FILE* fp=NULL;
878     if( !(fp = fopen(out_dfi_name[i].c_str(), "w")) )
879     {
880       printf("Can't open file.(%s)\n", out_dfi_name[i].c_str());
881       return false;
882     }
883
```

```
884     //成分数の取得
885     int nComp = dfi->GetNumComponent();
886
887     cio_FileInfo *dfi_Finfo = (cio_FileInfo *)dfi->GetcioFileInfo();
888
889     CIO::E_CIO_ARRAYSHAPE shape = dfi_Finfo->ArrayShape;
890     //if( dfi_Finfo->FileFormat == (CIO::E_CIO_FMT_BOV) ) {
891     if( (CIO::E_CIO_FORMAT)m_InputCntl->Get_OutputFormat() == (CIO::E_CIO_FMT_BOV) ) {
892       shape = (CIO::E_CIO_ARRAYSHAPE)m_InputCntl->Get_OutputArrayShape();
893     }
894
895     //FileInfo の出力
896     cio_FileInfo *Finfo = new cio_FileInfo(m_InputCntl->Get_OutputDir(),
897                                dfi_Finfo->TimeSliceDirFlag,
898                                dfi_Finfo->Prefix,
899                                (CIO::E_CIO_FORMAT)m_InputCntl->Get_OutputFormat(),
900                                m_InputCntl->Get_OutputGuideCell(),
901                                (CIO::E_CIO_DTYPE)m_InputCntl->Get_OutputDataType(),
902                                dfi_Finfo->Endian,
903                                //(CIO::E_CIO_ARRAYSHAPE)m_InputCntl->Get_OutputArrayShape(),
904                                shape,
905                                nComp);
906
907     for(int n=0; n<nComp; n++) {
908        std::string variable = dfi->getComponentVariable(n);
909        if( variable != "" ) Finfo->setComponentVariable(n,variable);
910     }
911
912     if( Finfo->Write(fp, 0) != CIO::E_CIO_SUCCESS ) {
913       fclose(fp);
914       return false;
915     }
916     delete Finfo;
917
918     //FilePath の出力
919     cio_FilePath *dfi_Fpath = (cio_FilePath *)dfi->GetcioFilePath();
920     cio_FilePath *Fpath = new cio_FilePath(out_proc_name[i]);
921     if( Fpath->Write(fp, 1) != CIO::E_CIO_SUCCESS )
922     {
923       fclose(fp);
924       return false;
925     }
926     delete Fpath;
927
928     //Unit の出力
929     cio_Unit *dfi_Unit = (cio_Unit *)dfi->GetcioUnit();
930     if( dfi_Unit->Write(fp, 0) != CIO::E_CIO_SUCCESS )
931     {
932       fclose(fp);
933       return false;
934     }
935
936     //TimeSlice の出力
937     const cio_TimeSlice *dfi_TSlice = dfi->GetcioTimeSlice();
938     cio_TimeSlice *TSlice = new cio_TimeSlice();
939     int nsize = nComp;
940     if( nComp > 1 ) nsize++;
941     double* minmax = new double[nsize*2];
942     for(int j=0; j<dfi_TSlice->SliceList.size(); j++) {
943       for(int n=0; n<nsize; n++) {
944         minmax[n*2+0] = minmaxList[i]->Min[j*nsize+n];
945         minmax[n*2+1] = minmaxList[i]->Max[j*nsize+n];
946       }
947       TSlice->AddSlice(dfi_TSlice->SliceList[j].step,
948                        dfi_TSlice->SliceList[j].time,
949                        minmax,
950                        nComp,
951                        true,
952                        0,
953                        0.0);
954
955     }
956
957     if( TSlice->Write(fp, 1) != CIO::E_CIO_SUCCESS )
958     {
959       fclose(fp);
960       return false;
961     }
962
963     //fclose(fp);
964
965   }
966
967   return true;
968 }
```

4.1.3.26 bool CONV::WriteProcDfiFile (std::string *proc_name,* cio_Domain * *out_domain,* cio_MPI * *out_mpi,* cio_Process * *out_process*)

proc.dfi の出力

conv.C の 1048 行で定義されています。

参照元 convMxM::exec(), と convMx1::exec().

```
1052 {
1053
1054   FILE* fp = NULL;
1055
1056   if( out_domain == NULL || out_mpi == NULL || out_process == NULL ) return false;
1057
1058   //proc.dfi ファイルオープン
1059   if( !(fp = fopen(proc_name.c_str(), "w")) )
1060   {
1061      printf("Can't open file.(%s)\n", proc_name.c_str());
1062      return false;
1063   }
1064
1065   //Domain {} の出力
1066   if( out_domain->Write(fp, 0) != CIO::E_CIO_SUCCESS )
1067   {
1068     fclose(fp);
1069     return false;
1070   }
1071
1072   //MPI {} の出力
1073   if( out_mpi->Write(fp, 0) != CIO::E_CIO_SUCCESS )
1074   {
1075     fclose(fp);
1076     return false;
1077   }
1078
1079   //Process {} の出力
1080   if( out_process->Write(fp, 0) != CIO::E_CIO_SUCCESS )
1081   {
1082     fclose(fp);
1083     return false;
1084   }
1085
1086   fclose(fp);
1087
1088   return true;
1089 }
```

4.1.3.27 void CONV::WriteTime (double * *tt*)

所要時間の記述

引数

in	*tt*	所要時間

conv.C の 455 行で定義されています。

参照先 m_fplog.

参照元 main().

```
456 {
457   fprintf(m_fplog,"\n\n");
458   fprintf(m_fplog,"TIME : ReadInit      %10.3f sec.\n", tt[0]);
459   fprintf(m_fplog,"TIME : ReadDfiFiles  %10.3f sec.\n", tt[1]);
460   fprintf(m_fplog,"TIME : Converter     %10.3f sec.\n", tt[2]);
461   fprintf(m_fplog,"TIME : Total Time    %10.3f sec.\n", tt[3]);
462 }
```

4.1.4 変数

4.1.4.1 bool CONV::m_bgrid_interp_flag

節点への補間フラグ

conv.h の 90 行で定義されています。

参照元 CONV(), ConvInit(), convMx1::convMx1_out_ijkn(), convMx1::convMx1_out_nijk(), convMx1::exec(), convMxN::exec(), convMxM::mxmsolv(), と convMxN::VoxelInit().

4.1.4.2 FILE* CONV::m_fplog [protected]

conv.h の 115 行で定義されています。

参照元 CloseLogFile(), convMx1::exec(), OpenLogFile(), ReadDfiFiles(), と WriteTime().

4.1.4.3 std::string CONV::m_HostName

ホスト名

conv.h の 88 行で定義されています。

参照元 convMxM::mxmsolv(), OpenLogFile(), と convMxN::VoxelInit().

4.1.4.4 vector<cio_DFI *> CONV::m_in_dfi

conv.h の 104 行で定義されています。

参照元 CheckConvData(), CONV(), convMx1::convMx1_out_ijkn(), convMx1::convMx1_out_nijk(), convMxM::exec(), convMx1::exec(), convMxN::exec(), makeRankList(), makeStepList(), ReadDfiFiles(), convMxN::VoxelInit(), と ~CONV().

4.1.4.5 InputParam* CONV::m_InputCntl

InputParam Class.

conv.h の 82 行で定義されています。

参照元 CheckConvData(), convMx1::convMx1_out_ijkn(), convMx1::convMx1_out_nijk(), copyArray(), convMxM::exec(), convMx1::exec(), convMxN::exec(), makeProcInfo(), convMxM::mxmsolv(), ReadDfiFiles(), convMxN::VoxelInit(), と WriteIndexDfiFile().

4.1.4.6 int CONV::m_lflag

conv.h の 101 行で定義されています。

参照元 CONV(), と main().

4.1.4.7 int CONV::m_lflagv

conv.h の 102 行で定義されています。

参照元 CONV(), と main().

4.1.4.8 int CONV::m_myRank

自ノードのランク番号

conv.h の 86 行で定義されています。

参照元 CONV(), convMxM::exec(), convMx1::exec(), makeRankList(), makeStepList(), OpenLogFile(), と convMxN::VoxelInit().

4.1.4.9 int CONV::m_numProc

全ランク数

conv.h の 87 行で定義されています。

参照元 CONV(), convMxN::exec(), makeRankList(), makeStepList(), と OpenLogFile().

4.1.4.10 cpm_ParaManager* CONV::m_paraMngr

Cartesian Partition Manager.

conv.h の 80 行で定義されています。

参照元 convMxN::exec(), と convMxN::VoxelInit().

4.1.4.11 int CONV::m_pflag

conv.h の 99 行で定義されています。

参照元 CONV(), と main().

4.1.4.12 int CONV::m_pflagv

conv.h の 100 行で定義されています。

参照元 CONV(), と main().

4.1.4.13 int CONV::m_procGrp

プロセスグループ番号

conv.h の 85 行で定義されています。

参照元 CONV(), と OpenLogFile().

4.1.4.14 unsigned CONV::m_staging [private]

conv.h の 95 行で定義されています。

参照元 CONV().

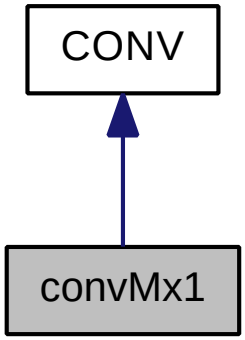
このクラスの説明は次のファイルから生成されました:

- conv.h
- conv.C
- conv_inline.h

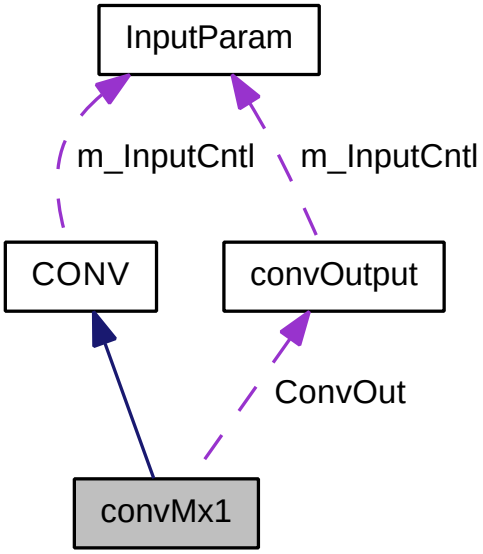
4.2 クラス convMx1

```
#include <convMx1.h>
```

convMx1 に対する継承グラフ



convMx1 のコラボレーション図



Public 型

- typedef std::map< int, int > headT

Public メソッド

- convMx1 ()
- ~convMx1 ()
- bool exec ()

  Mx1 **の実行**
- bool convMx1_out_nijk (FILE *fp, std::string inPath, int l_step, double l_dtime, int d_type, bool mio, int div[3], int sz[3], cio_DFI *dfi, cio_Process *DFI_Process, headT mapHeadX, headT mapHeadY, headT mapHeadZ, double *min, double *max)

  並列形状 *nijk* **を** *nijk* **でコンバートして出力**
- bool convMx1_out_ijkn (FILE *fp, std::string inPath, int l_step, double l_dtime, int d_type, bool mio, int div[3], int sz[3], cio_DFI *dfi, cio_Process *DFI_Process, headT mapHeadX, headT mapHeadY, headT mapHeadZ, double *min, double *max)

  並列形状 *nijk* **を** *ijkn* **または** *ijkn* **を** *ijkn* **にコンバートして出力**

- bool InterPolate (cio_Array *src_old, cio_Array *src, cio_Array *outArray, int ivar_src, int ivar_out)
  補間処理
- template<class T >
  void zeroClearArray (cio_TypeArray< T > *data, int ivar_out)
  配列のゼロクリア
- template<class T >
  bool setGridData_XY (cio_TypeArray< T > *O, cio_TypeArray< T > *S, int ivar_out, int ivar_src)
  図心データを格子点に補間
- template<class T >
  void VolumeDataDivide8 (cio_TypeArray< T > *O, int ivar_out)
  内部の格子点のデータを重み付けでで割る
- cio_Array * nijk_to_ijk (cio_Array *src, int ivar)
  NIJK **配列をスカラーの***IJK* **配列にコピーコントロール**
- template<class T >
  void copyArray_nijk_ijk (cio_TypeArray< T > *S, cio_TypeArray< T > *O, int ivar)
  NIJK **配列をスカラーの***IJK* **配列にコピー**
- template<class T >
  CONV_INLINE void zeroClearArray (cio_TypeArray< T > *data, int ivar_out)
  配列のゼロクリア
- template<class T >
  CONV_INLINE bool setGridData_XY (cio_TypeArray< T > *O, cio_TypeArray< T > *S, int ivar_out, int ivar_src)
  Scalar **の格子点での値をセット**
- template<class T >
  CONV_INLINE void VolumeDataDivide8 (cio_TypeArray< T > *O, int n)
- template<class T >
  CONV_INLINE void copyArray_nijk_ijk (cio_TypeArray< T > *S, cio_TypeArray< T > *O, int ivar)

Public 変数

- convOutput * ConvOut
- vector< step_rank_info > m_StepRankList
  並列処理用インデックスリスト

Additional Inherited Members

4.2.1 説明

convMx1.h の 24 行で定義されています。

4.2.2 型定義

4.2.2.1 typedef std::map<int,int> convMx1::headT

convMx1.h の 30 行で定義されています。

4.2.3 コンストラクタとデストラクタ

4.2.3.1 convMx1::convMx1 ( )

コンストラクタ

convMx1.C の 21 行で定義されています。

参照先 m_StepRankList.

```
22 {
23
24   m_StepRankList.clear();
25
26 }
```

4.2.3.2 convMx1::∼convMx1 ( )

デストラクタ

convMx1.C の 30 行で定義されています。

```
31 {
32
33 }
```

4.2.4 関数

4.2.4.1 bool convMx1::convMx1_out_ijkn (FILE ∗ *fp*, std::string *inPath*, int *l_step*, double *l_dtime*, int *d_type*, bool *mio*, int *div[3]*, int *sz[3]*, cio_DFI ∗ *dfi*, cio_Process ∗ *DFI_Process*, headT *mapHeadX*, headT *mapHeadY*, headT *mapHeadZ*, double ∗ *min*, double ∗ *max*)

並列形状 nijk を ijkn または ijkn を ijkn にコンバートして出力

引数

in	*fp*	出力ファイルポインタ
in	*inPath*	dfi のディレクトリパス
in	*l_step*	出力 step 番号
in	*l_dtime*	出力時刻
in	*d_type*	データタイプ
in	*mio*	分割出力指示
in	*div*	分割数
in	*sz*	サイズ
in	*dfi*	dfi
in	*DFI_Process*	cio_Process
in	*mapHeadX*	
in	*mapHeadY*	
in	*mapHeadZ*	
out	*min*	最小値
out	*max*	最大値

Loop itx

Loop ity

Loop kp

Loop itz

Loop n

convMx1.C の 674 行で定義されています。

参照先 CONV::convertXY(), ConvOut, CONV::DtypeMinMax(), InputParam::Get_OutputGuideCell(), InputParam::Get_ThinOut(), InterPolate(), CONV::m_bgrid_interp_flag, CONV::m_in_dfi, CONV::m_InputCntl, nijk_to_ijk(), と convOutput::WriteFieldData().

参照元 exec().

```
686 {
687
688   cio_Domain* DFI_Domian = (cio_Domain *)m_in_dfi[0]->GetcioDomain();
689
690   int thin_count = m_InputCntl->Get_ThinOut();
691
692   int outGc=0;
693   int interp_Gc=0;
694
695   //出力ガイドセルの設定
696   if( m_InputCntl->Get_OutputGuideCell() > 1 ) outGc = m_InputCntl->
      Get_OutputGuideCell();
697   if( outGc > 1 ) {
698     const cio_FileInfo* DFI_FInfo = dfi->GetcioFileInfo();
699     if( outGc > DFI_FInfo->GuideCell ) outGc=DFI_FInfo->GuideCell;
700   }
701
702   //間引きありのとき、出力ガイドセルを 0 に設定
703   if( thin_count > 1 ) outGc=0;
704   interp_Gc = outGc;
705
706   //格子点出力のときガイドセルが 0 のとき 1 にセット
707   if( m_bgrid_interp_flag && outGc==0 ) interp_Gc=1;
708
709   //cell 出力のとき、出力ガイドセルを 0 に設定
710   if( !m_bgrid_interp_flag ) interp_Gc=0;
711
712   //出力のヘッダー、フッターをセット
713   int headS[3],tailS[3];
714   headS[0]=0-outGc;
715   headS[1]=0-outGc;
716   //tailS[0]=headS[0]+sz[0]+outGc-1;
717   //tailS[1]=headS[1]+sz[1]+outGc-1;
718   tailS[0]=headS[0]+DFI_Domian->GlobalVoxel[0]+outGc-1;
719   tailS[1]=headS[1]+DFI_Domian->GlobalVoxel[1]+outGc-1;
720
721   //成分数の取り出し
722   int nComp = dfi->GetNumComponent();
723
724   //出力バッファのインスタンス（読込み配列形状での DFI でインスタンス）
725   cio_Array* src = cio_Array::instanceArray
726                    ( (CIO::E_CIO_DTYPE)d_type
727                    , dfi->GetArrayShape()
728                    , sz
729                    , outGc
730                    , nComp );
731
732   //補間用バッファ（読込み配列形状での DFI でインスタンス）
733   cio_Array* src_old = NULL;
734   cio_Array* outArray = NULL;
735   if( m_bgrid_interp_flag ) {
736     src_old = cio_Array::instanceArray
737               ( (CIO::E_CIO_DTYPE)d_type
738               , dfi->GetArrayShape()
739               , sz
740               , outGc
741               , nComp );
742
743     int szOut[3];
744     for(int i=0; i<2; i++) szOut[i]=sz[i]+1;
745     szOut[2]=sz[2];
746     outArray = cio_Array::instanceArray
747              ( (CIO::E_CIO_DTYPE)d_type
748              , dfi->GetArrayShape()
749              , szOut
750              , outGc
751              , 1 );
752   }
753
754   int kdiv,jdiv,idiv;
755   int l_rank;
756   std::string infile;
757
758   //成分数のループ
759   for(int n=0; n<nComp; n++) {
760
761     //z 方向の分割数回のループ
762     for( headT::iterator itz=mapHeadZ.begin(); itz!= mapHeadZ.end(); itz++ ) {
763
764       //z 層のスタートエンドを設定
765       kdiv = itz->second;
766       int kp_sta,kp_end;
767       kp_sta = itz->first;
768       int nrank = _CIO_IDX_IJK(0,0,kdiv,div[0],div[1],div[2],0);
769       kp_end = kp_sta + DFI_Process->RankList[nrank].VoxelSize[2];
770
771       //z 層のスタートエンドをガイドセルの考慮
```

```
772       if( kdiv == 0 ) kp_sta = kp_sta-outGc;
773       if( kdiv == div[2]-1 ) kp_end = kp_end+outGc;
774
775       //同一 Z 面のループ
776       for(int kp=kp_sta; kp< kp_end; kp++) {
777
778         int kk = kp-1;
779         //間引きの層のときスキップ
780         if( kk%thin_count != 0 ) continue;
781
782         //y 方向の分割数のループ
783         for( headT::iterator ity=mapHeadY.begin(); ity!= mapHeadY.end(); ity++ ) {
784
785           //y のスタートエンドの設定
786           jdiv = ity->second;
787           int jp_sta,jp_end;
788           jp_sta = ity->first;
789           int nrank = _CIO_IDX_IJK(0,jdiv,kdiv,div[0],div[1],div[2],0);
790           jp_end = jp_sta + DFI_Process->RankList[nrank].VoxelSize[1];
791
792           //x 方向の分割数のループ
793           for( headT::iterator itx=mapHeadX.begin(); itx!= mapHeadX.end(); itx++ ) {
794
795             //x のスタートエンドの設定
796             idiv = itx->second;
797             int ip_sta,ip_end;
798             ip_sta = itx->first;
799             int nrank = _CIO_IDX_IJK(idiv,jdiv,kdiv,div[0],div[1],div[2],0);
800             ip_end = ip_sta + DFI_Process->RankList[nrank].VoxelSize[0];
801
802             int RankID = _CIO_IDX_IJK(idiv,jdiv,kdiv,div[0],div[1],div[2],0);
803
804             int read_sta[3],read_end[3];
805             read_sta[0]=ip_sta;
806             read_sta[1]=jp_sta;
807             read_sta[2]=kp;
808             read_end[0]=ip_end-1;
809             read_end[1]=jp_end-1;
810             read_end[2]=kp;
811
812             //ガイドセルを考慮して読込み範囲を更新
813             if( idiv == 0 ) read_sta[0] = read_sta[0]-outGc;
814             if( idiv == div[0]-1 ) read_end[0] = read_end[0]+outGc;
815             if( jdiv == 0 ) read_sta[1] = read_sta[1]-outGc;
816             if( jdiv == div[1]-1 ) read_end[1] = read_end[1]+outGc;
817
818             l_rank=DFI_Process->RankList[RankID].RankID;
819             //連結対象ファイル名の生成
820             infile = CIO::cioPath_ConnectPath(inPath,dfi->Generate_FieldFileName(l_rank,l_step,mio));
821             unsigned int avr_step;
822             double avr_time;
823             CIO::E_CIO_ERRORCODE ret;
824             //連結対象ファイルの読込み
825             cio_Array* buf = dfi->ReadFieldData(infile, l_step, l_dtime,
826                                                 read_sta, read_end,
827                                                 DFI_Process->RankList[RankID].HeadIndex,
828                                                 DFI_Process->RankList[RankID].TailIndex,
829                                                 true, avr_step, avr_time, ret);
830             //headIndex を０スタートにしてセット
831             int headB[3];
832             headB[0]=read_sta[0]-1;
833             headB[1]=read_sta[1]-1;
834             headB[2]=read_sta[2]-1;
835             buf->setHeadIndex( headB );
836
837             int headS0[3];
838             headS0[0]=headS[0];
839             headS0[1]=headS[1];
840             headS0[2]=kk/thin_count;
841             src->setHeadIndex( headS0 );
842
843             headS0[2]=kk;
844             tailS[2]=headS0[2];
845
846             //出力配列へのコンバイン
847             convertXY(buf,src,headS0,tailS,n);
848
849             //minmax を求める
850             if( n==0 ) if( !DtypeMinMax(buf,min,max) ) return false;
851
852             delete buf;
853
854           }
855         }
856         //補間処理
857         if( m_bgrid_interp_flag ) {
858           if( kp == kp_sta ) {
```

```
859             if( !InterPolate(src,src,outArray,n,0) ) return false;
860           } else {
861             if( !InterPolate(src_old,src,outArray,n,0) ) return false;
862           }
863         } else {
864           //NIJK レコードを IJK にコピー
865           outArray = nijk_to_ijk(src,n);
866
867         }
868
869         //一層分出力
870         if( outArray ) {
871           const int* szOutArray = outArray->getArraySizeInt();
872           size_t dLen = szOutArray[0]*szOutArray[1]*szOutArray[2]*outArray->getNcomp();
873           if( ConvOut->WriteFieldData(fp,
874                                       outArray,
875                                       dLen ) != true ) return false;
876         }
877         //補間ありのとき、読込んだ層の配列ポインタを src_old にコピー
878         if( m_bgrid_interp_flag ) {
879           cio_Array* tmp = src;
880           src = src_old;
881           src_old = tmp;
882         }
883       }
884     }
885
886     if( m_bgrid_interp_flag ) {
887       for(int n=0; n<nComp; n++) {
888         if( !InterPolate(src_old,src_old,outArray,n,0) ) return false;
889       }
890       if( outArray ) {
891         const int* szOutArray = outArray->getArraySizeInt();
892         size_t dLen = szOutArray[0]*szOutArray[1]*szOutArray[2]*outArray->getNcomp();
893         if( ConvOut->WriteFieldData(fp,
894                                     outArray,
895                                     dLen ) != true ) return false;
896       } else return false;
897     }
898   }
899
900   delete src;
901   if( m_bgrid_interp_flag ) {
902     delete src_old;
903     delete outArray;
904   }
905
906   return true;
907
908 }
```

4.2.4.2 bool convMx1::convMx1_out_nijk (FILE * *fp*, std::string *inPath*, int *l_step*, double *l_dtime*, int *d_type*, bool *mio*, int *div[3]*, int *sz[3]*, cio_DFI * *dfi*, cio_Process * *DFI_Process*, headT *mapHeadX*, headT *mapHeadY*, headT *mapHeadZ*, double * *min*, double * *max*)

並列形状 nijk を nijk でコンバートして出力

引数

in	*fp*	出力ファイルポインタ
in	*inPath*	dfi のディレクトリパス
in	*l_step*	出力 step 番号
in	*l_dtime*	出力時刻
in	*d_type*	データタイプ
in	*mio*	分割出力指示
in	*div*	分割数
in	*sz*	サイズ
in	*dfi*	dfi

in	*DFI_Process*	cio_Process
in	*mapHeadX*	
in	*mapHeadY*	
in	*mapHeadZ*	
out	*min*	最小値
out	*max*	最大値

Loop itx

Loop ity

Loop kp

Loop itz

convMx1.C の 420 行で定義されています。

参照先 CONV::convertXY(), ConvOut, CONV::DtypeMinMax(), InputParam::Get_OutputGuideCell(), InputParam::Get_ThinOut(), InterPolate(), CONV::m_bgrid_interp_flag, CONV::m_in_dfi, CONV::m_InputCntl, と convOutput::WriteFieldData().

参照元 exec().

```
432 {
433
434   cio_Domain* DFI_Domian = (cio_Domain *)m_in_dfi[0]->GetcioDomain();
435
436   int thin_count = m_InputCntl->Get_ThinOut();
437
438   int outGc=0;
439   int interp_Gc=0;
440
441   //出力ガイドセルの設定
442   if( m_InputCntl->Get_OutputGuideCell() > 1 ) outGc = m_InputCntl->
      Get_OutputGuideCell();
443   if( outGc > 0 ) {
444     const cio_FileInfo* DFI_FInfo = dfi->GetcioFileInfo();
445     if( outGc > DFI_FInfo->GuideCell ) outGc=DFI_FInfo->GuideCell;
446   }
447
448   //間引きありのとき、出力ガイドセルを 0 に設定
449   if( thin_count > 1 ) outGc=0;
450   interp_Gc = outGc;
451
452   //格子点出力のときガイドセルが 0 のとき 1 にセット
453   if( m_bgrid_interp_flag && outGc==0 ) interp_Gc=1;
454
455   //cell 出力のとき、出力ガイドセルを 0 に設定
456   if( !m_bgrid_interp_flag ) interp_Gc=0;
457
458   //出力のヘッダー、フッターをセット
459   int headS[3],tailS[3];
460   headS[0]=0-outGc;
461   headS[1]=0-outGc;
462   //tailS[0]=headS[0]+sz[0]+outGc-1;
463   //tailS[1]=headS[1]+sz[1]+outGc-1;
464   tailS[0]=headS[0]+DFI_Domian->GlobalVoxel[0]+outGc-1;
465   tailS[1]=headS[1]+DFI_Domian->GlobalVoxel[1]+outGc-1;
466
467   //成分数の取り出し
468   int nComp = dfi->GetNumComponent();
469
470   //配列形状の設定
471   CIO::E_CIO_ARRAYSHAPE out_shape;
472   if( nComp == 1 ) out_shape = CIO::E_CIO_IJKN;
473   else if( nComp > 1 ) out_shape = CIO::E_CIO_NIJK;
474
475   //セル中心出力のときガイドセル数を考慮してサイズ更新
476   if( !m_bgrid_interp_flag ) {
477     sz[0]=sz[0]+2*outGc;
478     sz[1]=sz[1]+2*outGc;
479   }
480
481   //出力バッファのインスタンス
482   cio_Array* src = cio_Array::instanceArray
483                    ( (CIO::E_CIO_DTYPE)d_type
484                    //, dfi->GetArrayShape()
485                    , out_shape
486                    , sz
487                    , interp_Gc
488                    , nComp );
489
```

```
490   //補間用バッファ，格子点出力バッファのインスタンス
491   cio_Array* src_old = NULL;
492   cio_Array* outArray = NULL;
493   if( m_bgrid_interp_flag ) {
494     src_old = cio_Array::instanceArray
495               ( (CIO::E_CIO_DTYPE)d_type
496               //, dfi->GetArrayShape()
497               , out_shape
498               , sz
499               , interp_Gc
500               , nComp );
501
502     int szOut[3];
503     for(int i=0; i<2; i++) szOut[i]=sz[i]+1;
504     szOut[2]=sz[2];
505     outArray = cio_Array::instanceArray
506               ( (CIO::E_CIO_DTYPE)d_type
507               //, dfi->GetArrayShape()
508               , out_shape
509               , szOut
510               , interp_Gc
511               , nComp );
512   }
513
514   int kdiv,jdiv,idiv;
515   int l_rank;
516   std::string infile;
517
518   //z 方向の分割数回のループ
519   for( headT::iterator itz=mapHeadZ.begin(); itz!= mapHeadZ.end(); itz++ ) {
520
521     //z 層のスタートエンドを設定
522     kdiv = itz->second;
523     int kp_sta,kp_end;
524     kp_sta = itz->first;
525     int nrank = _CIO_IDX_IJK(0,0,kdiv,div[0],div[1],div[2],0);
526     kp_end = kp_sta + DFI_Process->RankList[nrank].VoxelSize[2];
527
528     //z 層のスタートエンドをガイドセルの考慮
529     if( kdiv == 0 ) kp_sta = kp_sta-outGc;
530     if( kdiv == div[2]-1 ) kp_end = kp_end+outGc;
531
532     //同一 z 面のループ
533     for(int kp=kp_sta; kp< kp_end; kp++) {
534
535       int kk = kp-1;
536       //間引きの層のときスキップ
537       if( kk%thin_count != 0 ) continue;
538
539       //y 方向の分割数のループ
540       for( headT::iterator ity=mapHeadY.begin(); ity!= mapHeadY.end(); ity++ ) {
541
542         //y のスタートエンドの設定
543         jdiv = ity->second;
544         int jp_sta,jp_end;
545         jp_sta = ity->first;
546         int nrank = _CIO_IDX_IJK(0,jdiv,kdiv,div[0],div[1],div[2],0);
547         jp_end = jp_sta + DFI_Process->RankList[nrank].VoxelSize[1];
548
549         //x 方向の分割数のループ
550         for( headT::iterator itx=mapHeadX.begin(); itx!= mapHeadX.end(); itx++ ) {
551
552           //x のスタートエンドの設定
553           idiv = itx->second;
554           int ip_sta,ip_end;
555           ip_sta = itx->first;
556           int nrank = _CIO_IDX_IJK(idiv,jdiv,kdiv,div[0],div[1],div[2],0);
557           ip_end = ip_sta + DFI_Process->RankList[nrank].VoxelSize[0];
558
559           int RankID = _CIO_IDX_IJK(idiv,jdiv,kdiv,div[0],div[1],div[2],0);
560
561           //読込み範囲の設定
562           int read_sta[3],read_end[3];
563           read_sta[0]=ip_sta;
564           read_sta[1]=jp_sta;
565           read_sta[2]=kp;
566           read_end[0]=ip_end-1;
567           read_end[1]=jp_end-1;
568           read_end[2]=kp;
569
570           //ガイドセルを考慮して読込み範囲を更新
571           if( idiv == 0 ) read_sta[0] = read_sta[0]-outGc;
572           if( idiv == div[0]-1 ) read_end[0] = read_end[0]+outGc;
573           if( jdiv == 0 ) read_sta[1] = read_sta[1]-outGc;
574           if( jdiv == div[1]-1 ) read_end[1] = read_end[1]+outGc;
575
576           l_rank=DFI_Process->RankList[RankID].RankID;
```

```
577           //連結対象ファイル名の生成
578           infile = CIO::cioPath_ConnectPath(inPath,dfi->Generate_FieldFileName(l_rank,l_step,mio));
579           unsigned int avr_step;
580           double avr_time;
581           CIO::E_CIO_ERRORCODE ret;
582           //連結対象ファイルの読込み
583           cio_Array* buf = dfi->ReadFieldData(infile, l_step, l_dtime,
584                                               read_sta, read_end,
585                                               DFI_Process->RankList[RankID].HeadIndex,
586                                               DFI_Process->RankList[RankID].TailIndex,
587                                               true, avr_step, avr_time, ret);
588           //headIndex を0スタートにしてセット
589           int headB[3];
590           headB[0]=read_sta[0]-1;
591           headB[1]=read_sta[1]-1;
592           headB[2]=read_sta[2]-1;
593           buf->setHeadIndex( headB );
594
595           int headS0[3];
596           headS0[0]=headS[0];
597           headS0[1]=headS[1];
598           headS0[2]=kk/thin_count;
599           src->setHeadIndex( headS0 );
600
601           headS0[2]=kk;
602           tailS[2]=headS0[2];
603
604           //出力配列へのコンバイン
605           for(int n=0; n<nComp; n++) convertXY(buf,src,headS0,tailS,n);
606           delete buf;
607
608         }
609       }
610       //補間処理
611       if( m_bgrid_interp_flag ) {
612         if( kp == kp_sta ) {
613           for(int n=0; n<nComp; n++) {
614             if( !InterPolate(src,src,outArray,n,n) ) return false;
615           }
616         } else {
617           for(int n=0; n<nComp; n++) {
618             if( !InterPolate(src_old,src,outArray,n,n) ) return false;
619           }
620         }
621       } else outArray = src;
622
623       //一層分出力
624       if( outArray ) {
625         const int* szOutArray = outArray->getArraySizeInt();
626         size_t dLen = szOutArray[0]*szOutArray[1]*szOutArray[2]*outArray->getNcomp();
627         if( ConvOut->WriteFieldData(fp,
628                                     outArray,
629                                     dLen ) != true ) return false;
630       }
631
632       //minmax を求める
633       if( !DtypeMinMax(outArray,min,max) ) return false;
634
635       //補間ありのとき、読込んだ層の配列ポインタを src_old にコピー
636       if( m_bgrid_interp_flag ) {
637         cio_Array* tmp = src;
638         src = src_old;
639         src_old = tmp;
640       }
641     }
642   }
643
644   if( m_bgrid_interp_flag ) {
645     for(int n=0; n<nComp; n++) {
646       if( !InterPolate(src_old,src_old,outArray,n,n) ) return false;
647     }
648     if( outArray ) {
649       const int* szOutArray = outArray->getArraySizeInt();
650       size_t dLen = szOutArray[0]*szOutArray[1]*szOutArray[2]*outArray->getNcomp();
651       if( ConvOut->WriteFieldData(fp,
652                                   outArray,
653                                   dLen ) != true ) return false;
654
655        //minmax を求める
656        if( !DtypeMinMax(src,min,max) ) return false;
657
658     } else return false;
659   }
660   delete src;
661
662   if( m_bgrid_interp_flag ) {
663     delete src_old;
```

```
664     delete outArray;
665   }
666
667   return true;
668
669 }
```

4.2.4.3 template<class T > CONV_INLINE void convMx1::copyArray_nijk_ijk (cio_TypeArray< T > * *S*, cio_TypeArray< T > * *O*, int *ivar*)

convMx1_inline.h の 175 行で定義されています。

```
176 {
177   const int* sz = S->getArraySizeInt();
178   int gc = O->getGcInt();
179   if( S->getArrayShape() == CIO::E_CIO_NIJK ) {
180     for(int k=0-gc; k<sz[2]+gc; k++) {
181     for(int j=0-gc; j<sz[1]+gc; j++) {
182     for(int i=0-gc; i<sz[0]+gc; i++) {
183       O->val(i,j,k,0) = S->val(ivar,i,j,k);
184     }}}
185   }
186   else {
187     for(int k=0-gc; k<sz[2]+gc; k++) {
188     for(int j=0-gc; j<sz[1]+gc; j++) {
189     for(int i=0-gc; i<sz[0]+gc; i++) {
190       O->val(i,j,k,0) = S->val(i,j,k,ivar);
191     }}}
192   }
193 };
```

4.2.4.4 template<class T > void convMx1::copyArray_nijk_ijk (cio_TypeArray< T > * *S*, cio_TypeArray< T > * *O*, int *ivar*)

NIJK 配列をスカラーのIJK 配列にコピー

引数

in	*S*	コピー元配列
in	*O*	コピー先配列
in	*ivar*	コピーするコンポーネント位置

参照元 nijk_to_ijk().

4.2.4.5 bool convMx1::exec ( ) [virtual]

Mx1 の実行

戻り値

エラーコード

CONVを実装しています。

convMx1.C の 37 行で定義されています。

参照先 convMx1_out_ijkn(), convMx1_out_nijk(), ConvOut, CONV::dfi_MinMax::dfi, InputParam::Get_IndfiNameList(), InputParam::Get_OutdfiNameList(), InputParam::Get_OutprocNameList(), InputParam::Get_OutputArrayShape(), InputParam::Get_OutputDataType(), InputParam::Get_OutputFormat(), InputParam::Get_OutputGuideCell(), InputParam::Get_ThinOut(), convOutput::importInputParam(), LOG_OUT_, LOG_OUTV_, CONV::m_bgrid_interp_flag, CONV::m_fplog, CONV::m_in_dfi, CONV::m_InputCntl, CONV::m_myRank, m_StepRankList, CONV::makeProcInfo(), CONV::makeStepList(), CONV::MemoryRequirement(), convOutput::output_avs(), convOutput::OutputFile_Close(), convOutput::OutputFile_Open(), convOutput::OutputInit(), SPH_DOUBLE, SPH_FLOAT, STD_OUT_, STD_OUTV_, convOutput::WriteDataMarker(), convOutput::WriteGridData(), convOutput::WriteHeaderRecord(), CONV::WriteIndexDfiFile(), と CONV::WriteProcDfiFile().

```
38 {
39
40   // 出力ファイル形式クラスのインスタンス
41   ConvOut = convOutput::OutputInit(m_InputCntl->Get_OutputFormat());
42
43   // InputParam のインスタンス
44   if( !ConvOut->importInputParam(m_InputCntl) ) {
45     return false;
46   }
47
48   string prefix,outfile,infile,inPath;
49   FILE *fp;
50   int dummy;
51
52   int l_rank;
53   int l_d_type, d_type;
54   int l_step, l_imax, l_jmax, l_kmax;
55   float l_time;
56   double l_dorg[3], l_dpit[3];
57   double l_dtime;
58   int xsize,ysize,zsize,asize,vsize;
59   int dim;
60
61   int l_imax_th, l_jmax_th, l_kmax_th;
62
63   //間引き数のセット
64   int thin_count = m_InputCntl->Get_ThinOut();
65
66   // 出力モード
67   bool mio = false;
68   const cio_MPI* DFI_MPI = m_in_dfi[0]->GetcioMPI();
69   if( DFI_MPI->NumberOfRank > 1) mio=true;
70
71   //dfi_name の取得
72   vector<std::string> in_dfi_name = m_InputCntl->Get_IndfiNameList();
73
74   //step 基準のリスト生成
75   makeStepList(m_StepRankList);
76
77
78   //minmax の格納構造体のインスタンス
79   vector<dfi_MinMax*> minmaxList;
80
81   for(int i=0; i<m_in_dfi.size(); i++){
82     const cio_TimeSlice* TSlice = m_in_dfi[i]->GetcioTimeSlice();
83     int nComp = m_in_dfi[i]->GetNumComponent();
84
85     dfi_MinMax *MinMax;
86     if( nComp == 1 ) MinMax = new dfi_MinMax(TSlice->SliceList.size(),nComp);
87     else             MinMax = new dfi_MinMax(TSlice->SliceList.size(),nComp+1);
88
89     MinMax->dfi = m_in_dfi[i];
90     minmaxList.push_back(MinMax);
91   }
92
93   /*
94   LOG_OUTV_ {
95     fprintf(m_fplog,"\n");
96     for(int j=0; j< m_stepList.size(); j++ ) {
97       fprintf(m_fplog,"\tstepList[%4d] = %d\n",j,m_stepList[j].step);
98     }
99   }
100   STD_OUTV_ {
101     printf("\n");
102     for(int j=0; j< m_stepList.size(); j++ ) {
103       printf("\tstepList[%4d] = %d\n",j,m_stepList[j].step);
104     }
105   }
106   */
107
108   cio_Domain* DFI_Domian = (cio_Domain *)m_in_dfi[0]->GetcioDomain();
109   cio_Process* DFI_Process = (cio_Process *)m_in_dfi[0]->GetcioProcess();
110
111   int div[3];
112   div[0]=DFI_Domian->GlobalDivision[0];
113   div[1]=DFI_Domian->GlobalDivision[1];
114   div[2]=DFI_Domian->GlobalDivision[2];
115
116   //sph のオリジンとピッチを作成
117   l_dpit[0]=DFI_Domian->GlobalRegion[0]/(double)DFI_Domian->GlobalVoxel[0];
118   l_dpit[1]=DFI_Domian->GlobalRegion[1]/(double)DFI_Domian->GlobalVoxel[1];
119   l_dpit[2]=DFI_Domian->GlobalRegion[2]/(double)DFI_Domian->GlobalVoxel[2];
120   l_dorg[0]=DFI_Domian->GlobalOrigin[0]+0.5*l_dpit[0];
121   l_dorg[1]=DFI_Domian->GlobalOrigin[1]+0.5*l_dpit[1];
122   l_dorg[2]=DFI_Domian->GlobalOrigin[2]+0.5*l_dpit[2];
123
124   //全体サイズのキープ
```

```
125   l_imax= DFI_Domian->GlobalVoxel[0];
126   l_jmax= DFI_Domian->GlobalVoxel[1];
127   l_kmax= DFI_Domian->GlobalVoxel[2];
128
129   //間引きを考慮
130   l_imax_th=l_imax/thin_count;//間引き後の x サイズ
131   l_jmax_th=l_jmax/thin_count;//間引き後の y サイズ
132   l_kmax_th=l_kmax/thin_count;//間引き後の z サイズ
133   if(l_imax%thin_count != 0) l_imax_th++;
134   if(l_jmax%thin_count != 0) l_jmax_th++;
135   if(l_kmax%thin_count != 0) l_kmax_th++;
136
137   //GRID データ 出力
138   const cio_FileInfo* DFI_FInfo = m_in_dfi[0]->GetcioFileInfo();
139   int sz[3];
140   sz[0]=l_imax;
141   sz[1]=l_jmax;
142   sz[2]=l_kmax;
143   ConvOut->WriteGridData(0, m_myRank,
144                          DFI_FInfo->GuideCell, l_dorg, l_dpit, sz);
145
146
147   //mapHeadX,Y,Z の生成
148   //typedef std::map<int,int> headT;
149   headT mapHeadX;
150   headT mapHeadY;
151   headT mapHeadZ;
152   DFI_Process->CreateRankList(*DFI_Domian,
153                               mapHeadX,
154                               mapHeadY,
155                               mapHeadZ);
156
157   //dfi*step のループ
158   for (int i=0;i<m_StepRankList.size();i++) {
159
160     //dfi ファイルのディレクトリの取得
161     inPath = CIO::cioPath_DirName(m_StepRankList[i].dfi->get_dfi_fname());
162
163     const cio_FileInfo* DFI_FInfo = m_StepRankList[i].dfi->GetcioFileInfo();
164     prefix=DFI_FInfo->Prefix;
165     LOG_OUTV_ fprintf(m_fplog,"  COMBINE SPH START : %s\n", prefix.c_str());
166     STD_OUTV_ printf("  COMBINE SPH START : %s\n", prefix.c_str());
167
168     //Scalar or Vector
169     dim=m_StepRankList[i].dfi->GetNumComponent();
170
171     const cio_TimeSlice* TSlice = m_StepRankList[i].dfi->GetcioTimeSlice();
172
173     //step Loop
174     for(int j=m_StepRankList[i].stepStart; j<=m_StepRankList[i].stepEnd; j++) {
175
176       //minmax の初期化
177       int nsize = dim;
178       if( dim > 1 ) nsize++;
179       double *min = new double[nsize];
180       double *max = new double[nsize];
181       for(int n=0; n<nsize; n++) {
182          min[n]=DBL_MAX;
183          max[n]=-DBL_MAX;
184       }
185
186       l_step=TSlice->SliceList[j].step;
187       l_time=(float)TSlice->SliceList[j].time;
188
189       LOG_OUTV_ fprintf(m_fplog,"\tstep = %d\n", l_step);
190       STD_OUTV_ printf("\tstep = %d\n", l_step);
191
192       //連結出力ファイルオープン
193       fp = ConvOut->OutputFile_Open(prefix, l_step, 0, false);
194
195       //m_d_type のセット (float or double)
196       if( m_StepRankList[i].dfi->GetDataType() == CIO::E_CIO_FLOAT32 ) {
197         l_d_type = SPH_FLOAT;
198       } else if( m_StepRankList[i].dfi->GetDataType() == CIO::E_CIO_FLOAT64 ) {
199         l_d_type = SPH_DOUBLE;
200       } else return false;
201
202       if( m_InputCntl->Get_OutputDataType() == CIO::E_CIO_DTYPE_UNKNOWN )
203       {
204         d_type = m_StepRankList[i].dfi->GetDataType();
205       } else {
206         d_type = m_InputCntl->Get_OutputDataType();
207       }
208
209       //ヘッダーレコードを出力
210       double out_dpit[3];
211
```

```
212       int outGc=0;
213       if( m_InputCntl->Get_OutputGuideCell() > 1 ) outGc = m_InputCntl->
      Get_OutputGuideCell();
214       if( outGc > 0 ) {
215         if( outGc > DFI_FInfo->GuideCell ) outGc=DFI_FInfo->GuideCell;
216       }
217       if( thin_count > 1 || m_bgrid_interp_flag ) outGc-0;
218
219       for(int ic=0;ic<3;ic++) out_dpit[ic]=l_dpit[ic]*double(thin_count);
220       if( !(ConvOut->WriteHeaderRecord(l_step, dim, d_type,
221                                        l_imax_th+2*outGc, l_jmax_th+2*outGc, l_kmax_th+2*outGc,
222                                        l_time, l_dorg, out_dpit, prefix, fp)) ) {
223         printf("\twrite header error\n");
224         return false;
225       }
226       //全体の大きさの計算とデータのヘッダ書き込み
227       size_t dLen;
228
229       //dLen = size_t(l_imax_th) * size_t(l_jmax_th) * size_t(l_kmax_th);
230       dLen = size_t(l_imax_th+2*outGc) * size_t(l_jmax_th+2*outGc) * size_t(l_kmax_th+2*outGc);
231       if( dim == 3 ) dLen *= 3;
232       if( m_InputCntl->Get_OutputDataType() == CIO::E_CIO_FLOAT32 ) {
233          dummy = dLen * sizeof(float);
234       } else {
235          dummy = dLen * sizeof(double);
236       }
237       if( !(ConvOut->WriteDataMarker(dummy, fp)) ) {
238         printf("\twrite data header error\n");
239         return false;
240       }
241
242       //書き込み workarea のサイズ決め
243       xsize=l_imax_th;
244       ysize=l_jmax_th;
245       asize=xsize*ysize;
246       vsize=0;
247       for(int n=0; n< DFI_Process->RankList.size(); n++ ) {
248         int szx,szy,szz;
249         szx=DFI_Process->RankList[n].VoxelSize[0];
250         szy=DFI_Process->RankList[n].VoxelSize[1];
251         szz=DFI_Process->RankList[n].VoxelSize[2];
252         int vdum=szx*szy*szz;
253         if(vsize < vdum) vsize=vdum;
254       }
255       // メモリチェック
256       LOG_OUTV_ fprintf(m_fplog,"\tNode %4d - Node %4d\n", 0,
257                         DFI_Domian->GlobalVoxel[2] -1);
258       STD_OUTV_ printf("\tNode %4d - Node %4d\n", 0,
259                        DFI_Domian->GlobalVoxel[2] -1);
260       double mc1 = (double)asize*(double)dim;
261       double mc2 = (double)vsize*(double)dim;
262       if(mc1>(double)INT_MAX){// 整数値あふれ出しチェック //参考 894*894*894*3=2143550952 INT_MAX 2147483647
263         printf("\tsize error : mc1>INT_MAX\n");
264         return false;
265       }
266       if(mc2>(double)INT_MAX){// 整数値あふれ出しチェック //参考 894*894*894*3=2143550952 INT_MAX 2147483647
267         printf("\tsize error : mc2>INT_MAX\n");
268         return false;
269       }
270       double TotalMemory=0.0; // = mc * (double)sizeof(REAL_TYPE);
271       if( m_InputCntl->Get_OutputDataType() == CIO::E_CIO_FLOAT32 ) {
272         TotalMemory = TotalMemory + mc1 * (double)sizeof(float);
273       } else {
274         TotalMemory = TotalMemory + mc1 * (double)sizeof(double);
275       }
276
277       if( l_d_type == SPH_FLOAT ) {
278         TotalMemory = TotalMemory + mc2 * (double)sizeof(float);
279       } else {
280         TotalMemory = TotalMemory + mc2 * (double)sizeof(double);
281       }
282       LOG_OUT_ MemoryRequirement(TotalMemory,m_fplog);
283       STD_OUT_ MemoryRequirement(TotalMemory,stdout);
284
285       int szS[3];
286       szS[0]=l_imax_th;
287       szS[1]=l_jmax_th;
288       szS[2]=1;
289
290       CIO::E_CIO_ARRAYSHAPE output_AShape = (CIO::E_CIO_ARRAYSHAPE)m_InputCntl->
      Get_OutputArrayShape();
291
292       if( output_AShape == CIO::E_CIO_NIJK ||
293           DFI_FInfo->Component == 1 ){
294
295         //output nijk
296         if( !convMx1_out_nijk(fp,
```

```
297                             inPath,
298                             l_step,
299                             l_dtime,
300                             d_type,
301                             mio,
302                             div,
303                             szS,
304                             //m_stepList[i].dfi,
305                             m_StepRankList[i].dfi,
306                             DFI_Process,
307                             mapHeadX,mapHeadY,mapHeadZ,
308                             min,max
309                             ) ) return false;
310 /*
311         //dfi ごとに minmax を登録
312         for(int ndfi = 0; ndfi<minmaxList.size(); ndfi++) {
313           if( minmaxList[ndfi]->dfi != m_StepRankList[i].dfi ) continue;
314           for(int n=0; n<nsize; n++) {
315             if( minmaxList[ndfi]->Min[j*nsize+n] > min[n] ) minmaxList[ndfi]->Min[j*nsize+n] = min[n];
316             if( minmaxList[ndfi]->Max[j*nsize+n] < max[n] ) minmaxList[ndfi]->Max[j*nsize+n] = max[n];
317           }
318         }
319 */
320
321       } else {
322         //output IJKN
323         if( !convMx1_out_ijkn(fp,
324                             inPath,
325                             l_step,
326                             l_dtime,
327                             d_type,
328                             mio,
329                             div,
330                             szS,
331                             //m_stepList[i].dfi,
332                             m_StepRankList[i].dfi,
333                             DFI_Process,
334                             mapHeadX,mapHeadY,mapHeadZ,
335                             min,max
336                             ) ) return false;
337
338       }
339
340       //dfi ごとに minmax を登録
341       for(int ndfi = 0; ndfi<minmaxList.size(); ndfi++) {
342         if( minmaxList[ndfi]->dfi != m_StepRankList[i].dfi ) continue;
343         for(int n=0; n<nsize; n++) {
344           if( minmaxList[ndfi]->Min[j*nsize+n] > min[n] ) minmaxList[ndfi]->Min[j*nsize+n] = min[n];
345           if( minmaxList[ndfi]->Max[j*nsize+n] < max[n] ) minmaxList[ndfi]->Max[j*nsize+n] = max[n];
346         }
347       }
348
349       //データのフッタ書き込み
350       if( !(ConvOut->WriteDataMarker(dummy, fp)) ) {
351         printf("\twrite data error\n");
352         return false;
353       }
354
355       //出力ファイルクローズ
356       ConvOut->OutputFile_Close(fp);
357
358     }
359   }
360
361   //avs のヘッダーファイル出力
362   ConvOut->output_avs(m_myRank, m_in_dfi);
363
364   //出力 dfi ファイル名の取得
365   vector<std::string> out_dfi_name = m_InputCntl->Get_OutdfiNameList();
366   vector<std::string> out_proc_name = m_InputCntl->Get_OutprocNameList();
367
368   //出力 dfi ファイルの出力
369   if( out_dfi_name.size() == 0 || out_proc_name.size() == 0 ) return true;
370
371   //ランク間で通信して MINMAX を求めてランク 0 に送信
372   for(int i=0; i<minmaxList.size(); i++) {
373    int nComp = minmaxList[i]->dfi->GetNumComponent();
374    const cio_TimeSlice* TSlice = minmaxList[i]->dfi->GetcioTimeSlice();
375    int nStep = TSlice->SliceList.size();
376
377    int n = nComp*nStep;
378    if( nComp > 1 ) n = (nComp+1)*nStep;
379
380     //min の通信
381    double *send1 =  minmaxList[i]->Min;
382    double *recv1 = new double[n];
383    MPI_Reduce(send1, recv1, n, MPI_DOUBLE, MPI_MIN, 0, MPI_COMM_WORLD);
```

```
384    minmaxList[i]->Min = recv1;
385
386    //max の通信
387    double *send2 = minmaxList[i]->Max;
388    double *recv2 = new double[n];
389    MPI_Reduce(send2, recv2, n, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
390    minmaxList[i]->Max = recv2;
391
392   }
393
394   if( m_myRank == 0 ) {
395
396     WriteIndexDfiFile(minmaxList);
397
398     for(int i=0; i<m_in_dfi.size(); i++) {
399       cio_Domain* out_domain = NULL;
400       cio_MPI* out_mpi = NULL;
401       cio_Process* out_process = NULL;
402       const cio_MPI* dfi_mpi = m_in_dfi[i]->GetcioMPI();
403       int numProc = dfi_mpi->NumberOfRank;
404
405       //Proc 情報の生成
406       makeProcInfo(m_in_dfi[i],out_domain,out_mpi,out_process,1);
407
408       //Proc ファイル出力
409       WriteProcDfiFile(out_proc_name[i],out_domain,out_mpi,out_process);
410      }
411   }
412
413   return true;
414
415 }
```

4.2.4.6 bool convMx1::InterPolate (cio_Array * *src_old*, cio_Array * *src*, cio_Array * *outArray*, int *ivar_src*, int *ivar_out*)

補間処理

引数

in	*src_old*	1 つ前の層
in	*src*	処理する層
out	*outArray*	足しこむ配列
in	*ivar_src*	図心データの コンポーネント位置
in	*ivar_out*	格子データの コンポーネント位置

convMx1.C の 913 行で定義されています。

参照先 setGridData_XY(), VolumeDataDivide8(), と zeroClearArray().

参照元 convMx1_out_ijkn(), と convMx1_out_nijk().

```
915 {
916
917   if( !src_old || !src || !outArray ) return false;
918   //if( !src_old || !src ) return NULL;
919
920   //データタイプの取得
921   //int nComp = src->getNcomp();
922   CIO::E_CIO_DTYPE dtype = src->getDataType();
923
924   //char
925   if( dtype == CIO::E_CIO_INT8 ) {
926     cio_TypeArray<char> *O = dynamic_cast<cio_TypeArray<char>*>(outArray);
927     cio_TypeArray<char> *S = dynamic_cast<cio_TypeArray<char>*>(src);
928     cio_TypeArray<char> *S_old = dynamic_cast<cio_TypeArray<char>*>(src_old);
929
930     //足しこみ領域のゼロクリア
931     zeroClearArray(O,ivar_out);
932     //src の足しこみ
933     setGridData_XY(O,S,    ivar_out,ivar_src);
934     //src_old の足しこみ
935     setGridData_XY(O,S_old,ivar_out,ivar_src);
936     //平均化（8で割る）
937     VolumeDataDivide8(O,ivar_out);
938   }
939   //short
940   else if( dtype == CIO::E_CIO_INT16 ) {
941     cio_TypeArray<short> *O = dynamic_cast<cio_TypeArray<short>*>(outArray);
```

```
942     cio_TypeArray<short> *S = dynamic_cast<cio_TypeArray<short>*>(src);
943     cio_TypeArray<short> *S_old = dynamic_cast<cio_TypeArray<short>*>(src_old);
944
945     //足しこみ領域のゼロクリア
946     zeroClearArray(O,ivar_out);
947     //src の足しこみ
948     setGridData_XY(O,S,    ivar_out,ivar_src);
949     //src_old の足しこみ
950     setGridData_XY(O,S_old,ivar_out,ivar_src);
951     //平均化（8で割る）
952     VolumeDataDivide8(O,ivar_out);
953   }
954   //int
955   else if( dtype == CIO::E_CIO_INT32 ) {
956     cio_TypeArray<int> *O = dynamic_cast<cio_TypeArray<int>*>(outArray);
957     cio_TypeArray<int> *S = dynamic_cast<cio_TypeArray<int>*>(src);
958     cio_TypeArray<int> *S_old = dynamic_cast<cio_TypeArray<int>*>(src_old);
959
960     //足しこみ領域のゼロクリア
961     zeroClearArray(O,ivar_out);
962     //src の足しこみ
963     setGridData_XY(O,S,    ivar_out,ivar_src);
964     //src_old の足しこみ
965     setGridData_XY(O,S_old,ivar_out,ivar_src);
966     //平均化（8で割る）
967     VolumeDataDivide8(O,ivar_out);
968   }
969   //float
970   else if( dtype == CIO::E_CIO_FLOAT32 ) {
971     cio_TypeArray<float> *O = dynamic_cast<cio_TypeArray<float>*>(outArray);
972     cio_TypeArray<float> *S = dynamic_cast<cio_TypeArray<float>*>(src);
973     cio_TypeArray<float> *S_old = dynamic_cast<cio_TypeArray<float>*>(src_old);
974
975     //足しこみ領域のゼロクリア
976     zeroClearArray(O,ivar_out);
977     //src の足しこみ
978     setGridData_XY(O,S,    ivar_out,ivar_src);
979     //src_old の足しこみ
980     setGridData_XY(O,S_old,ivar_out,ivar_src);
981
982     //平均化（8で割る）
983     VolumeDataDivide8(O,ivar_out);
984   }
985   //double
986   else if( dtype == CIO::E_CIO_FLOAT64 ) {
987     cio_TypeArray<double> *O = dynamic_cast<cio_TypeArray<double>*>(outArray);
988     cio_TypeArray<double> *S = dynamic_cast<cio_TypeArray<double>*>(src);
989     cio_TypeArray<double> *S_old = dynamic_cast<cio_TypeArray<double>*>(src_old);
990
991     //足しこみ領域のゼロクリア
992     zeroClearArray(O,ivar_out);
993     //src の足しこみ
994     setGridData_XY(O,S,    ivar_out,ivar_src);
995     //src_old の足しこみ
996     setGridData_XY(O,S_old,ivar_out,ivar_src);
997     //平均化（8で割る）
998     VolumeDataDivide8(O,ivar_out);
999   }
1000
1001    return outArray;
1002
1003 }
```

4.2.4.7 cio_Array * convMx1::nijk_to_ijk (cio_Array * *src*, int *ivar*)

NIJK 配列をスカラーのIJK 配列にコピーコントロール

引数

in	*src*	コピー元配列
in	*ivar*	コピーするコンポーネント位置

戻り値

IJK にコピーされて配列ポインタ

convMx1.C の 1008 行で定義されています。

参照先 copyArray_nijk_ijk().

参照元 convMx1_out_ijkn().

```
1009 {
1010
1011   //コピー元配列のサイズとデータタイプの取得
1012   const int *sz = src->getArraySizeInt();
1013   CIO::E_CIO_DTYPE d_type = src->getDataType();
1014
1015   cio_Array* outArray = cio_Array::instanceArray
1016                       ( d_type
1017                       , CIO::E_CIO_IJKN
1018                       , (int *)sz
1019                       , 0
1020                       , 1 );
1021   //unsigned char
1022   if( d_type == CIO::E_CIO_UINT8 ) {
1023     cio_TypeArray<unsigned char> *S = dynamic_cast<cio_TypeArray<unsigned char>*>(src);
1024     cio_TypeArray<unsigned char> *O = dynamic_cast<cio_TypeArray<unsigned char>*>(outArray);
1025     copyArray_nijk_ijk(S,O,ivar);
1026   }
1027   //char
1028   else if( d_type == CIO::E_CIO_INT8 ) {
1029     cio_TypeArray<char> *S = dynamic_cast<cio_TypeArray<char>*>(src);
1030     cio_TypeArray<char> *O = dynamic_cast<cio_TypeArray<char>*>(outArray);
1031     copyArray_nijk_ijk(S,O,ivar);
1032   }
1033   //unsigned short
1034   else if( d_type == CIO::E_CIO_UINT16 ) {
1035     cio_TypeArray<unsigned short> *S = dynamic_cast<cio_TypeArray<unsigned short>*>(src);
1036     cio_TypeArray<unsigned short> *O = dynamic_cast<cio_TypeArray<unsigned short>*>(outArray);
1037     copyArray_nijk_ijk(S,O,ivar);
1038   }
1039   //short
1040   else if( d_type == CIO::E_CIO_INT16 ) {
1041     cio_TypeArray<short> *S = dynamic_cast<cio_TypeArray<short>*>(src);
1042     cio_TypeArray<short> *O = dynamic_cast<cio_TypeArray<short>*>(outArray);
1043     copyArray_nijk_ijk(S,O,ivar);
1044   }
1045   //unsigned int
1046   else if( d_type == CIO::E_CIO_UINT32 ) {
1047     cio_TypeArray<unsigned int> *S = dynamic_cast<cio_TypeArray<unsigned int>*>(src);
1048     cio_TypeArray<unsigned int> *O = dynamic_cast<cio_TypeArray<unsigned int>*>(outArray);
1049     copyArray_nijk_ijk(S,O,ivar);
1050   }
1051   //int
1052   else if( d_type == CIO::E_CIO_INT32 ) {
1053     cio_TypeArray<int> *S = dynamic_cast<cio_TypeArray<int>*>(src);
1054     cio_TypeArray<int> *O = dynamic_cast<cio_TypeArray<int>*>(outArray);
1055     copyArray_nijk_ijk(S,O,ivar);
1056   }
1057   //unsigned long
1058   else if( d_type == CIO::E_CIO_UINT64 ) {
1059     cio_TypeArray<unsigned long> *S = dynamic_cast<cio_TypeArray<unsigned long>*>(src);
1060     cio_TypeArray<unsigned long> *O = dynamic_cast<cio_TypeArray<unsigned long>*>(outArray);
1061     copyArray_nijk_ijk(S,O,ivar);
1062   }
1063   //long
1064   else if( d_type == CIO::E_CIO_INT64 ) {
1065     cio_TypeArray<long> *S = dynamic_cast<cio_TypeArray<long>*>(src);
1066     cio_TypeArray<long> *O = dynamic_cast<cio_TypeArray<long>*>(outArray);
1067     copyArray_nijk_ijk(S,O,ivar);
1068   }
1069   //float
1070   else if( d_type == CIO::E_CIO_FLOAT32 ) {
1071     cio_TypeArray<float> *S = dynamic_cast<cio_TypeArray<float>*>(src);
1072     cio_TypeArray<float> *O = dynamic_cast<cio_TypeArray<float>*>(outArray);
1073     copyArray_nijk_ijk(S,O,ivar);
1074   }
1075   //double
1076   else if( d_type == CIO::E_CIO_FLOAT64 ) {
1077     cio_TypeArray<double> *S = dynamic_cast<cio_TypeArray<double>*>(src);
1078     cio_TypeArray<double> *O = dynamic_cast<cio_TypeArray<double>*>(outArray);
1079     copyArray_nijk_ijk(S,O,ivar);
1080   }
1081
1082   return outArray;
1083 }
```

4.2.4.8 template<class T > CONV_INLINE bool convMx1::setGridData_XY (cio_TypeArray< T > * *O*, cio_TypeArray< T > * *S*, int *ivar_out*, int *ivar_src*)

Scalar の格子点での値をセット

引数

out	*O*	格子点 data
in	*S*	セル中心 data
in	*ivar_out*	格子データの コンポーネント位置
in	*ivar_src*	図心データの コンポーネント位置

ガイドセルがない場合は処理しない

convMx1_inline.h の 64 行で定義されています。

```
69 {
70   if( O->getArrayShape() != S->getArrayShape() ) return false;
71
72   //ガイドセル数の取得
73   int gc = S->getGcInt();
74   if( gc < 1 ) return false;
75
76   //S(図心）の配列サイズをセット
77   const int* size = S->getArraySizeInt();
78   int ix = size[0];
79   int jx = size[1];
80   int kx = size[2];
81
82   //仮想セル領域へのコピー
83   if( S->getArrayShape() == CIO::E_CIO_NIJK ) {
84     for(int j=0; j<jx; j++) {
85       S->val(ivar_src,-1,j,0) = S->val(ivar_src,0,j,0);
86       S->val(ivar_src,ix,j,0) = S->val(ivar_src,ix-1,j,0);
87     }
88     for(int i=-1; i<ix+1; i++) {
89       S->val(ivar_src,i,-1,0) = S->val(ivar_src,i,0,0);
90       S->val(ivar_src,i,jx,0) = S->val(ivar_src,i,jx-1,0);
91     }
92   } else {
93     for(int j=0; j<jx; j++) {
94       S->val(-1,j,0,ivar_src) = S->val(0,j,0,ivar_src);
95       S->val(ix,j,0,ivar_src) = S->val(ix-1,j,0,ivar_src);
96     }
97     for(int i=-1; i<ix+1; i++) {
98       S->val(i,-1,0,ivar_src) = S->val(i,0,0,ivar_src);
99       S->val(i,jx,0,ivar_src) = S->val(i,jx-1,0,ivar_src);
100     }
101   }
102
103   //O（格子点）の配列サイズをセット
104   const int *Osz = O->getArraySizeInt();
105   int id = Osz[0];
106   int jd = Osz[1];
107   int kd = Osz[2];
108
109   //図心データを格子点に加える
110   if( O->getArrayShape() == CIO::E_CIO_NIJK ) {
111     for (int km=0; km<kx; km++) {
112     for (int jm=0-gc; jm<jx+gc; jm++) {
113     for (int im=0-gc; im<ix+gc; im++) {
114       O->val(ivar_out, im  ,jm  ,km) = O->val(ivar_out, im  ,jm  ,km)+S->val(ivar_src,im,jm,km);
115       O->val(ivar_out, im+1,jm  ,km) = O->val(ivar_out, im+1,jm  ,km)+S->val(ivar_src,im,jm,km);
116       O->val(ivar_out, im  ,jm+1,km) = O->val(ivar_out, im  ,jm+1,km)+S->val(ivar_src,im,jm,km);
117       O->val(ivar_out, im+1,jm+1,km) = O->val(ivar_out, im+1,jm+1,km)+S->val(ivar_src,im,jm,km);
118     }}}
119   } else {
120     for (int km=0; km<kx; km++) {
121     for (int jm=0-gc; jm<jx+gc; jm++) {
122     for (int im=0-gc; im<ix+gc; im++) {
123       O->val(im  ,jm  ,km ,ivar_out) = O->val(im  ,jm  ,km,ivar_out)+S->val(im,jm,km,ivar_src);
124       O->val(im+1,jm  ,km ,ivar_out) = O->val(im+1,jm  ,km,ivar_out)+S->val(im,jm,km,ivar_src);
125       O->val(im  ,jm+1,km ,ivar_out) = O->val(im  ,jm+1,km,ivar_out)+S->val(im,jm,km,ivar_src);
126       O->val(im+1,jm+1,km ,ivar_out) = O->val(im+1,jm+1,km,ivar_out)+S->val(im,jm,km,ivar_src);
127     }}}
128   }
129   return true;
130 };
```

4.2.4.9 template<class T > bool convMx1::setGridData_XY (cio_TypeArray< T > * *O*, cio_TypeArray< T > * *S*, int *ivar_out*, int *ivar_src*)

図心データを格子点に補間

引数

out	*O*	格子点データ
in	*S*	図心データ
in	*ivar_out*	格子データの コンポーネント位置
in	*ivar_src*	図心データの コンポーネント位置

参照元 InterPolate().

4.2.4.10 template<class T > CONV_INLINE void convMx1::VolumeDataDivide8 (cio_TypeArray< T > * *O*, int *n*)

convMx1_inline.h の 137 行で定義されています。

```
138 {
139   const int* szO = O->getArraySizeInt();
140   int id = szO[0];
141   int jd = szO[1];
142   int kd = szO[2];
143
144
145   //NIJK
146   if( O->getArrayShape() == CIO::E_CIO_NIJK ) {
147     //I
148     for(int k=0; k<kd; k++) {
149     for(int j=0; j<jd; j++) {
150     for(int i=0; i<id; i++) {
151     //for(int n=0; n<nComp; n++) {
152       O->val(n,i,j,k) = O->val(n,i,j,k)*0.125;
153     //}}}}
154     }}}
155
156   //IJKN
157   } else {
158     //I
159     //for (int n=0; n<nComp; n++){
160     for (int k=0; k<kd; k++){
161     for (int j=0; j<jd; j++){
162     for (int i=0; i<id; i++){
163       O->val(i,j,k,n) = O->val(i,j,k,n)*0.125;
164     //}}}}
165     }}}
166   }
167
168 };
```

4.2.4.11 template<class T > void convMx1::VolumeDataDivide8 (cio_TypeArray< T > * *O*, int *ivar_out*)

内部の格子点のデータを重み付けでで割る

引数

out	*O*	格子点データ
in	*ivar_out*	コンポーネント位置

参照元 InterPolate().

4.2.4.12 template<class T > CONV_INLINE void convMx1::zeroClearArray (cio_TypeArray< T > * *data*, int *ivar_out*)

配列のゼロクリア

引数

out	*data*	配列

in	*ivar_out*	コンポーネント位置

convMx1_inline.h の 34 行で定義されています。

```
35 {
36
37   const int *sz = data->getArraySizeInt();
38   CIO::E_CIO_ARRAYSHAPE shape = data->getArrayShape();
39
40   if( shape == CIO::E_CIO_NIJK ) {
41     for(int k=0; k<sz[2]; k++) {
42     for(int j=0; j<sz[1]; j++) {
43     for(int i=0; i<sz[0]; i++) {
44       data->val(ivar_out,i,j,k) = (T)0.0;
45     }}}
46   } else {
47     for(int k=0; k<sz[2]; k++) {
48     for(int j=0; j<sz[1]; j++) {
49     for(int i=0; i<sz[0]; i++) {
50       data->val(i,j,k,ivar_out) = (T)0.0;
51     }}}
52   }
53 };
```

4.2.4.13 template<class T > void convMx1::zeroClearArray (cio_TypeArray< T > * *data*, int *ivar_out*)

配列のゼロクリア

引数

out	*data*	配列
in	*ivar_out*	コンポーネント位置

参照元 InterPolate().

4.2.5 変数

4.2.5.1 convOutput* convMx1::ConvOut

convMx1.h の 28 行で定義されています。

参照元 convMx1_out_ijkn(), convMx1_out_nijk(), と exec().

4.2.5.2 vector<step_rank_info> convMx1::m_StepRankList

並列処理用インデックスリスト

convMx1.h の 33 行で定義されています。

参照元 convMx1(), と exec().

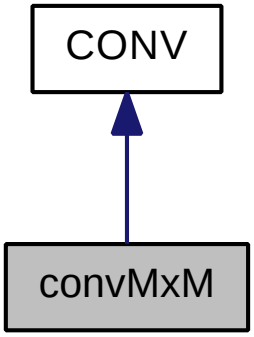
このクラスの説明は次のファイルから生成されました:

- convMx1.h
- convMx1.C
- convMx1_inline.h

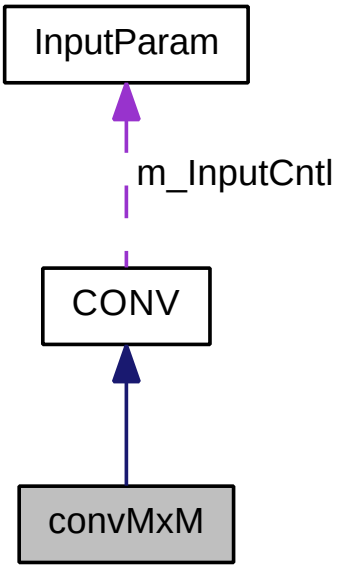
4.3 クラス convMxM

```
#include <convMxM.h>
```

convMxM に対する継承グラフ



convMxM のコラボレーション図



Public メソッド

- convMxM ()
- ~convMxM ()
- bool exec ()

  MxM **の実行**
- bool mxmsolv (std::string dfiname, cio_DFI *dfi, int l_step, double l_time, int rankID, double *min, double *max)

Public 変数

- vector< step_rank_info > m_StepRankList

  並列処理用インデックスリスト

Additional Inherited Members

4.3.1 説明

convMxM.h の 24 行で定義されています。

4.3.2 コンストラクタとデストラクタ

4.3.2.1 convMxM::convMxM ( )

コンストラクタ

convMxM.C の 21 行で定義されています。

参照先 m_StepRankList.

```
22 {
23
24   //m_stepList.clear();
25   //m_rankList.clear();
26   m_StepRankList.clear();
27
28 }
```

4.3.2.2 convMxM::～convMxM ( )

デストラクタ

convMxM.C の 32 行で定義されています。

```
33 {
34
35 }
```

4.3.3 関数

4.3.3.1 bool convMxM::exec ( ) [virtual]

MxM の実行

戻り値

エラーコード

CONVを実装しています。

convMxM.C の 39 行で定義されています。

参照先 CONV::dfi_MinMax::dfi, E_OUTPUT_CAST_UNKNOWN, E_OUTPUT_RANK, E_OUTPUT_STEP, InputParam::Get_MultiFileCasting(), InputParam::Get_OutdfiNameList(), InputParam::Get_OutprocNameList(), CONV::m_in_dfi, CONV::m_InputCntl, CONV::m_myRank, m_StepRankList, CONV::makeProcInfo(), CONV::makeRankList(), CONV::makeStepList(), mxmsolv(), CONV::WriteIndexDfiFile(), と CONV::WriteProcDfiFile().

```
40 {
41
42
43   //並列実行時のファイル割振り方法の取得
44   int outlist = m_InputCntl->Get_MultiFileCasting();
45
46   //並列実行時のファイル割振り方法でのリスト生成
47   if( outlist == E_OUTPUT_STEP || outlist == E_OUTPUT_CAST_UNKNOWN ) {
48     makeStepList(m_StepRankList);
49   } else if( outlist == E_OUTPUT_RANK ) {
50     makeRankList(m_StepRankList);
51   }
52
53   //出力 dfi ファイル名の取得
```

```
54   vector<std::string> out_dfi_name = m_InputCntl->Get_OutdfiNameList();
55   vector<std::string> out_proc_name = m_InputCntl->Get_OutprocNameList();
56
57   //minmax の格納構造体のインスタンス
58   vector<dfi_MinMax*> minmaxList;
59
60   for(int i=0; i<m_in_dfi.size(); i++){
61     const cio_TimeSlice* TSlice = m_in_dfi[i]->GetcioTimeSlice();
62     int nComp = m_in_dfi[i]->GetNumComponent();
63
64     dfi_MinMax *MinMax;
65     if( nComp == 1 ) MinMax = new dfi_MinMax(TSlice->SliceList.size(),nComp);
66     else             MinMax = new dfi_MinMax(TSlice->SliceList.size(),nComp+1);
67
68     MinMax->dfi = m_in_dfi[i];
69     minmaxList.push_back(MinMax);
70   }
71
72   //List のループ
73   for (int i=0; i<m_StepRankList.size(); i++) {
74
75     //dfi の step リストの取得
76     const cio_TimeSlice* TSlice = m_StepRankList[i].dfi->GetcioTimeSlice();
77
78     //成分数の取得
79     int nComp = m_StepRankList[i].dfi->GetNumComponent();
80
81     //step のループ
82     for(int j=m_StepRankList[i].stepStart; j<=m_StepRankList[i].stepEnd; j++) {
83
84       //minmax の初期化
85       int nsize = nComp;
86       if( nComp > 1 ) nsize++;
87       double *min = new double[nsize];
88       double *max = new double[nsize];
89       for(int n=0; n<nsize; n++) {
90          min[n]=DBL_MAX;
91          max[n]=-DBL_MAX;
92       }
93
94       //rank のループ
95       for(int k=m_StepRankList[i].rankStart; k<=m_StepRankList[i].rankEnd; k++) {
96         //MxM の読込みコンバート出力
97         if( !mxmsolv(m_StepRankList[i].dfi->get_dfi_fname(),
98                      m_StepRankList[i].dfi,
99                      TSlice->SliceList[j].step,
100                      (float)TSlice->SliceList[j].time,
101                      k,
102                      min,
103                      max) ) return false;
104       }
105
106       //dfi ごとに登録
107       for(int ndfi = 0; ndfi<minmaxList.size(); ndfi++) {
108         if( minmaxList[ndfi]->dfi != m_StepRankList[i].dfi ) continue;
109         for(int n=0; n<nsize; n++) {
110           if( minmaxList[ndfi]->Min[j*nsize+n] > min[n] ) minmaxList[ndfi]->Min[j*nsize+n] = min[n];
111           if( minmaxList[ndfi]->Max[j*nsize+n] < max[n] ) minmaxList[ndfi]->Max[j*nsize+n] = max[n];
112         }
113       }
114     }
115   }
116
117
118   //出力 dfi ファイルがないときは return
119   if( out_dfi_name.size() < 1 || out_proc_name.size() < 1 ) return true;
120
121   //ランク間で通信して MINMAX を求めてランク０に送信
122   for(int i=0; i<minmaxList.size(); i++) {
123     int nComp = minmaxList[i]->dfi->GetNumComponent();
124     const cio_TimeSlice* TSlice = minmaxList[i]->dfi->GetcioTimeSlice();
125     int nStep = TSlice->SliceList.size();
126
127     int n = nComp*nStep;
128     if( nComp > 1 ) n = (nComp+1)*nStep;
129
130     //min の通信
131     double *send1 =  minmaxList[i]->Min;
132     double *recv1 = new double[n];
133     MPI_Reduce(send1, recv1, n, MPI_DOUBLE, MPI_MIN, 0, MPI_COMM_WORLD);
134     minmaxList[i]->Min = recv1;
135
136     //max の通信
137     double *send2 = minmaxList[i]->Max;
138     double *recv2 = new double[n];
139     MPI_Reduce(send2, recv2, n, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
140     minmaxList[i]->Max = recv2;
```

```
141
142   }
143
144   //出力 dfi ファイルの出力
145   if( m_myRank == 0 ) {
146     WriteIndexDfiFile(minmaxList);
147
148     for(int i=0; i<m_in_dfi.size(); i++) {
149       cio_Domain* out_domain = NULL;
150       cio_MPI* out_mpi = NULL;
151       cio_Process* out_process = NULL;
152       const cio_MPI* dfi_mpi = m_in_dfi[i]->GetcioMPI();
153       int numProc = dfi_mpi->NumberOfRank;
154
155       //Proc 情報の生成
156       makeProcInfo(m_in_dfi[i],out_domain,out_mpi,out_process,numProc);
157
158       //Proc ファイル出力
159       WriteProcDfiFile(out_proc_name[i],out_domain,out_mpi,out_process);
160     }
161
162   }
163
164   return true;
165
166 }
```

4.3.3.2 bool convMxM::mxmsolv (std::string *dfiname*, cio_DFI * *dfi*, int *l_step*, double *l_time*, int *rankID*, double * *min*, double * *max*)

convMxM.C の 170 行で定義されています。

参照先 CONV::convertXY(), CONV::DtypeMinMax(), InputParam::Get_OutputArrayShape(), InputParam::Get_OutputDataType(), InputParam::Get_OutputDir(), InputParam::Get_OutputFilenameFormat(), InputParam::Get_OutputFormat(), InputParam::Get_OutputFormatType(), InputParam::Get_OutputGuideCell(), InputParam::Get_ThinOut(), CONV::m_bgrid_interp_flag, CONV::m_HostName, と CONV::m_InputCntl.

参照元 exec().

```
177 {
178
179   const cio_Process* DFI_Process = dfi->GetcioProcess();
180   cio_Domain* DFI_Domain = (cio_Domain *)dfi->GetcioDomain();
181   const cio_MPI* DFI_MPI = dfi->GetcioMPI();
182   const cio_FileInfo* DFI_FInfo = dfi->GetcioFileInfo();
183
184   bool mio = false;
185   if( DFI_MPI->NumberOfRank > 1) mio=true;
186
187   //間引き数のセット
188   int thin_count = m_InputCntl->Get_ThinOut();
189
190   //出力ガイドセルの設定
191   int outGc=0;
192   if( m_InputCntl->Get_OutputGuideCell() > 1 ) outGc = m_InputCntl->
      Get_OutputGuideCell();
193   if( outGc > 0 ) {
194     const cio_FileInfo* DFI_FInfo = dfi->GetcioFileInfo();
195     if( outGc > DFI_FInfo->GuideCell ) outGc=DFI_FInfo->GuideCell;
196   }
197   //間引きありのとき、出力ガイドセルを 0 に設定
198   if( thin_count > 1 ) outGc=0;
199   //格子点出力のときガイドセルを 0 に設定
200   if( m_bgrid_interp_flag ) outGc=0;
201
202   //ピッチのセット
203   double l_dpit[3];
204   l_dpit[0]=DFI_Domain->GlobalRegion[0]/(double)DFI_Domain->GlobalVoxel[0];
205   l_dpit[1]=DFI_Domain->GlobalRegion[1]/(double)DFI_Domain->GlobalVoxel[1];
206   l_dpit[2]=DFI_Domain->GlobalRegion[2]/(double)DFI_Domain->GlobalVoxel[2];
207   double out_dpit[3];
208   for (int i=0;i<3;i++) out_dpit[i]=l_dpit[i]*double(thin_count);
209
210   //全体のボクセルサイズを間引きを考慮して求める
211   int voxel[3];
212   for(int i=0; i<3; i++) {
213     voxel[i]=DFI_Domain->GlobalVoxel[i]/thin_count;
214     if( DFI_Domain->GlobalVoxel[i]%thin_count != 0 ) voxel[i]++;
215   }
216
```

```
217   //間引きを考慮したサイズのセット
218   int l_imax_th = DFI_Process->RankList[RankID].VoxelSize[0]/thin_count;
219   int l_jmax_th = DFI_Process->RankList[RankID].VoxelSize[1]/thin_count;
220   int l_kmax_th = DFI_Process->RankList[RankID].VoxelSize[2]/thin_count;
221
222   //間引き後のサイズが 1 つも無い領域のときエラー
223   if( l_imax_th < 1 || l_jmax_th < 1 || l_kmax_th < 1 ) {
224     printf("\toutput domain size error\n");
225     return false;
226   }
227
228   if(DFI_Process->RankList[RankID].VoxelSize[0]%thin_count != 0) l_imax_th++;
229   if(DFI_Process->RankList[RankID].VoxelSize[1]%thin_count != 0) l_jmax_th++;
230   if(DFI_Process->RankList[RankID].VoxelSize[2]%thin_count != 0) l_kmax_th++;
231
232   //間引き後の head インデックスを求める
233   int head[3];
234   head[0] = (DFI_Process->RankList[RankID].HeadIndex[0]-1)/thin_count;
235   head[1] = (DFI_Process->RankList[RankID].HeadIndex[1]-1)/thin_count;
236   head[2] = (DFI_Process->RankList[RankID].HeadIndex[2]-1)/thin_count;
237   if( (DFI_Process->RankList[RankID].HeadIndex[0]-1)%thin_count != 0 ) head[0]++;
238   if( (DFI_Process->RankList[RankID].HeadIndex[1]-1)%thin_count != 0 ) head[1]++;
239   if( (DFI_Process->RankList[RankID].HeadIndex[2]-1)%thin_count != 0 ) head[2]++;
240
241   //間引き後のオリジンを求める
242   double l_dorg[3];
243   l_dorg[0]= DFI_Domain->GlobalOrigin[0]+0.5*out_dpit[0]+head[0]*out_dpit[0];
244   l_dorg[1]= DFI_Domain->GlobalOrigin[1]+0.5*out_dpit[1]+head[1]*out_dpit[1];
245   l_dorg[2]= DFI_Domain->GlobalOrigin[2]+0.5*out_dpit[2]+head[2]*out_dpit[2];
246
247   //出力タイプのセット
248   int d_type;
249   if( m_InputCntl->Get_OutputDataType() == CIO::E_CIO_DTYPE_UNKNOWN )
250   {
251     d_type = dfi->GetDataType();
252   } else {
253     d_type = m_InputCntl->Get_OutputDataType();
254   }
255
256   //出力バッファのインスタンス
257   int szS[3];
258   szS[0]=l_imax_th;
259   szS[1]=l_jmax_th;
260   szS[2]=l_kmax_th;
261   cio_Array* src = cio_Array::instanceArray
262                  ( (CIO::E_CIO_DTYPE)d_type
263                  //, dfi->GetArrayShape()
264                  , (CIO::E_CIO_ARRAYSHAPE)m_InputCntl->Get_OutputArrayShape()
265                  , szS
266                  //, 0
267                  , outGc
268                  , dfi->GetNumComponent() );
269
270   //読込みファイル名の生成
271   std::string inPath = CIO::cioPath_DirName(dfiname);
272   std::string infile =  CIO::cioPath_ConnectPath(inPath,dfi->Generate_FieldFileName(
273                         RankID,l_step,mio));
274
275   //ファイルの読込み
276   unsigned int avr_step;
277   double l_dtime, avr_time;
278   CIO::E_CIO_ERRORCODE ret;
279   int read_sta[3],read_end[3];
280   for(int i=0; i<3; i++) {
281     read_sta[i]=DFI_Process->RankList[RankID].HeadIndex[i]-outGc;
282     read_end[i]=DFI_Process->RankList[RankID].TailIndex[i]+outGc;
283   }
284
285   cio_Array* buf = dfi->ReadFieldData(infile, l_step, l_dtime,
286                                       //DFI_Process->RankList[RankID].HeadIndex,
287                                       //DFI_Process->RankList[RankID].TailIndex,
288                                       read_sta,
289                                       read_end,
290                                       DFI_Process->RankList[RankID].HeadIndex,
291                                       DFI_Process->RankList[RankID].TailIndex,
292                                       true, avr_step, avr_time, ret);
293   if( ret != CIO::E_CIO_SUCCESS ) return false;
294
295   //間引き及び型変換がない場合
296   if( thin_count == 1 && buf->getDataType() == src->getDataType() &&
297       buf->getArrayShape() == src->getArrayShape() ) {
298     src = buf;
299   } else {
300   //間引きまたは型変換がある場合
301     int headS[3],tailS[3];
302     for(int i=0; i<3; i++) {
303       headS[i]=DFI_Process->RankList[RankID].HeadIndex[i]-1-outGc;
```

```
304       tailS[i]=DFI_Process->RankList[RankID].TailIndex[i]-1+outGc;
305     }
306     buf->setHeadIndex( headS );
307     src->setHeadIndex( head );
308     for(int n=0; n<dfi->GetNumComponent(); n++) convertXY(buf,src,headS,tailS,n);
309     //delete buf;
310   }
311
312   //出力 DFI のインスタンス
313   int tail[3];
314   head[0]=head[0]+1;
315   head[1]=head[1]+1;
316   head[2]=head[2]+1;
317   tail[0]=head[0]+l_imax_th-1;
318   tail[1]=head[1]+l_jmax_th-1;
319   tail[2]=head[2]+l_kmax_th-1;
320   cio_DFI* out_dfi = cio_DFI::WriteInit(
321                      MPI_COMM_WORLD,
322                      "",
323                      m_InputCntl->Get_OutputDir(),
324                      DFI_FInfo->Prefix,
325                      (CIO::E_CIO_FORMAT)m_InputCntl->Get_OutputFormat(),
326                      //0,
327                      outGc,
328                      //DFI_FInfo->DataType,
329                      (CIO::E_CIO_DTYPE)d_type,
330                      //DFI_FInfo->ArrayShape,
331                      (CIO::E_CIO_ARRAYSHAPE)m_InputCntl->Get_OutputArrayShape(),
332                      DFI_FInfo->Component,
333                      "",
334                      voxel,
335                      out_dpit,
336                      l_dorg,
337                      DFI_Domain->GlobalDivision,
338                      head,
339                      tail,
340                      m_HostName,
341                      CIO::E_CIO_OFF);
342   if( out_dfi == NULL ) {
343     printf("\tFails to instance dfi\n");
344     return false;
345   }
346
347   out_dfi->set_RankID(RankID);
348   out_dfi->SetcioMPI(*DFI_MPI);
349   out_dfi->SetcioProcess(*DFI_Process);
350
351   //出力
352   out_dfi->set_output_type((CIO::E_CIO_OUTPUT_TYPE)m_InputCntl->
      Get_OutputFormatType());
353   int output_fname = m_InputCntl->Get_OutputFilenameFormat();
354   out_dfi->set_output_fname((CIO::E_CIO_OUTPUT_FNAME)output_fname);
355   double tmp_minmax[8];
356   unsigned idummy=0;
357   double ddummy=0.0;
358   ret = out_dfi->WriteData(
359                            (unsigned)l_step,
360                            //0,
361                            outGc,
362                            l_time,
363                            src,
364                            tmp_minmax,
365                            true,
366                            idummy,
367                            ddummy);
368
369   if( ret != CIO::E_CIO_SUCCESS ) return false;
370
371   //minmax を求める
372   if( !DtypeMinMax(src,min,max) ) return false;
373
374   //delete
375   delete out_dfi;
376   delete src;
377
378   return true;
379 }
```

4.3.4 変数

4.3.4.1 vector<step_rank_info> convMxM::m_StepRankList

並列処理用インデックスリスト

convMxM.h の 30 行で定義されています。

参照元 convMxM(), と exec().

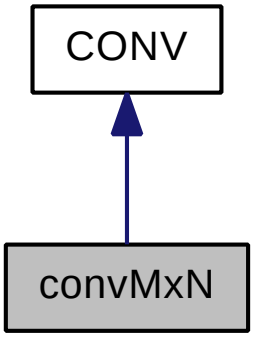
このクラスの説明は次のファイルから生成されました:

- convMxM.h
- convMxM.C

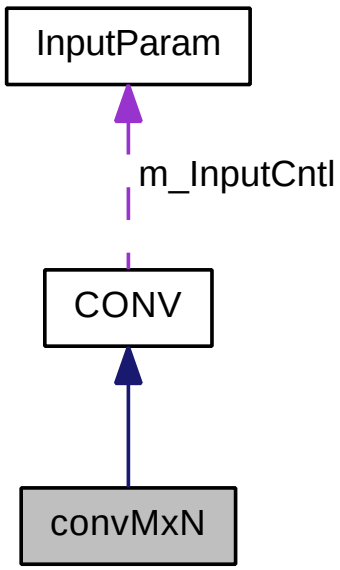
4.4 クラス convMxN

`#include <convMxN.h>`

convMxN に対する継承グラフ



convMxN のコラボレーション図



Public メソッド

- convMxN ()
- ~convMxN ()
- void VoxelInit ()
  領域分割と出力*DFI* のインスタンス

- bool exec ()
  MxN の実行

Public 変数

- int m_Gvoxel [3]
- int m_Gdiv [3]
- int m_Head [3]
- int m_Tail [3]
- vector< cio_DFI * > m_out_dfi

Additional Inherited Members

4.4.1 説明

convMxN.h の 24 行で定義されています。

4.4.2 コンストラクタとデストラクタ

4.4.2.1 convMxN::convMxN ( )

コンストラクタ

convMxN.C の 21 行で定義されています。

```
22 {
23
24 }
```

4.4.2.2 convMxN::~convMxN ( )

デストラクタ

convMxN.C の 28 行で定義されています。

参照先 m_out_dfi.

```
29 {
30   for(int i=0; i<m_out_dfi.size(); i++ ) if( !m_out_dfi[i] ) delete m_out_dfi[i];
31 }
```

4.4.3 関数

4.4.3.1 bool convMxN::exec ( ) [virtual]

MxN の実行

戻り値

エラーコード

CONVを実装しています。

convMxN.C の 200 行で定義されています。

参照先 CONV::convertXY(), CONV::DtypeMinMax(), InputParam::Get_OutdfiNameList(), InputParam::Get_OutputArrayShape(), InputParam::Get_OutputDataType(), InputParam::Get_OutputFilenameFormat(), InputParam::Get_OutputFormat(), InputParam::Get_OutputGuideCell(), InputParam::Get_ThinOut(), convOutput::importInputParam(), CONV::m_bgrid_interp_flag, m_Gdiv, m_Gvoxel, m_Head, CONV::m_in_dfi, CONV::m_InputCntl, CONV::m_numProc, m_out_dfi, CONV::m_paraMngr, m_Tail, と convOutput::OutputInit().

```
201 {
202
203   // 出力ファイル形式クラスのインスタンス
204   convOutput *ConvOut = convOutput::OutputInit(m_InputCntl->Get_OutputFormat());
205
206   // InputParam のインスタンス
207   if( !ConvOut->importInputParam(m_InputCntl) ) {
208     //Exit(0);
209     return false;
210   }
211
212   //出力ファイル名の取得
213   vector<std::string> out_dfi_name = m_InputCntl->Get_OutdfiNameList();
214   std::string prefix,outfile;
215
216   FILE *fp;
217   int dummy;
218
219   int d_type;
220
221   CIO::E_CIO_ERRORCODE ret;
222   double rtime;
223   unsigned idummy;
224   double ddummy;
225   float fminmax[8];
226   double dminmax[8];
227
228   bool mio;
229   mio = false;
230   if( m_numProc > 1 ) mio=true;
231
232   //間引き数のセット
233   int thin_count = m_InputCntl->Get_ThinOut();
234
235   //自ノードのボクセルサイズの取得
236   int sz[3];
237   const int* tmp = m_paraMngr->GetLocalVoxelSize();
238   for(int i=0; i<3; i++) sz[i]=tmp[i];
239
240   //出力 workarea のサイズ
241   int szS[3];
242   for(int i=0; i<3; i++) {
243     szS[i]=sz[i]/thin_count;
244     if( szS[i] < 1 ) {
245       printf("\toutput domain size error\n");
246       return false;
247     }
248     if( sz[i]%thin_count != 0 ) szS[i]++;
249   }
250
251   int head[3],tail[3];
252   for(int i=0; i<3; i++) {
253     head[i]=(m_Head[i]-1)/thin_count;
254     if( (m_Head[i]-1)%thin_count != 0 ) head[i]++;
255     tail[i]=(m_Tail[i]-1)/thin_count;
256   }
257   const double* dtmp;
258   double pit[3],org[3];
259   dtmp = m_paraMngr->GetPitch();
260   for(int i=0; i<3; i++) pit[i]=dtmp[i]*double(thin_count);
261   dtmp = m_paraMngr->GetGlobalOrigin();
262   for(int i=0; i<3; i++) org[i]=dtmp[i]+0.5*pit[i];
263   for(int i=0; i<3; i++) org[i]+=double(head[i])*pit[i];
264
265   const cio_FileInfo* DFI_FInfo = m_in_dfi[0]->GetcioFileInfo();
266
267   //dfi のループ
268   for (int i=0; i<m_in_dfi.size(); i++) {
269     int nComp = m_in_dfi[i]->GetNumComponent();
270
271     int outGc=0;
272     if( m_InputCntl->Get_OutputGuideCell() > 1 ) outGc = m_InputCntl->
    Get_OutputGuideCell();
273     if( outGc > 0 ) {
274       const cio_FileInfo* DFI_FInfo = m_in_dfi[i]->GetcioFileInfo();
275       if( outGc > DFI_FInfo->GuideCell ) outGc = DFI_FInfo->GuideCell;
276     }
277
278     if( thin_count > 1 ) outGc=0;
```

```
279     if( m_bgrid_interp_flag ) outGc=0;
280
281     //読込みバッファのインスタンス
282     cio_Array* buf = cio_Array::instanceArray
283     ( m_in_dfi[i]->GetDataType(),
284       m_in_dfi[i]->GetArrayShape(),
285       sz,
286       //0,
287       outGc,
288       //m_in_dfi[i]->GetNumComponent());
289       nComp);
290
291     //出力タイプのセット
292     if( m_InputCntl->Get_OutputDataType() == CIO::E_CIO_DTYPE_UNKNOWN )
293     {
294       d_type = m_in_dfi[i]->GetDataType();
295     } else {
296       d_type = m_InputCntl->Get_OutputDataType();
297     }
298
299     //出力バッファのインスタンス
300     cio_Array* src = cio_Array::instanceArray
301     ( (CIO::E_CIO_DTYPE)d_type,
302       //m_in_dfi[i]->GetArrayShape(),
303       (CIO::E_CIO_ARRAYSHAPE)m_InputCntl->Get_OutputArrayShape(),
304       szS,
305       //0,
306       outGc,
307       //m_in_dfi[i]->GetNumComponent());
308       nComp);
309
310     //DFI_FInfo クラスの取得
311     const cio_FileInfo* DFI_FInfo = m_in_dfi[i]->GetcioFileInfo();
312     prefix=DFI_FInfo->Prefix;
313
314     //TimeSlice クラスの取得
315     const cio_TimeSlice* TSlice = m_in_dfi[i]->GetcioTimeSlice();
316
317
318     //ステップ数のループ
319     for ( int j=0; j<TSlice->SliceList.size(); j++ ) {
320       //MxN の読込み
321       ret = m_in_dfi[i]->ReadData(buf,
322                                   (unsigned)TSlice->SliceList[j].step,
323                                   //0,
324                                   outGc,
325                                   m_Gvoxel,
326                                   m_Gdiv,
327                                   m_Head,
328                                   m_Tail,
329                                   rtime,
330                                   true,
331                                   idummy,
332                                   ddummy);
333       if( ret != CIO::E_CIO_SUCCESS ) {
334         printf("ReadData Error\n");
335         return false;
336       }
337
338       //読込みバッファの headIndex のセット
339       int headB[3];
340       for(int k=0; k<3; k++) headB[k]=m_Head[k]-1;
341       buf->setHeadIndex( headB );
342
343
344       //間引き及び型変換がない場合
345       if( thin_count == 1 && buf->getDataType() == src->getDataType() &&
346           buf->getArrayShape() == src->getArrayShape() ) {
347         src=buf;
348       } else {
349       //間引きまたは型変換がある場合
350         //出力バッファの間引きなしでの HeadIndex,TailIndex
351         int headS[3],tailS[3];
352         for(int k=0; k<3; k++) {
353           headS[k]=m_Head[k]-1;
354           tailS[k]=m_Tail[k]-1;
355         }
356         //出力バッファの HeadIndex セット
357         int headS0[3];
358         for(int k=0; k<3; k++) {
359           headS0[k]=headS[k]/thin_count;
360           if( headS[k]%thin_count != 0 ) headS0[k]++;
361         }
362         src->setHeadIndex( headS0 );
363         for(int n=0; n<nComp; n++) convertXY(buf,src,headS,tailS,n);
364       }
365
```

```
366       int output_fname = m_InputCntl->Get_OutputFilenameFormat();
367       m_out_dfi[i]->set_output_fname((CIO::E_CIO_OUTPUT_FNAME)output_fname);
368
369       //minmax の初期化
370       int nsize = nComp;
371       if( nComp > 1 ) nsize++;
372       double *min = new double[nsize];
373       double *max = new double[nsize];
374       for(int n=0; n<nsize; n++) {
375         min[n]=DBL_MAX;
376         max[n]=-DBL_MAX;
377       }
378       //minmax を求める
379       if( !DtypeMinMax(src,min,max) ) return false;
380
381       if( out_dfi_name.size() > 1 ) {
382         //ランク間で通信して MINMAX を求めてランク 0 に送信
383         int nbuff = nsize*1;
384         //min の通信
385         double *send1 = min;
386         double *recv1 = new double[nbuff];
387         MPI_Reduce(send1, recv1, nbuff, MPI_DOUBLE, MPI_MIN, 0, MPI_COMM_WORLD);
388         min = recv1;
389         //max の通信
390         double *send2 = max;
391         double *recv2 = new double[nbuff];
392         MPI_Reduce(send2, recv2, nbuff, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
393         max = recv2;
394       }
395
396       //出力処理
397       //double tmp_minmax[8];
398       double *tmp_minmax = new double[nsize*2];
399       for(int n=0; n<nsize; n++ ) {
400         tmp_minmax[n*2+0] = min[n];
401         tmp_minmax[n*2+1] = max[n];
402       }
403
404       ret = m_out_dfi[i]->WriteData(
405                                     (unsigned)TSlice->SliceList[j].step,
406                                     //0,
407                                     outGc,
408                                     rtime,
409                                     src,
410                                     tmp_minmax,
411                                     true,
412                                     idummy,
413                                     ddummy);
414
415
416     }
417     delete src;
418   }
419
420   return true;
421
422 }
```

4.4.3.2 void convMxN::VoxelInit ( ) [virtual]

領域分割と出力DFI のインスタンス

CONVを再定義しています。

convMxN.C の 35 行で定義されています。

参照先 Exit, InputParam::Get_OutdfiNameList(), InputParam::Get_OutprocNameList(), InputParam::Get_OutputArrayShape(), InputParam::Get_OutputDataType(), InputParam::Get_OutputDir(), InputParam::Get_OutputDivision(), InputParam::Get_OutputFormat(), InputParam::Get_OutputFormatType(), InputParam::Get_OutputGuideCell(), InputParam::Get_ThinOut(), CONV::m_bgrid_interp_flag, m_Gdiv, m_Gvoxel, m_Head, CONV::m_HostName, CONV::m_in_dfi, CONV::m_InputCntl, CONV::m_myRank, m_out_dfi, CONV::m_paraMngr, と m_Tail.

```
36 {
37   std::string outdfiname;
38   int iret=0;
39   const cio_Domain *DFI_Domain = m_in_dfi[0]->GetcioDomain();
40
41
42   //出力領域の分割数の取得
43   int* Gdiv = m_InputCntl->Get_OutputDivision();
```

```
44
45   if( Gdiv[0]>0 && Gdiv[1]>0 && Gdiv[2]>0 ) {
46     //分割数が指示されている場合
47     iret = m_paraMngr->VoxelInit(Gdiv, (int *)DFI_Domain->GlobalVoxel,
48                                  (double *)DFI_Domain->GlobalOrigin,
49                                  (double *)DFI_Domain->GlobalRegion, 0, 0);
50     if( iret != 0 ) {
51        printf("\tVoxelInit Error cpm_ErrorCode : %d\n",iret);
52        Exit(0);
53     }
54   } else {
55     //分割数が指示されていない場合
56     iret =  m_paraMngr->VoxelInit((int *)DFI_Domain->GlobalVoxel,
57                                   (double *)DFI_Domain->GlobalOrigin,
58                                   (double *)DFI_Domain->GlobalRegion, 0, 0);
59     if( iret != 0 ) {
60        printf("\tVoxelInit Error cpm_ErrorCode : %d\n",iret);
61        Exit(0);
62     }
63   }
64
65   //間引き数のセット
66   int thin_count = m_InputCntl->Get_ThinOut();
67
68   int voxel[3];
69   for(int i=0; i<3; i++) {
70     voxel[i]=DFI_Domain->GlobalVoxel[i]/thin_count;
71     if( DFI_Domain->GlobalVoxel[i]%thin_count != 0 ) voxel[i]++;
72   }
73
74   const int* tmp;
75   tmp = m_paraMngr->GetGlobalVoxelSize();
76   for(int i=0; i<3; i++) m_Gvoxel[i]=tmp[i];
77   tmp = m_paraMngr->GetVoxelHeadIndex();
78   for(int i=0; i<3; i++) m_Head[i]=tmp[i]+1;
79   tmp = m_paraMngr->GetVoxelTailIndex();
80   for(int i=0; i<3; i++) m_Tail[i]=tmp[i]+1;
81   tmp = m_paraMngr->GetDivNum();
82   for(int i=0; i<3; i++) m_Gdiv[i]=tmp[i];
83
84   int head[3],tail[3];
85   for(int i=0; i<3; i++) {
86     head[i]=(m_Head[i]-1)/thin_count;
87     if( (m_Head[i]-1)%thin_count != 0 ) head[i]++;
88     tail[i]=(m_Tail[i]-1)/thin_count;
89   }
90
91   const double* dtmp;
92   double pit[3],org[3];
93   dtmp = m_paraMngr->GetPitch();
94   for(int i=0; i<3; i++) pit[i]=dtmp[i];
95   for(int i=0; i<3; i++) pit[i]=dtmp[i]*double(thin_count);
96   dtmp = m_paraMngr->GetGlobalOrigin();
97   for(int i=0; i<3; i++) org[i]=dtmp[i];
98   //for(int i=0; i<3; i++) org[i]=dtmp[i]+0.5*pit[i];
99   for(int i=0; i<3; i++) org[i]+=double(head[i])*pit[i];
100
101   for(int i=0; i<3; i++) {
102     head[i]=head[i]+1;
103     tail[i]=tail[i]+1;
104   }
105
106   //出力ファイル名の取得
107   vector<std::string> out_dfi_name = m_InputCntl->Get_OutdfiNameList();
108   vector<std::string> out_proc_name = m_InputCntl->Get_OutprocNameList();
109
110   //出力 DFI の初期化
111   for(int i=0; i<m_in_dfi.size(); i++) {
112     const cio_FileInfo* DFI_FInfo = m_in_dfi[i]->GetcioFileInfo();
113
114     printf("ID : %d org : %e %e %e\n",m_myRank,org[0],org[1],org[2]);
115
116     std::string outdfifname="";
117     if( out_dfi_name.size()>1 ) outdfifname=out_dfi_name[i];
118     std::string outprocfname="";
119     if( out_proc_name.size()>1 ) outprocfname=out_proc_name[i];
120
121     //出力タイプのセット
122     int d_type;
123     if( m_InputCntl->Get_OutputDataType() == CIO::E_CIO_DTYPE_UNKNOWN )
124     {
125       d_type = m_in_dfi[i]->GetDataType();
126     } else {
127       d_type = m_InputCntl->Get_OutputDataType();
128     }
129
130     //出力ガイドセルの設定
```

```
131     int outGc=0;
132     if( m_InputCntl->Get_OutputGuideCell() > 1 ) outGc = m_InputCntl->
    Get_OutputGuideCell();
133     if( outGc > 1 ) {
134       const cio_FileInfo* DFI_FInfo = m_in_dfi[i]->GetcioFileInfo();
135       if( outGc > DFI_FInfo->GuideCell ) outGc=DFI_FInfo->GuideCell;
136     }
137     //間引きありのとき、出力ガイドセルを 0 に設定
138     if( thin_count > 1 ) outGc=0;
139     //格子点出力のとき、出力ガイドセルを 0 に設定
140     if( m_bgrid_interp_flag ) outGc=0;
141
142     cio_DFI* dfi=cio_DFI::WriteInit(MPI_COMM_WORLD,
143                             outdfifname,
144                             m_InputCntl->Get_OutputDir(),
145                             DFI_FInfo->Prefix,
146                             (CIO::E_CIO_FORMAT)m_InputCntl->Get_OutputFormat(),
147                             //0,
148                             outGc,
149                             (CIO::E_CIO_DTYPE)d_type,
150                             (CIO::E_CIO_ARRAYSHAPE)m_InputCntl->Get_OutputArrayShape(),
151                             DFI_FInfo->Component,
152                             outprocfname,
153                             voxel,
154                             pit,
155                             org,
156                             m_Gdiv,
157                             head,
158                             tail,
159                             m_HostName,
160                             CIO::E_CIO_OFF);
161     if( dfi == NULL ) {
162       printf("\tFails to instance dfi\n");
163       Exit(0);
164     }
165
166     //Proc ファイル出力
167     if( out_proc_name.size()>1 ) dfi->WriteProcDfiFile(MPI_COMM_WORLD,false);
168
169     printf("head : %d %d %d tail : %d %d %d\n",head[0],head[1],head[2],
170             tail[0],tail[1],tail[2]);
171
172     //出力形式 (ascii,binary,Fbinary) のセット
173     dfi->set_output_type((CIO::E_CIO_OUTPUT_TYPE)m_InputCntl->
    Get_OutputFormatType());
174
175     //Unit のセット
176     std::string unit;
177     double ref;
178     double diff;
179     bool bdiff;
180     m_in_dfi[i]->GetUnit("Length",unit,ref,diff,bdiff);
181     dfi->AddUnit("Length",unit,ref,diff,bdiff);
182     m_in_dfi[i]->GetUnit("Velocity",unit,ref,diff,bdiff);
183     dfi->AddUnit("Velocity",unit,ref,diff,bdiff);
184     m_in_dfi[i]->GetUnit("Pressure",unit,ref,diff,bdiff);
185     dfi->AddUnit("Pressure",unit,ref,diff,bdiff);
186
187     //成分名の取り出しとセット
188     for(int n=0; n<DFI_FInfo->Component; n++) {
189        std::string variable = m_in_dfi[i]->getComponentVariable(n);
190        if( variable != "" ) dfi->setComponentVariable(n,variable);
191     }
192
193     m_out_dfi.push_back(dfi);
194   }
195
196 }
```

4.4.4 変数

4.4.4.1 int convMxN::m_Gdiv[3]

convMxN.h の 28 行で定義されています。

参照元 exec(), と VoxelInit().

4.4.4.2 int convMxN::m_Gvoxel[3]

convMxN.h の 27 行で定義されています。

参照元 exec(), と VoxelInit().

4.4.4.3 int convMxN::m_Head[3]

convMxN.h の 29 行で定義されています。

参照元 exec(), と VoxelInit().

4.4.4.4 vector<cio_DFI *> convMxN::m_out_dfi

convMxN.h の 32 行で定義されています。

参照元 exec(), VoxelInit(), と ~convMxN().

4.4.4.5 int convMxN::m_Tail[3]

convMxN.h の 30 行で定義されています。

参照元 exec(), と VoxelInit().

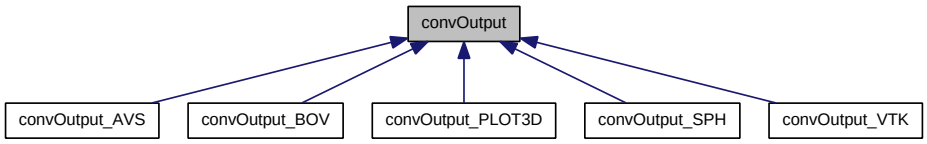
このクラスの説明は次のファイルから生成されました:

- convMxN.h
- convMxN.C

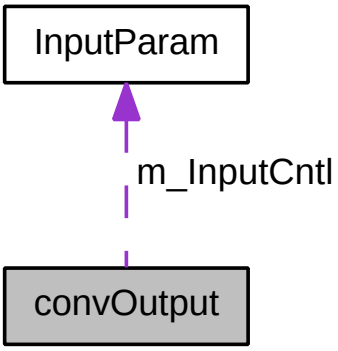
4.5 クラス convOutput

`#include <convOutput.h>`

convOutput に対する継承グラフ



convOutput のコラボレーション図



Public メソッド

- convOutput ()
- ~convOutput ()
- bool importInputParam (InputParam *InputCntl)

  InputParam のポインタをコピー
- virtual FILE * OutputFile_Open (const std::string prefix, const unsigned step, const int id, const bool mio=false)

  出力ファイルをオープンする
- virtual void OutputFile_Close (FILE *fp)

  出力ファイルをクローズする
- virtual void WriteGridData (int step, int myRank, int guide, double org[3], double pit[3], int sz[3])

  grid 出力 *(plot3d* 用*)*
- virtual bool WriteHeaderRecord (int step, int dim, int d_type, int imax, int jmax, int kmax, double time, double *org, double *pit, std::string prefix, FILE *fp)

  sph ファイルの *header* の書き込み
- virtual bool WriteFieldData (FILE *fp, cio_Array *src, size_t dLen)

  Field Datat 出力
- virtual bool WriteDataMarker (int dmy, FILE *fp, bool out=false)

  マーカーの書き込み
- virtual void output_avs (int myRank, vector< cio_DFI * >in_dfi, cpm_ParaManager *paraMngr=NULL, int *head=NULL)

  avs の　ヘッダーレコード出力コントロール

Static Public メソッド

- static convOutput * OutputInit (const int out_format)

  出力クラスのインスタンス

Public 変数

- InputParam * m_InputCntl

4.5.1 説明

convOutput.h の 36 行で定義されています。

4.5.2 コンストラクタとデストラクタ

4.5.2.1 convOutput::convOutput ( )

コンストラクタ

convOutput.C の 26 行で定義されています。

```
27 {
28
29 }
```

4.5.2.2 convOutput::∼convOutput ( )

デストラクタ

convOutput.C の 33 行で定義されています。

```
34 {
35
36
37 }
```

4.5.3 関数

4.5.3.1 bool convOutput::importInputParam (InputParam * *InputCntl*)

InputParam のポインタをコピー

引数

in	*InputCntl*	InputParam クラスポインタ

戻り値

エラーコード

convOutput.C の 41 行で定義されています。

参照先 m_InputCntl.

参照元 convMx1::exec(), と convMxN::exec().

```
42 {
43   if( !InputCntl ) return false;
44   m_InputCntl = InputCntl;
45   return true;
46 }
```

4.5.3.2 virtual void convOutput::output_avs (int *myRank*, vector< cio_DFI * > *in_dfi*, cpm_ParaManager * *paraMngr* = NULL, int * *head* = NULL) [inline], [virtual]

avs の ヘッダーレコード出力コントロール

引数

in	*myRank*	ランクID
in	*in_dfi*	dfi のポインタ配列
in	*paraMngr*	パラマネージャー（MxN 以外は省略)
in	*head*	head インデックス (MxN 以外は省略）

convOutput_AVSで再定義されています。

convOutput.h の 163 行で定義されています。

参照元 convMx1::exec().

```
167                                {};
```

4.5.3.3 virtual void convOutput::OutputFile_Close (FILE * *fp*) [inline], [virtual]

出力ファイルをクローズする

引数

in	*fp*	ファイルポインタ

convOutput_VTKで再定義されています。

convOutput.h の 85 行で定義されています。

参照元 convMx1::exec().

```
86   { fclose(fp); };
```

4.5.3.4 virtual FILE* convOutput::OutputFile_Open (const std::string *prefix*, const unsigned *step*, const int *id*, const bool *mio* = false) [inline], [virtual]

出力ファイルをオープンする

引数

in	*prefix*	ファイル接頭文字
in	*step*	ステップ数
in	*id*	ランク番号
in	*mio*	出力時の分割指定　true = local / false = gather(default)

convOutput_PLOT3D, convOutput_BOV, convOutput_SPH, convOutput_VTK, と convOutput_AVSで再定義されています。

convOutput.h の 73 行で定義されています。

参照元 convMx1::exec().

```
78   { return NULL; };
```

4.5.3.5 convOutput * convOutput::OutputInit (const int *out_format*) [static]

出力クラスのインスタンス

引数

in	*out_format*	出力ファイルフォーマット

戻り値

convOutput クラスポインタ

convOutput.C の 51 行で定義されています。

参照元 convMx1::exec(), と convMxN::exec().

```
52 {
53
54   convOutput *OutConv = NULL;
55
56   if     ( out_format == CIO::E_CIO_FMT_SPH    ) OutConv = new convOutput_SPH();
57   else if( out_format == CIO::E_CIO_FMT_BOV    ) OutConv = new convOutput_BOV();
58   else if( out_format == CIO::E_CIO_FMT_AVS    ) OutConv = new convOutput_AVS();
59   else if( out_format == CIO::E_CIO_FMT_VTK    ) OutConv = new convOutput_VTK();
60   else if( out_format == CIO::E_CIO_FMT_PLOT3D ) OutConv = new convOutput_PLOT3D();
61
62   return OutConv;
63
64 }
```

4.5.3.6 virtual bool convOutput::WriteDataMarker (int *dmy*, FILE * *fp*, bool *out* = false) [inline], [virtual]

マーカーの書き込み

引数

in	*dmy*	マーカー
in	*fp*	ファイルポインタ
in	*out*	plot3d 用Fortran 出力フラグ　通常は false

convOutput_PLOT3D, convOutput_VTK, と convOutput_SPHで再定義されています。

convOutput.h の 152 行で定義されています。

参照元 convMx1::exec().

```
152 { return true; };
```

4.5.3.7 bool convOutput::WriteFieldData (FILE * *fp*, cio_Array * *src*, size_t *dLen*) [virtual]

Field Datat 出力

引数

in	*fp*	出力ファイルポインタ
in	*src*	出力データ配列ポインタ
in	*dLen*	出力データサイズ

convOutput_PLOT3D, convOutput_VTK, と convOutput_AVSで再定義されています。

convOutput.C の 68 行で定義されています。

参照先 Exit.

参照元 convMx1::convMx1_out_ijkn(), と convMx1::convMx1_out_nijk().

```
69 {
70   if( src->writeBinary(fp) != dLen ) Exit(0);
71   return true;
72 }
```

4.5.3.8 virtual void convOutput::WriteGridData (int *step*, int *myRank*, int *guide*, double *org[3]*, double *pit[3]*, int *sz[3]*) [inline], [virtual]

grid 出力 (plot3d 用)

引数

in	*step*	step 番号
in	*myRank*	ランク番号
in	*guide*	ガイドセル数
in	*org*	原点座標値
in	*pit*	ピッチ
in	*sz*	ボクセルサイズ

convOutput_PLOT3Dで再定義されています。

convOutput.h の 98 行で定義されています。

参照元 convMx1::exec().

```
104                           {};
```

4.5.3.9 virtual bool convOutput::WriteHeaderRecord (int *step*, int *dim*, int *d_type*, int *imax*, int *jmax*, int *kmax*, double *time*, double * *org*, double * *pit*, std::string *prefix*, FILE * *fp*) [inline], [virtual]

sph ファイルの header の書き込み

引数

in	*step*	ステップ数
in	*dim*	成分数
in	*d_type*	データ型タイプ
in	*imax*	x 方向ボクセルサイズ
in	*jmax*	y 方向ボクセルサイズ
in	*kmax*	z 方向ボクセルサイズ
in	*time*	時間
in	*org*	原点座標
in	*pit*	ピッチ
in	*prefix*	ファイル接頭文字
in	*fp*	ファイルポインタ

convOutput_PLOT3D, convOutput_SPH, と convOutput_VTKで再定義されています。

convOutput.h の 121 行で定義されています。

参照元 convMx1::exec().

```
132   { return true; };
```

4.5.4 変数

4.5.4.1 InputParam* convOutput::m_InputCntl

convOutput.h の 40 行で定義されています。

参照元 importInputParam(), convOutput_AVS::output_avs(), convOutput_AVS::output_avs_coord(), convOutput_AVS::output_avs_header(), convOutput_AVS::output_avs_Mx1(), convOutput_AVS::output_avs_MxM(), convOutput_AVS::output_avs_MxN(), convOutput_AVS::OutputFile_Open(), convOutput_SPH::OutputFile_Open(), convOutput_VTK::OutputFile_Open(), convOutput_BOV::OutputFile_Open(), convOutput_PLOT3D::OutputFile_Open(), convOutput_PLOT3D::OutputPlot3D_xyz(), convOutput_PLOT3D::WriteBlockData(), convOutput_PLOT3D::WriteDataMarker(), convOutput_VTK::WriteFieldData(), convOutput_PLOT3D::WriteFuncBlockData(), convOutput_PLOT3D::WriteFuncData(), convOutput_PLOT3D::WriteGridData(), convOutput_VTK::WriteHeaderRecord(), convOutput_PLOT3D::WriteNgrid(), と convOutput_PLOT3D::WriteXYZData().

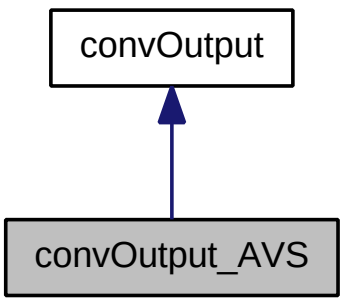
このクラスの説明は次のファイルから生成されました:

- convOutput.h
- convOutput.C

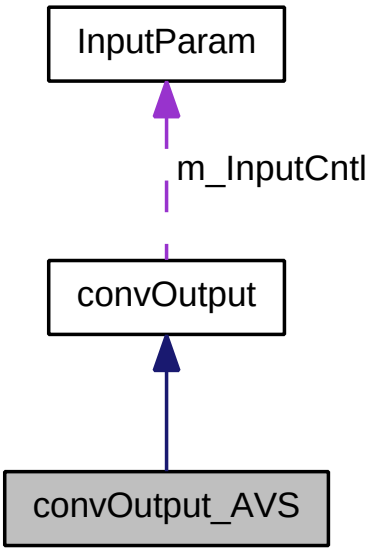
4.6 クラス convOutput_AVS

```
#include <convOutput_AVS.h>
```

convOutput_AVS に対する継承グラフ



convOutput_AVS のコラボレーション図



Public メソッド

- convOutput_AVS ()
- ~convOutput_AVS ()
- FILE * OutputFile_Open (const std::string prefix, const unsigned step, const int id, const bool mio)

  出力ファイルをオープンする
- bool WriteFieldData (FILE *fp, cio_Array *src, size_t dLen)

  Field Datat **出力**
- void output_avs (int myRank, vector< cio_DFI * >in_dfi, cpm_ParaManager *paraMngr=NULL, int *head=NULL)

  avs **ファイルのヘッダー処理**

Protected メソッド

- void output_avs_Mx1 (int myRank, vector< cio_DFI * >in_dfi)

  avs **ファイルのヘッダー処理（***Mx1)*
- void output_avs_MxM (int myRank, vector< cio_DFI * >in_dfi)

  avs **ファイルのヘッダー処理（***MxM)*
- void output_avs_MxN (int myRank, vector< cio_DFI * >in_dfi, cpm_ParaManager *paraMngr, int *head)

  avs **ファイルのヘッダー処理（***MxN)*
- void output_avs_coord (int RankID, bool mio, double min_ext[3], double max_ext[3])

  avs coord data **ファイル出力**
- void output_avs_header (cio_DFI *dfi, int RankID, bool mio, int ndim, int nspace, int dims[3])

  avs **ファイルヘッダー出力**

Additional Inherited Members

4.6.1 説明

convOutput_AVS.h の 23 行で定義されています。

4.6.2 コンストラクタとデストラクタ

4.6.2.1 convOutput_AVS::convOutput_AVS ( )

コンストラクタ

convOutput_AVS.C の 22 行で定義されています。

```
23 {
24
25
26 }
```

4.6.2.2 convOutput_AVS::~convOutput_AVS ( )

デストラクタ

convOutput_AVS.C の 30 行で定義されています。

```
31 {
32
33
34 }
```

4.6.3 関数

4.6.3.1 void convOutput_AVS::output_avs (int *myRank,* vector< cio_DFI * > *in_dfi,* cpm_ParaManager * *paraMngr* = NULL, int * *head* = NULL) [virtual]

avs ファイルのヘッダー処理

引数

in	*myRank*	rankID
in	*in_dfi*	dfi のポインター
in	*paraMngr*	パラマネージャー
in	*head*	head インデックス

convOutputを再定義しています。

convOutput_AVS.C の 94 行で定義されています。

参照先 E_OUTPUT_Mx1, E_OUTPUT_MxM, E_OUTPUT_MxN, InputParam::Get_ConvType(), convOutput::m_InputCntl, output_avs_Mx1(), output_avs_MxM(), と output_avs_MxN().

```
99 {
100
101   if      ( m_InputCntl->Get_ConvType() == E_OUTPUT_Mx1 ) {
102     output_avs_Mx1(myRank, in_dfi);
103   }else if( m_InputCntl->Get_ConvType() == E_OUTPUT_MxM ) {
104     output_avs_MxM(myRank, in_dfi);
105   }else if( m_InputCntl->Get_ConvType() == E_OUTPUT_MxN ) {
106     output_avs_MxN(myRank, in_dfi, paraMngr, head);
107   }
108
109 }
```

4.6.3.2 void convOutput_AVS::output_avs_coord (int *RankID,* bool *mio,* double *min_ext[3],* double *max_ext[3]*) [protected]

avs coord data ファイル出力

引数

in	*RankID*	ランクID
in	*mio*	分割出力指示
in	*min_ext*	座標値の最小値
in	*max_ext*	座標値の最大値

convOutput_AVS.C の 279 行で定義されています。

参照先 Exit, InputParam::Get_OutputDir(), InputParam::Get_OutputFilenameFormat(), と convOutput::m_InputCntl.

参照元 output_avs_Mx1(), output_avs_MxM(), と output_avs_MxN().

```
283 {
284
285   FILE* fp;
286   std::string cod_fname;
287
288   //座標値データファイルオープン
289   int fnameformat = m_InputCntl->Get_OutputFilenameFormat();
290   cod_fname = m_InputCntl->Get_OutputDir() +"/"+
291               cio_DFI::Generate_FileName("cord",
292                                          RankID,
293                                          -1,
294                                          "cod",
295                                          (CIO::E_CIO_OUTPUT_FNAME)fnameformat,
296                                          mio,
297                                          CIO::E_CIO_OFF);
298
299   if( (fp = fopen(cod_fname.c_str(),"w")) == NULL ) {
300     printf("\tCan't open file.(%s)\n",cod_fname.c_str());
301     Exit(0);
302   }
303
304   //座標値データ（min,max）の出力
305   fprintf(fp,"#### X #####\n");
306   fprintf(fp,"%.6f\n",min_ext[0]);
307   fprintf(fp,"%.6f\n",max_ext[0]);
308   fprintf(fp,"#### Y #####\n");
309   fprintf(fp,"%.6f\n",min_ext[1]);
310   fprintf(fp,"%.6f\n",max_ext[1]);
311   fprintf(fp,"#### Z #####\n");
312   fprintf(fp,"%.6f\n",min_ext[2]);
313   fprintf(fp,"%.6f\n",max_ext[2]);
314
315   //座標値データファイルクローズ
316   fclose(fp);
317
318 }
```

4.6.3.3 void convOutput_AVS::output_avs_header (cio_DFI * *dfi*, int *RankID*, bool *mio*, int *ndim*, int *nspace*, int *dims[3]*) [protected]

avs ファイルヘッダー出力

引数

in	*dfi*	cio_DFI クラスポインタ
in	*RankID*	ランクID
in	*mio*	分割出力指示
in	*ndim*	3
in	*nspace*	3
in	*dims*	サイズ

convOutput_AVS.C の 321 行で定義されています。

参照先 Exit, InputParam::Get_OutputDir(), InputParam::Get_OutputFilenameFormat(), と convOutput::m_InputCntl.

参照元 output_avs_Mx1(), output_avs_MxM(), と output_avs_MxN().

```
327 {
```

```
328
329   FILE* fp;
330   std::string dType;
331   std::string fld_fname, out_fname;
332   std::string cod_fname;
333
334   //cio_FileInfo クラスポインタの取得
335   const cio_FileInfo* DFI_FInfo = dfi->GetcioFileInfo();
336   //cio_TimeSlice クラスポインタの取得
337   const cio_TimeSlice* TSlice   = dfi->GetcioTimeSlice();
338
339   //データタイプのセット
340   if( dfi->GetDataType() == CIO::E_CIO_FLOAT32 ) {
341     dType = "float";
342   } else if( dfi->GetDataType() == CIO::E_CIO_FLOAT64 ) {
343     dType = "double";
344   } else {
345     dType = dfi->GetDataTypeString();
346     printf("\tillergal data type.(%s)\n",dType.c_str());
347     Exit(0);
348   }
349
350   //出力ヘッダーファイルオープン
351   int fnameformat = m_InputCntl->Get_OutputFilenameFormat();
352   fld_fname = m_InputCntl->Get_OutputDir() +"/"+
353               cio_DFI::Generate_FileName(DFI_FInfo->Prefix,
354                                          RankID,
355                                          -1,
356                                          "fld",
357                                          (CIO::E_CIO_OUTPUT_FNAME)fnameformat,
358                                          mio,
359                                          CIO::E_CIO_OFF);
360
361   if( (fp = fopen(fld_fname.c_str(),"w")) == NULL ) {
362     printf("\tCan't open file.(%s)\n",fld_fname.c_str());
363     Exit(0);
364   }
365
366   //先頭レコードの出力
367   fprintf(fp,"# AVS field file\n");
368
369   //計算空間の次元数を出力
370   fprintf(fp,"ndim=%d\n",ndim);
371
372   //計算空間サイズを出力
373   fprintf(fp,"dim1=%d\n",dims[0]+1);
374   fprintf(fp,"dim2=%d\n",dims[1]+1);
375   fprintf(fp,"dim3=%d\n",dims[2]+1);
376
377   //物理空間の次元数を出力
378   fprintf(fp,"nspace=%d\n",nspace);
379
380   //成分数の出力
381   fprintf(fp,"veclen=%d\n",DFI_FInfo->Component);
382
383   //データのタイプ出力
384   fprintf(fp,"data=%s\n",dType.c_str());
385
386   //座標定義情報の出力
387   fprintf(fp,"field=uniform\n");
388
389   //label の出力
390   for(int j=0; j<DFI_FInfo->Component; j++) {
391     std::string label=dfi->getComponentVariable(j);
392     if( label == "" ) continue;
393     fprintf(fp,"label=%s\n",label.c_str());
394   }
395
396   //step 毎の出力
397   if( TSlice->SliceList.size()>1 ) {
398     fprintf(fp,"nstep=%d\n",TSlice->SliceList.size());
399   }
400   for(int j=0; j<TSlice->SliceList.size(); j++ ) {
401     fprintf(fp,"time value=%.6f\n",TSlice->SliceList[j].time);
402     for(int n=1; n<=DFI_FInfo->Component; n++) {
403       int skip;
404       /*
405       if( dType == "float" ) {
406         skip=96+(n-1)*4;
407       } else {
408         skip=140+(n-1)*8;
409       }
410       */
411       skip=0;
412       out_fname = cio_DFI::Generate_FileName(DFI_FInfo->Prefix,
413                                              RankID,
414                                              TSlice->SliceList[j].step,
```

```
415                                             "sph",
416                                             (CIO::E_CIO_OUTPUT_FNAME)fnameformat,
417                                             mio,
418                                             CIO::E_CIO_OFF);
419
420       fprintf(fp,"variable %d file=%s filetype=binary skip=%d stride=%d\n",
421               n,out_fname.c_str(),skip,DFI_FInfo->Component);
422     }
423     //coord data
424     cod_fname = cio_DFI::Generate_FileName("cord",
425                                            RankID,
426                                            -1,
427                                            "cod",
428                                            (CIO::E_CIO_OUTPUT_FNAME)fnameformat,
429                                            mio,
430                                            CIO::E_CIO_OFF);
431
432     fprintf(fp,"coord 1 file=%s filetype=ascii skip=1\n",cod_fname.c_str());
433     fprintf(fp,"coord 2 file=%s filetype=ascii skip=4\n",cod_fname.c_str());
434     fprintf(fp,"coord 3 file=%s filetype=ascii skip=7\n",cod_fname.c_str());
435     fprintf(fp,"EOT\n");
436   }
437
438
439   //出力ヘッダーファイルクローズ
440   fclose(fp);
441
442 }
```

4.6.3.4 void convOutput_AVS::output_avs_Mx1 (int *myRank*, vector< cio_DFI * > *in_dfi*) [protected]

avs ファイルのヘッダー処理（Mx1)

引数

in	*myRank*	rankID
in	*in_dfi*	dfi のポインター

convOutput_AVS.C の 113 行で定義されています。

参照先 InputParam::Get_ThinOut(), convOutput::m_InputCntl, output_avs_coord(), と output_avs_header().

参照元 output_avs().

```
115 {
116
117   if( myRank != 0 ) return; //myRank==0 のときのみヘッダーレコードを出力
118
119   //間引き数のセット
120   int thin_count = m_InputCntl->Get_ThinOut();
121
122   int ndim,nspace;
123
124   int dims[3];
125   double min_ext[3],max_ext[3];
126
127   for(int i=0; i<in_dfi.size(); i++) {
128
129     //cio_Domain クラスポインタの取得
130     const cio_Domain* DFI_Domain  = in_dfi[i]->GetcioDomain();
131
132     //間引きを考慮しての計算空間サイズをセット
133     dims[0]=DFI_Domain->GlobalVoxel[0]/thin_count;
134     dims[1]=DFI_Domain->GlobalVoxel[1]/thin_count;
135     dims[2]=DFI_Domain->GlobalVoxel[2]/thin_count;
136     if(DFI_Domain->GlobalVoxel[0]%thin_count != 0 ) dims[0]++;
137     if(DFI_Domain->GlobalVoxel[1]%thin_count != 0 ) dims[1]++;
138     if(DFI_Domain->GlobalVoxel[2]%thin_count != 0 ) dims[2]++;
139
140     if( i==0 ) {
141       //座標値の最小値、最大値をセット
142       for(int j=0; j<3; j++) {
143         double pit=(DFI_Domain->GlobalRegion[j])/(double)(DFI_Domain->GlobalVoxel[j])*thin_count;
144         //min_ext[j]=DFI_Domain->GlobalOrigin[j]+0.5*(pit*(double)thin_count);
145         //
      max_ext[j]=(DFI_Domain->GlobalOrigin[j]+DFI_Domain->GlobalRegion[j])-0.5*(pit*(double)thin_count);
146         min_ext[j]=DFI_Domain->GlobalOrigin[j];
147         max_ext[j]=DFI_Domain->GlobalOrigin[j]+(pit*(double)dims[j]);
148       }
149
```

```
150       //coord データファイル出力
151       output_avs_coord(myRank, false, min_ext, max_ext);
152     }
153
154     //avs のヘッダーファイル出力
155     ndim=3;
156     nspace=3;
157     output_avs_header(in_dfi[i], myRank, false, ndim, nspace, dims);
158
159   }
160
161   return;
162 }
```

4.6.3.5 void convOutput_AVS::output_avs_MxM (int *myRank*, vector< cio_DFI * > *in_dfi*) [protected]

avs ファイルのヘッダー処理（MxM)

引数

in	*myRank*	rankID
in	*in_dfi*	dfi のポインター

convOutput_AVS.C の 165 行で定義されています。

参照先 InputParam::Get_ThinOut(), convOutput::m_InputCntl, output_avs_coord(), と output_avs_header().

参照元 output_avs().

```
167 {
168   if( myRank != 0 ) return; //myRank==0 のときのみヘッダーレコードを出力
169
170   //間引き数のセット
171   int thin_count = m_InputCntl->Get_ThinOut();
172
173   int ndim,nspace;
174
175   int dims[3];
176   double min_ext[3],max_ext[3];
177
178   double pit[3];
179
180   for(int i=0; i<in_dfi.size(); i++) {
181     //cio_Domain クラスポインタの取得
182     const cio_Domain* DFI_Domain  = in_dfi[i]->GetcioDomain();
183     //ピッチのセット
184     pit[0]=(DFI_Domain->GlobalRegion[0]/(double)DFI_Domain->GlobalVoxel[0])*(double)thin_count;
185     pit[1]=(DFI_Domain->GlobalRegion[1]/(double)DFI_Domain->GlobalVoxel[1])*(double)thin_count;
186     pit[2]=(DFI_Domain->GlobalRegion[2]/(double)DFI_Domain->GlobalVoxel[2])*(double)thin_count;
187
188     //cio_Process クラスポインタの取得
189     const cio_Process* DFI_Process = in_dfi[i]->GetcioProcess();
190
191     for(int j=0; j<DFI_Process->RankList.size(); j++) {
192       //間引きを考慮しての計算空間サイズをセット
193       dims[0]=DFI_Process->RankList[j].VoxelSize[0]/thin_count;
194       dims[1]=DFI_Process->RankList[j].VoxelSize[1]/thin_count;
195       dims[2]=DFI_Process->RankList[j].VoxelSize[2]/thin_count;
196       if(DFI_Process->RankList[j].VoxelSize[0]%thin_count != 0) dims[0]++;
197       if(DFI_Process->RankList[j].VoxelSize[1]%thin_count != 0) dims[1]++;
198       if(DFI_Process->RankList[j].VoxelSize[2]%thin_count != 0) dims[2]++;
199
200       if( i==0 ) {
201         //座標値の最小値、最大値をセット
202         for(int k=0; k<3; k++) {
203           int head = (DFI_Process->RankList[j].HeadIndex[k]-1)/thin_count;
204           if( (DFI_Process->RankList[j].HeadIndex[k]-1)%thin_count != 0 ) head++;
205           min_ext[k]=DFI_Domain->GlobalOrigin[k]+0.5*pit[k]+double(head)*pit[k];
206           max_ext[k]=min_ext[k]+(double(dims[k]-1))*pit[k];
207         }
208
209         //coord データファイル出力
210         output_avs_coord(j, true, min_ext, max_ext);
211       }
212
213       //avs のヘッダーファイル出力
214       ndim=3;
215       nspace=3;
216       output_avs_header(in_dfi[i], j, true, ndim, nspace, dims);
217
```

```
218     }
219   }
220
221   return;
222
223 }
```

4.6.3.6 void convOutput_AVS::output_avs_MxN (int *myRank,* vector< cio_DFI * > *in_dfi,* cpm_ParaManager * *paraMngr,* int * *head*) [protected]

avs ファイルのヘッダー処理（MxN)

引数

in	*myRank*	rankID
in	*in_dfi*	dfi のポインター
in	*paraMngr*	パラマネージャー
in	*head*	head インデックス

convOutput_AVS.C の 227 行で定義されています。

参照先 InputParam::Get_ThinOut(), convOutput::m_InputCntl, output_avs_coord(), と output_avs_header().

参照元 output_avs().

```
231 {
232   int ndim,nspace;
233   int dims[3];
234   double min_ext[3],max_ext[3];
235
236   //間引き数のセット
237   int thin_count = m_InputCntl->Get_ThinOut();
238
239   //自ノードでのボクセルサイズ取得
240   const int* tmp = paraMngr->GetLocalVoxelSize();
241
242   //間引きを考慮しての計算空間サイズをセット
243   for(int i=0; i<3; i++) {
244     dims[i]=tmp[i]/thin_count;
245     if( tmp[i]%thin_count != 0 ) dims[i]++;
246   }
247
248   //pit を取得
249   const double* dtmp;
250   double pit[3];
251   dtmp = paraMngr->GetPitch();
252   for(int i=0; i<3; i++) pit[i]=dtmp[i]*double(thin_count);
253
254   //座標値の最小値、最大値をセット
255   dtmp = paraMngr->GetGlobalOrigin();
256   for(int i=0; i<3; i++) {
257     int head=(mHead[i]-1)/thin_count;
258     if( (mHead[i]-1)%thin_count ) head++;
259     min_ext[i]=dtmp[i]+0.5*pit[i]+double(head)*pit[i];
260     max_ext[i]=min_ext[i]+(double(dims[i]-1))*pit[i];
261   }
262
263   //coord データファイル出力
264   output_avs_coord(myRank, true, min_ext, max_ext);
265
266   for(int i=0; i<in_dfi.size(); i++) {
267     //avs のヘッダーファイル出力
268     ndim=3;
269     nspace=3;
270     output_avs_header(in_dfi[i], myRank, true, ndim, nspace, dims);
271   }
272
273   return;
274
275 }
```

4.6.3.7 FILE * convOutput_AVS::OutputFile_Open (const std::string *prefix,* const unsigned *step,* const int *id,* const bool *mio*) [virtual]

出力ファイルをオープンする

引数

in	*prefix*	ファイル接頭文字
in	*step*	ステップ数
in	*id*	ランク番号
in	*mio*	出力時の分割指定　true = local / false = gather(default)

convOutputを再定義しています。

convOutput_AVS.C の 38 行で定義されています。

参照先 Exit, InputParam::Get_OutputDir(), InputParam::Get_OutputFilenameFormat(), と convOutput::m_InputCntl.

```
42 {
43   FILE* fp;
44
45   //ファイル名の生成
46   std::string outfile;
47   int fnameformat = m_InputCntl->Get_OutputFilenameFormat();
48   outfile = m_InputCntl->Get_OutputDir()+"/"+
49             cio_DFI::Generate_FileName(prefix,
50                                        id,
51                                        step,
52                                        "dat",
53                                        (CIO::E_CIO_OUTPUT_FNAME)fnameformat,
54                                        mio,
55                                        CIO::E_CIO_OFF);
56
57   printf("outfile : %s\n",outfile.c_str());
58
59   //ファイルオープン
60   if( (fp = fopen(outfile.c_str(), "wb")) == NULL ) {
61     printf("\tCan't open file.(%s)\n",outfile.c_str());
62     Exit(0);
63   }
64
65   return fp;
66 }
```

4.6.3.8 bool convOutput_AVS::WriteFieldData (FILE * *fp*, cio_Array * *src*, size_t *dLen*) [virtual]

Field Datat 出力

引数

in	*fp*	出力ファイルポインタ
in	*src*	出力データ配列ポインタ
in	*dLen*	出力データサイズ

convOutputを再定義しています。

convOutput_AVS.C の 70 行で定義されています。

参照先 Exit.

```
71 {
72
73   const int* sz = src->getArraySizeInt();
74
75   cio_Array *out = cio_Array::instanceArray
76                    (src->getDataType(),
77                     src->getArrayShape(),
78                     (int *)sz,
79                     0,
80                     src->getNcomp());
81   int ret = src->copyArray(out);
82   if( out->writeBinary(fp) != dLen ) {
83     delete out;
84     Exit(0);
85   }
86
87   delete out;
88   return true;
89 }
```

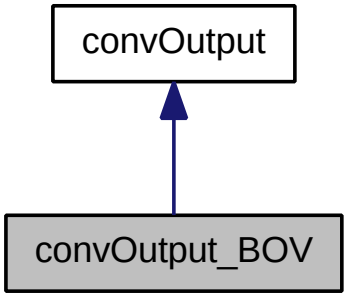
このクラスの説明は次のファイルから生成されました:

- convOutput_AVS.h
- convOutput_AVS.C

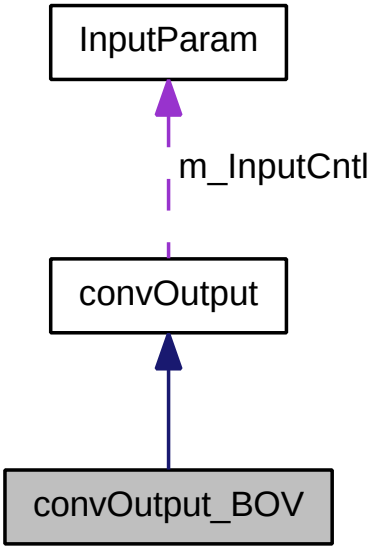
4.7 クラス convOutput_BOV

`#include <convOutput_BOV.h>`

convOutput_BOV に対する継承グラフ



convOutput_BOV のコラボレーション図



Public メソッド

- convOutput_BOV ()
- ~convOutput_BOV ()
- FILE * OutputFile_Open (const std::string prefix, const unsigned step, const int id, const bool mio)

  出力ファイルをオープンする

Additional Inherited Members

4.7.1 説明

convOutput_BOV.h の 23 行で定義されています。

4.7.2 コンストラクタとデストラクタ

4.7.2.1 convOutput_BOV::convOutput_BOV ( )

コンストラクタ

convOutput_BOV.C の 22 行で定義されています。

```
23 {
24
25
26 }
```

4.7.2.2 convOutput_BOV::∼convOutput_BOV ( )

デストラクタ

convOutput_BOV.C の 30 行で定義されています。

```
31 {
32
33
34 }
```

4.7.3 関数

4.7.3.1 FILE ∗ convOutput_BOV::OutputFile_Open (const std::string *prefix*, const unsigned *step*, const int *id*, const bool *mio*) [virtual]

出力ファイルをオープンする

引数

in	*prefix*	ファイル接頭文字
in	*step*	ステップ数
in	*id*	ランク番号
in	*mio*	出力時の分割指定　true = local / false = gather(default)

convOutputを再定義しています。

convOutput_BOV.C の 38 行で定義されています。

参照先 Exit, InputParam::Get_OutputDir(), InputParam::Get_OutputFilenameFormat(), と convOutput::m_InputCntl.

```
43 {
44   FILE* fp;
45
46   //ファイル名の生成
47   std::string outfile;
48   int fnameformat = m_InputCntl->Get_OutputFilenameFormat();
49   outfile = m_InputCntl->Get_OutputDir()+"/"+
50             cio_DFI::Generate_FileName(prefix,
51                                        id,
52                                        step,
53                                        "dat",
54                                        (CIO::E_CIO_OUTPUT_FNAME)fnameformat,
55                                        mio,
56                                        CIO::E_CIO_OFF);
57
58   printf("outfile : %s\n",outfile.c_str());
```

```
59
60   //ファイルオープン
61   if( (fp = fopen(outfile.c_str(), "wb")) == NULL ) {
62     printf("\tCan't open file.(%s)\n",outfile.c_str());
63     Exit(0);
64   }
65
66   return fp;
67
68 }
```

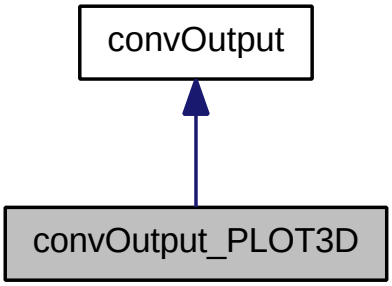
このクラスの説明は次のファイルから生成されました:

- convOutput_BOV.h
- convOutput_BOV.C

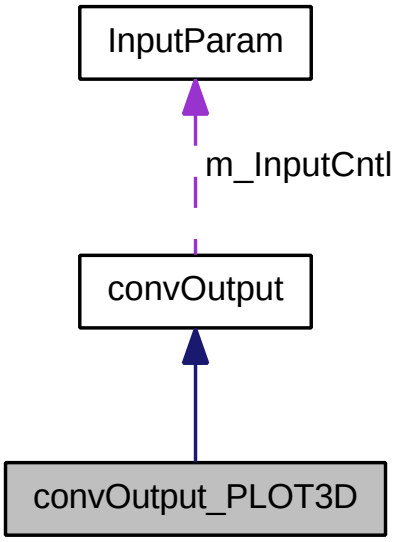
4.8 クラス convOutput_PLOT3D

`#include <convOutput_PLOT3D.h>`

convOutput_PLOT3D に対する継承グラフ



convOutput_PLOT3D のコラボレーション図



Public メソッド

- convOutput_PLOT3D ()

- ~convOutput_PLOT3D ()
- void WriteGridData (int step, int myRank, int guide, double org[3], double pit[3], int sz[3])

  GRID ファイル出力
- template<class T1 , class T2 >
  void OutputPlot3D_xyz (int step, int rank, int guide, T1 *origin, T1 *pitch, int *size, T2 *x, T2 *y, T2 *z)

  xyz ファイルの出力 *(template* 関数*)*
- void WriteNgrid (FILE *fp, int ngrid)

  グリッド数の書き出し
- void WriteBlockData (FILE *fp, int id, int jd, int kd)

  ブロックデータの書き出し
- template<class T >
  bool WriteXYZData (FILE *fp, int id, int jd, int kd, int ngrid, T *x, T *y, T *z)

  grid データ出力
- template<class T >
  void WriteXYZ_FORMATTED (FILE *fp, int id, int jd, int kd, T *x)

  Formatted 出力
- FILE * OutputFile_Open (const std::string prefix, const unsigned step, const int id, const bool mio)

  出力ファイルをオープンする
- bool WriteHeaderRecord (int step, int dim, int d_type, int imax, int jmax, int kmax, double time, double *org, double *pit, const std::string prefix, FILE *fp)

  func データファイルのの *header* 部の書き込み
- bool WriteFieldData (FILE *fp, cio_Array *src, size_t dLen)

  Field Datat 出力
- template<class T1 , class T2 >
  void setScalarGridData (cio_TypeArray< T1 > *P, cio_TypeArray< T2 > *S)

  Scalar の格子点での値をセット
- template<class T1 , class T2 >
  void setVectorComponentGridData (cio_TypeArray< T1 > *P, cio_TypeArray< T2 > *S, int ivar)

  成分別*Vector* の格子点での値をセット
- template<class T >
  void VolumeDataDivide (cio_TypeArray< T > *P)

  内部の格子点のデータを *8* で割る
- void WriteFuncBlockData (FILE *fp, int id, int jd, int kd, int nvar)

  Function ブロックデータの書き出し
- void WriteFuncData (FILE *fp, cio_Array *p3src)

  func data の出力
- bool WriteDataMarker (int dmy, FILE *fp, bool out)

  マーカーの書き込み
- template<class T1 , class T2 >
  CONV_INLINE void OutputPlot3D_xyz (int step, int rank, int guide, T1 *origin, T1 *pitch, int *size, T2 *x, T2 *y, T2 *z)

  xyz ファイルの出力
- template<class T >
  CONV_INLINE bool WriteXYZData (FILE *fp, int id, int jd, int kd, int ngrid, T *x, T *y, T *z)

  grid データ出力
- template<class T >
  CONV_INLINE void WriteXYZ_FORMATTED (FILE *fp, int id, int jd, int kd, T *x)

  Formatted 出力
- template<class T1 , class T2 >
  CONV_INLINE void setScalarGridData (cio_TypeArray< T1 > *P, cio_TypeArray< T2 > *S)

  Scalar の格子点での値をセット

- template<class T1 , class T2 >
  CONV_INLINE void setVectorComponentGridData (cio_TypeArray< T1 > *P, cio_TypeArray< T2 > *S, int ivar)
  成分別*Vector*の格子点での値をセット
- template<class T >
  CONV_INLINE void VolumeDataDivide (cio_TypeArray< T > *P)
  内部の格子点のデータを重み付けで補正

Additional Inherited Members

4.8.1 説明

convOutput_PLOT3D.h の 23 行で定義されています。

4.8.2 コンストラクタとデストラクタ

4.8.2.1 convOutput_PLOT3D::convOutput_PLOT3D ( )

コンストラクタ

convOutput_PLOT3D.C の 22 行で定義されています。

```
23 {
24
25
26 }
```

4.8.2.2 convOutput_PLOT3D::~convOutput_PLOT3D ( )

デストラクタ

convOutput_PLOT3D.C の 30 行で定義されています。

```
31 {
32
33
34 }
```

4.8.3 関数

4.8.3.1 FILE * convOutput_PLOT3D::OutputFile_Open (const std::string *prefix*, const unsigned *step*, const int *id*, const bool *mio*) [virtual]

出力ファイルをオープンする

引数

in	*prefix*	ファイル接頭文字
in	*step*	ステップ数
in	*id*	ランク番号
in	*mio*	出力時の分割指定　true = local / false = gather(default)

convOutputを再定義しています。

convOutput_PLOT3D.C の 123 行で定義されています。

参照先 Exit, InputParam::Get_OutputDir(), InputParam::Get_OutputFilenameFormat(), InputParam::Get_OutputFormatType(), と convOutput::m_InputCntl.

```
128 {
129   FILE* fp;
130
131   //ファイル名の生成
132   std::string outfile;
133   int fnameformat = m_InputCntl->Get_OutputFilenameFormat();
134   outfile = m_InputCntl->Get_OutputDir() +"/"+
135             cio_DFI::Generate_FileName(prefix,
136                                        id,
137                                        step,
138                                        "func",
139                                        (CIO::E_CIO_OUTPUT_FNAME)fnameformat,
140                                        mio,
141                                        CIO::E_CIO_OFF);
142
143   printf("outfile : %s\n",outfile.c_str());
144
145   //出力ファイルオープン
146   // ascii
147   if( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
148     if( (fp = fopen(outfile.c_str(), "wa")) == NULL ) {
149       printf("\tCan't open file.(%s)\n",outfile.c_str());
150       Exit(0);
151     }
152   } else {
153   //binary
154     if( (fp = fopen(outfile.c_str(), "wb")) == NULL ) {
155       printf("\tCan't open file.(%s)\n",outfile.c_str());
156       Exit(0);
157     }
158   }
159
160   return fp;
161
162 }
```

4.8.3.2 template<class T1 , class T2 > CONV_INLINE void convOutput_PLOT3D::OutputPlot3D_xyz (int *step*, int *rank*, int *guide*, T1 * *origin*, T1 * *pitch*, int * *size*, T2 * *x*, T2 * *y*, T2 * *z*)

xyz ファイルの出力

引数

in	*step*	ステップ
in	*rank*	ランク
in	*guide*	ガイドセル数
in	*origin*	基点座標
in	*pitch*	ピッチ
in	*size*	セルサイズ
in	*x*	x 方向座標ワーク
in	*y*	y 方向座標ワーク
in	*z*	z 方向座標ワーク

conv_plot3d_inline.h の 42 行で定義されています。

参照先 _F_IDX_S3D, Exit, InputParam::Get_OutputDir(), InputParam::Get_OutputFilenameFormat(), InputParam::Get_OutputFormatType(), InputParam::Get_ThinOut(), convOutput::m_InputCntl, WriteBlockData(), WriteNgrid(), と WriteXYZData().

```
51 {
52   //value
53   int ngrid=1;
54   int ix = size[0];
55   int jx = size[1];
56   int kx = size[2];
57   int gd = guide;
58   int gc_out = 0;//plot3d は常にガイドセルを出力しない
59
60   //間引き数の取得
61   int thin_count = m_InputCntl->Get_ThinOut();
62
63   int *iblank=NULL;//dummy
64   int id,jd,kd;//出力サイズ
65   id=size[0]+1;//+2*gc_out
66   jd=size[1]+1;//+2*gc_out
```

```
67   kd=size[2]+1;//+2*gc_out
68
69   //間引きのための処理
70   int irest=(id-1)%thin_count;
71   int jrest=(jd-1)%thin_count;
72   int krest=(kd-1)%thin_count;
73   id=(id-1)/thin_count;
74   jd=(jd-1)/thin_count;
75   kd=(kd-1)/thin_count;
76   id=id+1;
77   jd=jd+1;
78   kd=kd+1;
79   if(irest!=0) id=id+1;
80   if(jrest!=0) jd=jd+1;
81   if(krest!=0) kd=kd+1;
82
83   // ガイドセル出力があった場合オリジナルポイントを調整しておく
84   T2 l_org[3], l_pit[3];
85   for (int i=0; i<3; i++)
86   {
87     l_org[i] = (T2)origin[i] + (T2)pitch[i]*(T2)gd;
88     l_pit[i] = (T2)pitch[i];
89   }
90
91   // 出力ファイル名
92   std::string tmp;
93   std::string t_prefix="PLOT3DoutputGrid";
94   int fnameformat = m_InputCntl->Get_OutputFilenameFormat();
95   tmp = m_InputCntl->Get_OutputDir() +"/"+
96         cio_DFI::Generate_FileName(t_prefix,
97                                    rank,
98                                    step,
99                                    "xyz",
100                                    (CIO::E_CIO_OUTPUT_FNAME)fnameformat,
101                                    true,
102                                    CIO::E_CIO_OFF);
103
104  //open file
105  FILE*fp;
106  if( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
107    if( (fp = fopen(tmp.c_str(), "wa")) == NULL ) {
108      printf("\tCan't open file.(%s)\n",tmp.c_str());
109      Exit(0);
110    }
111  } else {
112    if( (fp = fopen(tmp.c_str(), "wb")) == NULL ) {
113      printf("\tCan't open file.(%s)\n",tmp.c_str());
114      Exit(0);
115    }
116  }
117
118  //write block data
119  WriteNgrid(fp,ngrid);//if multi grid
120  WriteBlockData(fp,id,jd,kd);
121
122  for(int k=0;k<kd;k++){
123  for(int j=0;j<jd;j++){
124  for(int i=0;i<id;i++){
125    size_t ip = _F_IDX_S3D(i+1, j+1, k+1, id, jd, kd, 0);
126    x[ip]=l_org[0]+(T2)thin_count*l_pit[0]*(T2)i;//-pitch[0]*(float)gc_out;
127    y[ip]=l_org[1]+(T2)thin_count*l_pit[1]*(T2)j;//-pitch[1]*(float)gc_out;
128    z[ip]=l_org[2]+(T2)thin_count*l_pit[2]*(T2)k;//-pitch[2]*(float)gc_out;
129  }}}
130
131  //x direction modify
132  if(irest!=0 && (id-2)>=0 ){
133    for(int k=0;k<kd;k++){
134    for(int j=0;j<jd;j++){
135      size_t ip = _F_IDX_S3D(id, j+1, k+1, id, jd, kd, 0);
136      x[ip]=l_org[0]+(T2)thin_count*l_pit[0]*(T2)(id-2)+(T2)irest*l_pit[0];//-pitch[0]*(float)gc_out;
137    }}
138  }
139
140  //y direction modify
141  if(jrest!=0 && (jd-2)>=0 ){
142    for(int k=0;k<kd;k++){
143    for(int i=0;i<id;i++){
144      size_t ip = _F_IDX_S3D(i+1, jd, k+1, id, jd, kd, 0);
145      y[ip]=l_org[1]+(T2)thin_count*l_pit[1]*(T2)(jd-2)+(T2)jrest*l_pit[1];//-pitch[1]*(float)gc_out;
146    }}
147  }
148
149  //z direction modify
150  if(krest!=0 && (kd-2)>=0 ){
151    for(int j=0;j<jd;j++){
152    for(int i=0;i<id;i++){
153      size_t ip = _F_IDX_S3D(i+1, j+1, kd, id, jd, kd, 0);
```

```
154       z[ip]=l_org[2]+(T2)thin_count*l_pit[2]*(T2)(kd-2)+(T2)krest*l_pit[2];//-pitch[2]*(float)gc_out;
155     }}
156   }
157
158   //z direction modify
159   if(krest!=0){
160     for(int k=kd-1;k<kd;k++){
161     for(int j=0;j<jd;j++){
162     for(int i=0;i<id;i++){
163       size_t ip = _F_IDX_S3D(i+1, j+1, k+1, id, jd, kd, 0);
164       z[ip]=l_org[2]+(T2)krest*l_pit[2]*(T2)k;//-pitch[2]*(float)gc_out;
165     }}}
166   }
167
168   //write
169   if(!WriteXYZData(fp, id, jd, kd, ngrid, x, y, z)) printf("\terror WriteXYZData\n");
170
171   //close file
172   fclose(fp);
173
174 }
```

4.8.3.3 template<class T1 , class T2 > void convOutput_PLOT3D::OutputPlot3D_xyz (int *step*, int *rank*, int *guide*, T1 * *origin*, T1 * *pitch*, int * *size*, T2 * *x*, T2 * *y*, T2 * *z*)

xyz ファイルの出力 (template 関数)

引数

in	*step*	ステップ
in	*rank*	ランク
in	*guide*	ガイドセル数
in	*origin*	基点座標
in	*pitch*	ピッチ
in	*size*	ボクセルサイズ
in	*x*	x 方向座標ワーク
in	*y*	y 方向座標ワーク
in	*z*	z 方向座標ワーク

参照元 WriteGridData().

4.8.3.4 template<class T1 , class T2 > void convOutput_PLOT3D::setScalarGridData (cio_TypeArray< T1 > * *P*, cio_TypeArray< T2 > * *S*)

Scalar の格子点での値をセット

引数

out	*P*	格子点データ
in	*S*	セル中心 data

4.8.3.5 template<class T1 , class T2 > CONV_INLINE void convOutput_PLOT3D::setScalarGridData (cio_TypeArray< T1 > * *P*, cio_TypeArray< T2 > * *S*)

Scalar の格子点での値をセット

引数

out	*P*	格子点 data

in	*S*	セル中心 data

<0,0,0>

<1,0,0>

<1,0,1>

<0,0,1>

<0,1,0>

<1,1,0>

<1,1,1>

<0,1,1>

conv_plot3d_inline.h の 475 行で定義されています。

参照先 VolumeDataDivide().

```
478 {
479
480   T2* data = S->getData();
481   const int* size = S->getArraySizeInt();
482
483   int ix = size[0];
484   int jx = size[1];
485   int kx = size[2];
486
487   //size_t mip;
488   //float ddd;
489   //int i,j,k;
490
491   T1* d    = P->getData();
492   const int* Psz = P->getArraySizeInt();
493   int id = Psz[0];
494   int jd = Psz[1];
495   int kd = Psz[2];
496
497   size_t dsize = (size_t)(id*jd*kd);
498   for (size_t l=0; l<dsize; l++) d[l]=0.0;
499
500 /*
501   for (int km=1-gc_out; km<=kx+gc_out; km++) {
502   for (int jm=1-gc_out; jm<=jx+gc_out; jm++) {
503   for (int im=1-gc_out; im<=ix+gc_out; im++) {
504 */
505   for (int km=0; km<kx; km++) {
506   for (int jm=0; jm<jx; jm++) {
507   for (int im=0; im<ix; im++) {
508 /*
509     mip = _F_IDX_S3D(im, jm, km, ix, jx, kx, gd);
510     ddd=(T1)data[mip];
511     i=im-1+gc_out;
512     j=jm-1+gc_out;
513     k=km-1+gc_out;
514     size_t ip1 = _F_IDX_S3D(i+1, j+1, k+1, id, jd, kd, 0);
515     size_t ip2 = _F_IDX_S3D(i+2, j+1, k+1, id, jd, kd, 0);
516     size_t ip3 = _F_IDX_S3D(i+2, j+2, k+1, id, jd, kd, 0);
517     size_t ip4 = _F_IDX_S3D(i+1, j+2, k+1, id, jd, kd, 0);
518     size_t ip5 = _F_IDX_S3D(i+1, j+1, k+2, id, jd, kd, 0);
519     size_t ip6 = _F_IDX_S3D(i+2, j+1, k+2, id, jd, kd, 0);
520     size_t ip7 = _F_IDX_S3D(i+2, j+2, k+2, id, jd, kd, 0);
521     size_t ip8 = _F_IDX_S3D(i+1, j+2, k+2, id, jd, kd, 0);
522     d[ip1]=d[ip1]+ddd;
523     d[ip2]=d[ip2]+ddd;
524     d[ip3]=d[ip3]+ddd;
525     d[ip4]=d[ip4]+ddd;
526     d[ip5]=d[ip5]+ddd;
527     d[ip6]=d[ip6]+ddd;
528     d[ip7]=d[ip7]+ddd;
529     d[ip8]=d[ip8]+ddd;
530 */
531     P->val(im  ,jm  ,km  ) = P->val(im  ,jm  ,km  )+S->val(im,jm,km);
532     P->val(im+1,jm  ,km  ) = P->val(im+1,jm  ,km  )+S->val(im,jm,km);
533     P->val(im+1,jm  ,km+1) = P->val(im+1,jm  ,km+1)+S->val(im,jm,km);
534     P->val(im  ,jm  ,km+1) = P->val(im  ,jm  ,km+1)+S->val(im,jm,km);
535     P->val(im  ,jm+1,km  ) = P->val(im  ,jm+1,km  )+S->val(im,jm,km);
536     P->val(im+1,jm+1,km  ) = P->val(im+1,jm+1,km  )+S->val(im,jm,km);
537     P->val(im+1,jm+1,km+1) = P->val(im+1,jm+1,km+1)+S->val(im,jm,km);
538     P->val(im  ,jm+1,km+1) = P->val(im  ,jm+1,km+1)+S->val(im,jm,km);
539   }}}
540
```

```
541   //内部の格子点のデータの補正
542   VolumeDataDivide(P);
543
544 /*
545   //内部の格子点のデータを 8 で割る
546   //VolumeDataDivideBy8(d, id, jd, kd);
547   VolumeDataDivideBy8(P);
548
549   //面上の格子点のデータを 4 で割る
550   //FaceDataDivideBy4(d, id, jd, kd);
551   FaceDataDivideBy4(P);
552
553   //辺上の格子点のデータを 2 で割る
554   //LineDataDivideBy2(d, id, jd, kd);
555   LineDataDivideBy2(P);
556 */
557
558 };
```

4.8.3.6 template<class T1 , class T2 > void convOutput_PLOT3D::setVectorComponentGridData (cio_TypeArray< T1 > * *P*, cio_TypeArray< T2 > * *S*, int *ivar*)

成分別Vector の格子点での値をセット

引数

out	*P*	格子点 data
in	*S*	セル中心 data
in	*ivar*	ベクトル成分 =0:x =1:y =2:z

4.8.3.7 template<class T1 , class T2 > CONV_INLINE void convOutput_PLOT3D::setVectorComponentGridData (cio_TypeArray< T1 > * *P*, cio_TypeArray< T2 > * *S*, int *ivar*)

成分別Vector の格子点での値をセット

引数

out	*P*	格子点 data
in	*S*	セル中心 data
in	*ivar*	ベクトル成分 =0:x =1:y =2:z

<0,0,0>

<1,0,0>

<1,0,1>

<0,0,1>

<0,1,0>

<1,1,0>

<1,1,1>

<0,1,1>

conv_plot3d_inline.h の 569 行で定義されています。

参照先 VolumeDataDivide().

```
573 {
574
575   T2* data = S->getData();
576   const int* size = S->getArraySizeInt();
577
578   int ix = size[0];
579   int jx = size[1];
580   int kx = size[2];
581   //int gd = S->getGcInt();
582
583   size_t mip;
```

```
584   T1 ddd;
585   int i,j,k;
586
587   T1* d    = P->getData();
588   const int* Psz = P->getArraySizeInt();
589   int id = Psz[0];
590   int jd = Psz[1];
591   int kd = Psz[2];
592
593   size_t dsize = (size_t)(id*jd*kd);
594   for (size_t l=0; l<dsize; l++) d[l]=0.0;
595
596 /*
597   for (int km=1-gc_out; km<=kx+gc_out; km++) {
598   for (int jm=1-gc_out; jm<=jx+gc_out; jm++) {
599   for (int im=1-gc_out; im<=ix+gc_out; im++) {
600 */
601
602   for (int km=0; km<kx; km++) {
603   for (int jm=0; jm<jx; jm++) {
604   for (int im=0; im<ix; im++) {
605 /*
606     mip = _F_IDX_V3DEX(ivar, im, jm, km, ix, jx, kx, gd); //(3,i,j,k)
607     ddd=(T1)data[mip];
608
609     if( ivar==0 ) printf("mip : %d ddd %e\n",mip,ddd);
610
611     i=im-1+gc_out;
612     j=jm-1+gc_out;
613     k=km-1+gc_out;
614     size_t ip1 = _F_IDX_S3D(i+1, j+1, k+1, id, jd, kd, 0);
615     size_t ip2 = _F_IDX_S3D(i+2, j+1, k+1, id, jd, kd, 0);
616     size_t ip3 = _F_IDX_S3D(i+2, j+2, k+1, id, jd, kd, 0);
617     size_t ip4 = _F_IDX_S3D(i+1, j+2, k+1, id, jd, kd, 0);
618     size_t ip5 = _F_IDX_S3D(i+1, j+1, k+2, id, jd, kd, 0);
619     size_t ip6 = _F_IDX_S3D(i+2, j+1, k+2, id, jd, kd, 0);
620     size_t ip7 = _F_IDX_S3D(i+2, j+2, k+2, id, jd, kd, 0);
621     size_t ip8 = _F_IDX_S3D(i+1, j+2, k+2, id, jd, kd, 0);
622     d[ip1]=d[ip1]+ddd;
623     d[ip2]=d[ip2]+ddd;
624     d[ip3]=d[ip3]+ddd;
625     d[ip4]=d[ip4]+ddd;
626     d[ip5]=d[ip5]+ddd;
627     d[ip6]=d[ip6]+ddd;
628     d[ip7]=d[ip7]+ddd;
629     d[ip8]=d[ip8]+ddd;
630 */
631     P->val(im  ,jm  ,km  ) = P->val(im  ,jm  ,km  )+S->val(ivar,im,jm,km);
632     P->val(im+1,jm  ,km  ) = P->val(im+1,jm  ,km  )+S->val(ivar,im,jm,km);
633     P->val(im+1,jm  ,km+1) = P->val(im+1,jm  ,km+1)+S->val(ivar,im,jm,km);
634     P->val(im  ,jm  ,km+1) = P->val(im  ,jm  ,km+1)+S->val(ivar,im,jm,km);
635     P->val(im  ,jm+1,km  ) = P->val(im  ,jm+1,km  )+S->val(ivar,im,jm,km);
636     P->val(im+1,jm+1,km  ) = P->val(im+1,jm+1,km  )+S->val(ivar,im,jm,km);
637     P->val(im+1,jm+1,km+1) = P->val(im+1,jm+1,km+1)+S->val(ivar,im,jm,km);
638     P->val(im  ,jm+1,km+1) = P->val(im  ,jm+1,km+1)+S->val(ivar,im,jm,km);
639
640   }}}
641
642   //内部の格子点のデータの補正
643   VolumeDataDivide(P);
644
645 /*
646   //内部の格子点のデータを 8 で割る
647   //VolumeDataDivideBy8(d, id, jd, kd);
648   VolumeDataDivideBy8(P);
649
650   //面上の格子点のデータを 4 で割る
651   //FaceDataDivideBy4(d, id, jd, kd);
652   FaceDataDivideBy4(P);
653
654   //辺上の格子点のデータを 2 で割る
655   //LineDataDivideBy2(d, id, jd, kd);
656   LineDataDivideBy2(P);
657 */
658
659 };
```

4.8.3.8 template<class T > void convOutput_PLOT3D::VolumeDataDivide (cio_TypeArray< T > * *P*)

内部の格子点のデータを 8 で割る

引数

out	*P*	格子点 data 面上の格子点のデータを 4 で割る
out	*P*	格子点 data 辺上の格子点のデータを 2 で割る
out	*P*	格子点 data 内部の格子点のデータを重み付けで補正
out	*P*	格子点 data

参照元 setScalarGridData(), と setVectorComponentGridData().

4.8.3.9 template<class T > CONV_INLINE void convOutput_PLOT3D::VolumeDataDivide (cio_TypeArray< T > * *P*)

内部の格子点のデータを重み付けで補正

引数

out	*P*	格子点 data

conv_plot3d_inline.h の 667 行で定義されています。

```
668 {
669   int i,j,k,n;
670   const int* szP = P->getArraySizeInt();
671   int id = szP[0];
672   int jd = szP[1];
673   int kd = szP[2];
674
675   int ncomp = P->getNcompInt();
676
677   //printf("**** ncomp : %d\n",ncomp);
678
679   if( P->getArrayShape() == CIO::E_CIO_NIJK ) {
680
681     //I
682     for (k=0; k<kd;    k++){
683     for (j=0; j<jd;    j++){
684     for (i=1; i<id-1;  i++){
685     for (n=0; n<ncomp; n++){
686       P->val(n,i,j,k) = P->val(n,i,j,k)*0.5;
687     }}}}
688
689     //J
690     for (k=0; k<kd;    k++){
691     for (j=1; j<jd-1;  j++){
692     for (i=0; i<id;    i++){
693     for (n=0; n<ncomp; n++){
694       P->val(n,i,j,k) = P->val(n,i,j,k)*0.5;
695     }}}}
696
697     //K
698     for (k=1; k<kd-1;  k++){
699     for (j=0; j<jd;    j++){
700     for (i=0; i<id;    i++){
701     for (n=0; n<ncomp; n++){
702       P->val(n,i,j,k) = P->val(n,i,j,k)*0.5;
703     }}}}
704
705   } else {
706
707     //I
708     for (n=0; n<ncomp; n++){
709     for (k=0; k<kd;    k++){
710     for (j=0; j<jd;    j++){
711     for (i=1; i<id-1;  i++){
712       P->val(i,j,k,n) = P->val(i,j,k,n)*0.5;
713     }}}}
714
715     //J
716     for (n=0; n<ncomp; n++){
717     for (k=0; k<kd;    k++){
718     for (j=1; j<jd-1;  j++){
719     for (i=0; i<id;    i++){
720       P->val(i,j,k,n) = P->val(i,j,k,n)*0.5;
721     }}}}
722
723     //K
724     for (n=0; n<ncomp; n++){
725     for (k=1; k<kd-1;  k++){
726     for (j=0; j<jd;    j++){
727     for (i=0; i<id;    i++){
```

```
728       P->val(i,j,k,n) = P->val(i,j,k,n)*0.5;
729     }}}}
730
731   }
732 };
```

4.8.3.10 void convOutput_PLOT3D::WriteBlockData (FILE * *fp*, int *id*, int *jd*, int *kd*)

ブロックデータの書き出し

引数

in	*fp*	出力ファイルポインタ
in	*id*	i 方向のサイズ
in	*jd*	j 方向のサイズ
in	*kd*	k 方向のサイズ

convOutput_PLOT3D.C の 94 行で定義されています。

参照先 InputParam::Get_OutputFormatType(), convOutput::m_InputCntl, と WriteDataMarker().

参照元 OutputPlot3D_xyz().

```
95 {
96
97   switch (m_InputCntl->Get_OutputFormatType()) {
98     case CIO::E_CIO_OUTPUT_TYPE_FBINARY:
99       unsigned int dmy;
100       dmy = sizeof(int)*3;
101       WriteDataMarker(dmy,fp,true);
102       fwrite(&id, sizeof(int), 1, fp);
103       fwrite(&jd, sizeof(int), 1, fp);
104       fwrite(&kd, sizeof(int), 1, fp);
105       WriteDataMarker(dmy,fp,true);
106       break;
107     case CIO::E_CIO_OUTPUT_TYPE_ASCII:
108       fprintf(fp,"%5d%5d%5d\n",id,jd,kd);
109       break;
110     case CIO::E_CIO_OUTPUT_TYPE_BINARY:
111       fwrite(&id, sizeof(int), 1, fp);
112       fwrite(&jd, sizeof(int), 1, fp);
113       fwrite(&kd, sizeof(int), 1, fp);
114       break;
115     default:
116       break;
117     }
118
119 }
```

4.8.3.11 bool convOutput_PLOT3D::WriteDataMarker (int *dmy*, FILE * *fp*, bool *out*) [virtual]

マーカーの書き込み

引数

in	*dmy*	マーカー
in	*fp*	出力ファイルポインタ
in	*out*	Fortran マーカー出力フラグ

convOutputを再定義しています。

convOutput_PLOT3D.C の 368 行で定義されています。

参照先 InputParam::Get_OutputFormatType(), と convOutput::m_InputCntl.

参照元 WriteBlockData(), WriteFuncBlockData(), WriteFuncData(), WriteNgrid(), と WriteXYZData().

```
369 {
370   if( !out ) return true;
371   if( m_InputCntl->Get_OutputFormatType() != CIO::E_CIO_OUTPUT_TYPE_FBINARY ) return true;
```

```
372   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
373   return true;
374 }
```

4.8.3.12 bool convOutput_PLOT3D::WriteFieldData (FILE * *fp*, cio_Array * *src*, size_t *dLen*) [virtual]

Field Datat 出力

引数

in	*fp*	出力ファイルポインタ
in	*src*	出力データ配列ポインタ
in	*dLen*	出力データサイズ

convOutputを再定義しています。

convOutput_PLOT3D.C の 199 行で定義されています。

参照先 WriteFuncData().

```
202 {
203
204   const int* sz = src->getArraySizeInt();
205
206   cio_Array *out = cio_Array::instanceArray
207                    (src->getDataType(),
208                     src->getArrayShape(),
209                     (int *)sz,
210                     0,
211                     src->getNcomp());
212
213   int ret = src->copyArray(out);
214
215   WriteFuncData(fp, out);
216   delete out;
217
218 /*
219   //配列サイズの取得
220   const int *szS = src->getArraySizeInt();
221
222   //配列成分の取得
223   int ncomp = src->getNcomp();
224
225   //block data の出力
226   int szP[3];
227   szP[0]=szS[0]+1;
228   szP[1]=szS[1]+1;
229   szP[2]=szS[2]+1;
230
231   //出力バッファのインスタンス
232   cio_Array* p3src = cio_Array::instanceArray
233                    ( src->getDataType(),
234                      CIO::E_CIO_IJKN,
235                      szP,
236                      0,
237                      1);
238
239   int head_S[3],head_P[3];
240   head_S[0]=head_S[1]=head_S[2]=0;
241   head_P[0]=head_P[1]=head_P[2]=0;
242
243   src->setHeadIndex(head_S);
244   p3src->setHeadIndex(head_P);
245
246   unsigned int dmy;
247   size_t Len = (size_t)szP[0]*(size_t)szP[1]*(size_t)szP[2]*ncomp;
248   if(      p3src->getDataType() == CIO::E_CIO_FLOAT32) dmy = sizeof(float)*Len;
249   else if( p3src->getDataType() == CIO::E_CIO_FLOAT64) dmy = sizeof(double)*Len;
250   WriteDataMarker(dmy,fp,true);
251
252   //float
253   if( src->getDataType() == CIO::E_CIO_FLOAT32 ) {
254     cio_TypeArray<float> *S = dynamic_cast<cio_TypeArray<float>*>(src);
255     cio_TypeArray<float> *P = dynamic_cast<cio_TypeArray<float>*>(p3src);
256     //Scalar Data
257     if( ncomp == 1 ) {
258       setScalarGridData(P,S);
259       //func data の出力
260       WriteDataMarker(dmy,fp,true);
```

```
261     } else if( ncomp == 3 ) {
262     //Vector Data
263       for(int ixyz=0; ixyz<3; ixyz++) {
264         setVectorComponentGridData(P,S,ixyz);
265         //func data の出力
266         WriteFuncData(fp, p3src);
267         WriteDataMarker(dmy,fp,true);
268       }
269     }
270   } else {
271   //double
272     cio_TypeArray<double> *S = dynamic_cast<cio_TypeArray<double>*>(src);
273     cio_TypeArray<double> *P = dynamic_cast<cio_TypeArray<double>*>(p3src);
274     //Scalar Data
275     if( ncomp == 1 ) {
276       setScalarGridData(P,S);
277       //func data の出力
278       WriteFuncData(fp, p3src);
279       WriteDataMarker(dmy,fp,true);
280     } else if( ncomp == 3 ) {
281     //Vector Data
282       for(int ixyz=0; ixyz<3; ixyz++) {
283         setVectorComponentGridData(P,S,ixyz);
284         //func data の出力
285         WriteFuncData(fp, p3src);
286         WriteDataMarker(dmy,fp,true);
287       }
288     }
289   }
290
291   if( p3src ) delete p3src;
292 */
293
294   return true;
295
296 }
```

4.8.3.13 void convOutput_PLOT3D::WriteFuncBlockData (FILE * *fp*, int *id*, int *jd*, int *kd*, int *nvar*)

Function ブロックデータの書き出し

引数

in	*fp*	出力ファイルポインタ
in	*id*	i 方向のサイズ
in	*jd*	j 方向のサイズ
in	*kd*	k 方向のサイズ
in	*nvar*	出力項目数

convOutput_PLOT3D.C の 300 行で定義されています。

参照先 InputParam::Get_OutputFormatType(), convOutput::m_InputCntl, と WriteDataMarker().

参照元 WriteHeaderRecord().

```
301 {
302
303   switch (m_InputCntl->Get_OutputFormatType()) {
304     case CIO::E_CIO_OUTPUT_TYPE_FBINARY:
305       unsigned int dmy;
306       dmy = sizeof(int)*4;
307       WriteDataMarker(dmy,fp,true);
308       fwrite(&id, sizeof(int), 1, fp);
309       fwrite(&jd, sizeof(int), 1, fp);
310       fwrite(&kd, sizeof(int), 1, fp);
311       fwrite(&nvar, sizeof(int), 1, fp);
312       WriteDataMarker(dmy,fp,true);
313       break;
314     case CIO::E_CIO_OUTPUT_TYPE_ASCII:
315       fprintf(fp,"%5d%5d%5d%5d\n",id,jd,kd,nvar);
316       break;
317     case CIO::E_CIO_OUTPUT_TYPE_BINARY:
318       fwrite(&id, sizeof(int), 1, fp);
319       fwrite(&jd, sizeof(int), 1, fp);
320       fwrite(&kd, sizeof(int), 1, fp);
321       fwrite(&nvar, sizeof(int), 1, fp);
322       break;
323     default:
```

```
324       break;
325   }
326
327 }
```

4.8.3.14 void convOutput_PLOT3D::WriteFuncData (FILE * *fp*, cio_Array * *p3src*)

func data の出力

引数

in	*fp*	出力ファイルポインタ
in	*p3src*	plot3d func データ配列ポインタ

convOutput_PLOT3D.C の 330 行で定義されています。

参照先 InputParam::Get_OutputFormatType(), convOutput::m_InputCntl, と WriteDataMarker().

参照元 WriteFieldData().

```
331 {
332
333   const int* sz = p3src->getArraySizeInt();
334   size_t dLen = (size_t)sz[0]*(size_t)sz[1]*(size_t)sz[2]*p3src->getNcomp();
335
336   switch (m_InputCntl->Get_OutputFormatType()) {
337     case CIO::E_CIO_OUTPUT_TYPE_FBINARY:
338       unsigned int dmy;
339       if( p3src->getDataType() == CIO::E_CIO_FLOAT32 ) {
340         dmy = sizeof(float)*dLen;
341       } else {
342         dmy = sizeof(double)*dLen;
343       }
344       WriteDataMarker(dmy,fp,true);
345       p3src->writeBinary(fp);
346       WriteDataMarker(dmy,fp,true);
347       break;
348     case CIO::E_CIO_OUTPUT_TYPE_ASCII:
349       if( p3src->getDataType() == CIO::E_CIO_FLOAT32) {
350         float *data = (float*)p3src->getData();
351         for(int i=0; i<dLen; i++) fprintf(fp,"%15.6E\n",data[i]);
352       } else if( p3src->getDataType() == CIO::E_CIO_FLOAT64) {
353         double *data = (double*)p3src->getData();
354         for(int i=0; i<dLen; i++) fprintf(fp,"%15.6E\n",data[i]);
355       }
356       break;
357     case CIO::E_CIO_OUTPUT_TYPE_BINARY:
358       p3src->writeBinary(fp);
359       break;
360     default:
361       break;
362   }
363
364 }
```

4.8.3.15 void convOutput_PLOT3D::WriteGridData (int *step*, int *myRank*, int *guide*, double *org[3]*, double *pit[3]*, int *sz[3]*) [virtual]

GRID ファイル出力

引数

in	*step*	step 番号
in	*myRank*	ランク番号
in	*guide*	ガイドセル数
in	*org*	原点座標値

in	*pit*	ピッチ
in	*sz*	ボクセルサイズ

convOutputを再定義しています。

convOutput_PLOT3D.C の 38 行で定義されています。

参照先 InputParam::Get_OutputDataType(), convOutput::m_InputCntl, と OutputPlot3D_xyz().

```
45 {
46
47   //step 0 以外は出力しない
48   if( step != 0 ) return;
49
50   int id,jd,kd;
51   id=sz[0]+1;
52   jd=sz[1]+1;
53   kd=sz[2]+1;
54   size_t maxsize = (size_t)id*(size_t)jd*(size_t)kd*3;
55   size_t outsize = (size_t)id*(size_t)jd*(size_t)kd;
56
57   if( m_InputCntl->Get_OutputDataType() == CIO::E_CIO_FLOAT64 ) {
58     double* x = new double[maxsize];
59     OutputPlot3D_xyz(step, myRank, guide, org, pit, sz, &x[0], &x[outsize] ,
60                      &x[outsize*2] );
61   }else if( m_InputCntl->Get_OutputDataType() == CIO::E_CIO_FLOAT32 ) {
62     float* x = new float[maxsize];
63     OutputPlot3D_xyz(step, myRank, guide, org, pit, sz, &x[0], &x[outsize] ,
64                      &x[outsize*2] );
65   }
66
67 }
```

4.8.3.16 bool convOutput_PLOT3D::WriteHeaderRecord (int *step*, int *dim*, int *d_type*, int *imax*, int *jmax*, int *kmax*, double *time*, double * *org*, double * *pit*, const std::string *prefix*, FILE * *fp*) [virtual]

func データファイルのの header 部の書き込み

引数

in	*step*	ステップ数
in	*dim*	成分数
in	*d_type*	データ型タイプ
in	*imax*	x 方向ボクセルサイズ
in	*jmax*	y 方向ボクセルサイズ
in	*kmax*	z 方向ボクセルサイズ
in	*time*	時間
in	*org*	原点座標
in	*pit*	ピッチ
in	*prefix*	ファイル接頭文字
in	*fp*	出力ファイルポインタ

convOutputを再定義しています。

convOutput_PLOT3D.C の 166 行で定義されています。

参照先 WriteFuncBlockData(), と WriteNgrid().

```
178 {
179   if( !fp ) return false;
180
181   //ngird の初期化
182   int ngrid=1;
183
184   //ngrid の出力
185   WriteNgrid(fp,ngrid);
186
187   //block data の出力
188
189   WriteFuncBlockData(fp,imax+1,jmax+1,kmax+1,dim);
190
191   return true;
```

```
192
193 }
```

4.8.3.17 void convOutput_PLOT3D::WriteNgrid (FILE * *fp*, int *ngrid*)

グリッド数の書き出し

引数

in	*fp*	出力ファイルポインタ
in	*ngrid*	グリッド数

convOutput_PLOT3D.C の 71 行で定義されています。

参照先 InputParam::Get_OutputFormatType(), convOutput::m_InputCntl, と WriteDataMarker().

参照元 OutputPlot3D_xyz(), と WriteHeaderRecord().

```
72 {
73   switch (m_InputCntl->Get_OutputFormatType()) {
74     case CIO::E_CIO_OUTPUT_TYPE_FBINARY:
75       unsigned int dmy;
76       dmy = sizeof(int);
77       WriteDataMarker(dmy,fp,true);
78       fwrite(&ngrid, sizeof(int), 1, fp);
79       WriteDataMarker(dmy,fp,true);
80       break;
81     case CIO::E_CIO_OUTPUT_TYPE_ASCII:
82       fprintf(fp,"%5d\n",ngrid);
83       break;
84     case CIO::E_CIO_OUTPUT_TYPE_BINARY:
85       fwrite(&ngrid, sizeof(int), 1, fp);
86       break;
87     default:
88       break;
89     }
90 }
```

4.8.3.18 template<class T > void convOutput_PLOT3D::WriteXYZ_FORMATTED (FILE * *fp*, int *id*, int *jd*, int *kd*, T * *x*)

Formatted 出力

引数

in	*fp*	出力ファイルポインタ
in	*id*	i 方向サイズ
in	*jd*	j 方向のサイズ
in	*kd*	k 方向のサイズ
in	*x*	出力座標値配列

参照元 WriteXYZData().

4.8.3.19 template<class T > CONV_INLINE void convOutput_PLOT3D::WriteXYZ_FORMATTED (FILE * *fp*, int *id*, int *jd*, int *kd*, T * *x*)

Formatted 出力

引数

in	*fp*	出力ファイルポインタ

in	*id*	i 方向サイズ
in	*jd*	j 方向サイズ
in	*kd*	k 方向サイズ
in	*x*	出力座標値配列

conv_plot3d_inline.h の 249 行で定義されています。

```
254 {
255
256   int s12 =(size_t)id*(size_t)jd;
257   int ns12=s12/10;
258
259 /*
260   for(int k=0; k<kd; k++) {
261     //x-y 面の出力
262     for(int i=0; i<ns12; i++) {
263       for(int ii=0; ii<10; ii++) {
264         fprintf(fp,"%15.6E",x[(k*id*jd)+(i*10)+ii]);
265       }
266       fprintf(fp,"\n");
267     }
268     //余りの出力
269     if( s12%10 > 0 ) {
270       for(int i=0; i<s12%10; i++) fprintf(fp,"%15.6E",x[k*id*jd+(ns12*10)+i]);
271     }
272     fprintf(fp,"\n");
273   }
274 */
275    for(int i=0; i<id*jd*kd; i++) {
276      fprintf(fp,"%15.6E\n",x[i]);
277    }
278
279 }
```

4.8.3.20 template<class T > bool convOutput_PLOT3D::WriteXYZData (FILE * *fp*, int *id*, int *jd*, int *kd*, int *ngrid*, T * *x*, T * *y*, T * *z*)

grid データ出力

引数

in	*fp*	出力ファイルポインタ
in	*id*	i 方向サイズ
in	*jd*	j 方向のサイズ
in	*kd*	k 方向のサイズ
in	*ngrid*	1
in	*x*	x 座標値
in	*y*	y 座標値
in	*z*	z 座標値

参照元 OutputPlot3D_xyz().

4.8.3.21 template<class T > CONV_INLINE bool convOutput_PLOT3D::WriteXYZData (FILE * *fp*, int *id*, int *jd*, int *kd*, int *ngrid*, T * *x*, T * *y*, T * *z*)

grid データ出力

引数

in	*fp*	出力ファイルポインタ
in	*id*	i 方向サイズ
in	*jd*	j 方向のサイズ

in	*kd*	k 方向のサイズ
in	*ngrid*	1
in	*x*	x 座標値
in	*y*	y 座標値
in	*z*	z 座標値

conv_plot3d_inline.h の 190 行で定義されています。

参照先 InputParam::Get_OutputFormatType(), convOutput::m_InputCntl, WriteDataMarker(), と WriteXYZ_FORMATTED().

```
198 {
199
200   size_t sz = (size_t)id*(size_t)jd*(size_t)kd;
201   unsigned int dmy;
202
203   int s12 =(size_t)id*(size_t)jd;
204   int ns12=s12/10;
205
206   switch (m_InputCntl->Get_OutputFormatType()) {
207
208     //Fortran Binary 出力
209     case CIO::E_CIO_OUTPUT_TYPE_FBINARY:
210       dmy = sizeof(T)*sz*3;
211       WriteDataMarker(dmy,fp,true);
212       fwrite(x, sizeof(T), sz, fp);
213       fwrite(y, sizeof(T), sz, fp);
214       fwrite(z, sizeof(T), sz, fp);
215       WriteDataMarker(dmy,fp,true);
216       break;
217
218     //ascii 出力
219     case CIO::E_CIO_OUTPUT_TYPE_ASCII:
220       WriteXYZ_FORMATTED(fp, id, jd, kd, x);
221       WriteXYZ_FORMATTED(fp, id, jd, kd, y);
222       WriteXYZ_FORMATTED(fp, id, jd, kd, z);
223       break;
224
225     //C Binary 出力
226     case CIO::E_CIO_OUTPUT_TYPE_BINARY:
227       fwrite(x, sizeof(T), sz, fp);
228       fwrite(y, sizeof(T), sz, fp);
229       fwrite(z, sizeof(T), sz, fp);
230       break;
231     default:
232       return false;
233       break;
234   }
235   return true;
236 }
```

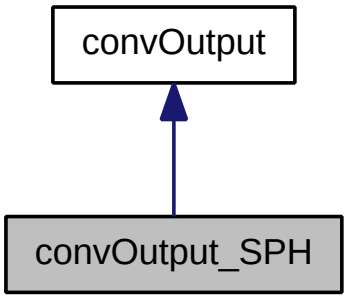
このクラスの説明は次のファイルから生成されました:

- convOutput_PLOT3D.h
- convOutput_PLOT3D.C
- conv_plot3d_inline.h

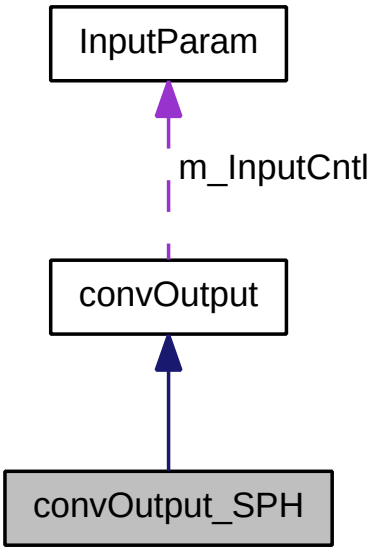
4.9 クラス convOutput_SPH

```
#include <convOutput_SPH.h>
```

convOutput_SPH に対する継承グラフ



convOutput_SPH のコラボレーション図



Public メソッド

- convOutput_SPH ()
- ~convOutput_SPH ()
- FILE * OutputFile_Open (const std::string prefix, const unsigned step, const int id, const bool mio)

  出力ファイルをオープンする
- bool WriteHeaderRecord (int step, int dim, int d_type, int imax, int jmax, int kmax, double time, double *org, double *pit, const std::string prefix, FILE *fp)

  sph **ファイルの** *header* **の書き込み**
- bool WriteDataMarker (int dmy, FILE *fp, bool out)

  マーカーの書き込み

Additional Inherited Members

4.9.1 説明

convOutput_SPH.h の 23 行で定義されています。

4.9.2 コンストラクタとデストラクタ

4.9.2.1 convOutput_SPH::convOutput_SPH ( )

コンストラクタ

convOutput_SPH.C の 22 行で定義されています。

```
23 {
24
25
26 }
```

4.9.2.2 convOutput_SPH::∼convOutput_SPH ( )

デストラクタ

convOutput_SPH.C の 30 行で定義されています。

```
31 {
32
33
34 }
```

4.9.3 関数

4.9.3.1 FILE ∗ convOutput_SPH::OutputFile_Open (const std::string *prefix,* const unsigned *step,* const int *id,* const bool *mio*) [virtual]

出力ファイルをオープンする

引数

in	*prefix*	ファイル接頭文字
in	*step*	ステップ数
in	*id*	ランク番号
in	*mio*	出力時の分割指定　true = local / false = gather(default)

convOutputを再定義しています。

convOutput_SPH.C の 38 行で定義されています。

参照先 Exit, InputParam::Get_OutputDir(), InputParam::Get_OutputFilenameFormat(), と convOutput::m_InputCntl.

```
43 {
44   FILE* fp;
45
46   //ファイル名の生成
47   std::string outfile;
48   int fnameformat = m_InputCntl->Get_OutputFilenameFormat();
49   outfile = m_InputCntl->Get_OutputDir()+"/"+
50             cio_DFI::Generate_FileName(prefix,
51                                        id,
52                                        step,
53                                        "sph",
54                                        (CIO::E_CIO_OUTPUT_FNAME)fnameformat,
55                                        mio,
56                                        CIO::E_CIO_OFF);
57
58   printf("outfile : %s\n",outfile.c_str());
59
60   //ファイルオープン
61   if( (fp = fopen(outfile.c_str(), "wb")) == NULL ) {
62     printf("\tCan't open file.(%s)\n",outfile.c_str());
63     Exit(0);
64   }
65
66   return fp;
```

```
67
68 }
```

4.9.3.2 bool convOutput_SPH::WriteDataMarker (int *dmy*, FILE * *fp*, bool *out*) [virtual]

マーカーの書き込み

引数

in	*dmy*	マーカー
in	*fp*	出力ファイルポインタ
in	*out*	plot3d 用

convOutputを再定義しています。

convOutput_SPH.C の 176 行で定義されています。

```
177 {
178   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
179   return true;
180 }
```

4.9.3.3 bool convOutput_SPH::WriteHeaderRecord (int *step*, int *dim*, int *d_type*, int *imax*, int *jmax*, int *kmax*, double *time*, double * *org*, double * *pit*, const std::string *prefix*, FILE * *fp*) [virtual]

sph ファイルの header の書き込み

引数

in	*step*	ステップ数
in	*dim*	成分数
in	*d_type*	データ型タイプ
in	*imax*	x 方向ボクセルサイズ
in	*jmax*	y 方向ボクセルサイズ
in	*kmax*	z 方向ボクセルサイズ
in	*time*	時間
in	*org*	原点座標
in	*pit*	ピッチ
in	*prefix*	ファイル接頭文字
in	*fp*	出力ファイルポインタ

convOutputを再定義しています。

convOutput_SPH.C の 72 行で定義されています。

参照先 SPH_DOUBLE, SPH_FLOAT, SPH_SCALAR, と SPH_VECTOR.

```
84 {
85   if( !fp ) return false;
86
87   unsigned int dmy;
88
89   //出力データ種別フラグの設定
90   int sv_type;
91   if( dim == 1 ) sv_type = SPH_SCALAR;
92   else if( dim == 3 ) sv_type = SPH_VECTOR;
93
94   //出力データ型フラグの設定
95   int d_type;
96   if( out_type == CIO::E_CIO_FLOAT32 ) d_type = SPH_FLOAT;
97   else if( out_type == CIO::E_CIO_FLOAT64 ) d_type = SPH_DOUBLE;
98
99   dmy = 2 * sizeof(int);
100   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
101   if( fwrite(&sv_type, sizeof(int), 1, fp) != 1 ) return false;
102   if( fwrite(&d_type, sizeof(int), 1, fp) != 1 ) return false;
```

```
103   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
104
105   if( d_type == SPH_FLOAT ) {
106     dmy = 3 * sizeof(int);
107   } else if( d_type == SPH_DOUBLE ) {
108     dmy = 3 * sizeof(long long);
109   }
110
111   //dmy = 3 * sizeof(long long);
112   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
113   if( d_type == SPH_FLOAT ) {
114     if( fwrite(&imax, sizeof(int), 1, fp) != 1 ) return false;
115     if( fwrite(&jmax, sizeof(int), 1, fp) != 1 ) return false;
116     if( fwrite(&kmax, sizeof(int), 1, fp) != 1 ) return false;
117   } else {
118     if( fwrite(&imax, sizeof(long long), 1, fp) != 1 ) return false;
119     if( fwrite(&jmax, sizeof(long long), 1, fp) != 1 ) return false;
120     if( fwrite(&kmax, sizeof(long long), 1, fp) != 1 ) return false;
121   }
122   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
123
124   if( d_type == SPH_FLOAT ) {
125     dmy = 3 * sizeof(float);
126   } else {
127     dmy = 3 * sizeof(double);
128   }
129   //dmy = 3 * sizeof(double);
130   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
131   if( d_type == SPH_FLOAT ) {
132     float tmp[3];
133     tmp[0] = (float)org[0];
134     tmp[1] = (float)org[1];
135     tmp[2] = (float)org[2];
136     if( fwrite(tmp, sizeof(float), 3, fp) != 3 ) return false;
137   } else {
138     if( fwrite(org, sizeof(double), 3, fp) != 3 ) return false;
139   }
140   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
141
142   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
143   if( d_type == SPH_FLOAT ) {
144     float tmp[3];
145     tmp[0] = (float)pit[0];
146     tmp[1] = (float)pit[1];
147     tmp[2] = (float)pit[2];
148     if( fwrite(tmp, sizeof(float), 3, fp) != 3 ) return false;
149   } else {
150     if( fwrite(pit, sizeof(double), 3, fp) != 3 ) return false;
151   }
152   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
153
154   if( d_type == SPH_FLOAT ) {
155     dmy = sizeof(int) + sizeof(float);
156   } else {
157     dmy = sizeof(long long) + sizeof(double);
158   }
159   //dmy = sizeof(long long) + sizeof(double);
160   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
161   if( d_type == SPH_FLOAT ) {
162     float ftmp = (float)time;
163     if( fwrite(&step, sizeof(int), 1, fp) != 1 ) return false;
164     if( fwrite(&ftmp, sizeof(float), 1, fp) != 1 ) return false;
165   } else {
166     if( fwrite(&step, sizeof(long long), 1, fp) != 1 ) return false;
167     if( fwrite(&time, sizeof(double), 1, fp) != 1 ) return false;
168   }
169   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
170
171   return true;
172 }
```

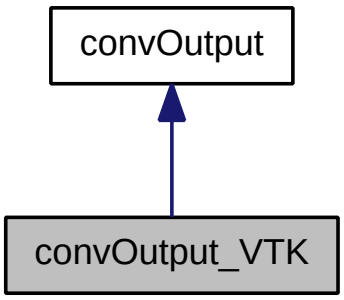
このクラスの説明は次のファイルから生成されました:

- convOutput_SPH.h
- convOutput_SPH.C

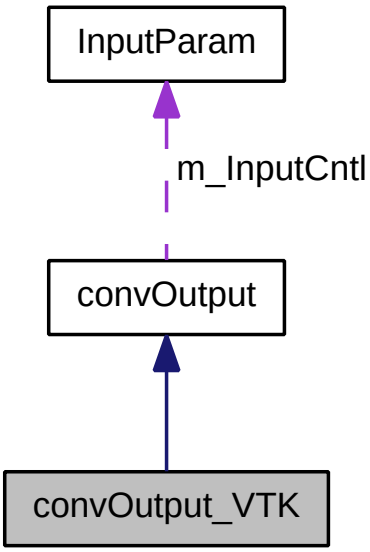
4.10 クラス convOutput_VTK

```
#include <convOutput_VTK.h>
```

convOutput_VTK に対する継承グラフ



convOutput_VTK のコラボレーション図



Public メソッド

- convOutput_VTK ()
- ~convOutput_VTK ()
- FILE * OutputFile_Open (const std::string prefix, const unsigned step, const int id, const bool mio)

  出力ファイルをオープンする
- bool WriteHeaderRecord (int step, int dim, int d_type, int imax, int jmax, int kmax, double time, double *org, double *pit, const std::string prefix, FILE *fp)

  vtk **ファイルの** *header* **の書き込み**
- bool WriteFieldData (FILE *fp, cio_Array *src, size_t dLen)

  Field Datat **出力**
- bool WriteDataMarker (int dmy, FILE *fp, bool out)

  マーカーの書き込み
- void OutputFile_Close (FILE *fp)

  出力ファイルをクローズする

Additional Inherited Members

4.10.1 説明

convOutput_VTK.h の 23 行で定義されています。

4.10.2 コンストラクタとデストラクタ

4.10.2.1 convOutput_VTK::convOutput_VTK ( )

コンストラクタ

convOutput_VTK.C の 22 行で定義されています。

```
23 {
24
25
26 }
```

4.10.2.2 convOutput_VTK::~convOutput_VTK ( )

デストラクタ

convOutput_VTK.C の 30 行で定義されています。

```
31 {
32
33
34 }
```

4.10.3 関数

4.10.3.1 void convOutput_VTK::OutputFile_Close (FILE * *fp*) [virtual]

出力ファイルをクローズする

引数

in	*fp*	ファイルポインタ

convOutputを再定義しています。

convOutput_VTK.C の 203 行で定義されています。

```
204 {
205   fprintf( fp, "\n" );
206   fclose(fp);
207 }
```

4.10.3.2 FILE * convOutput_VTK::OutputFile_Open (const std::string *prefix*, const unsigned *step*, const int *id*, const bool *mio*) [virtual]

出力ファイルをオープンする

引数

in	*prefix*	ファイル接頭文字

in	*step*	ステップ数
in	*id*	ランク番号
in	*mio*	出力時の分割指定　true = local / false = gather(default)

convOutputを再定義しています。

convOutput_VTK.C の 38 行で定義されています。

参照先 Exit, InputParam::Get_OutputDir(), InputParam::Get_OutputFilenameFormat(), InputParam::Get_OutputFormatType(), と convOutput::m_InputCntl.

```
43 {
44   FILE* fp;
45
46   //ファイル名の生成
47   std::string outfile;
48   int fnameformat = m_InputCntl->Get_OutputFilenameFormat();
49   outfile = m_InputCntl->Get_OutputDir()+ "/"+
50             cio_DFI::Generate_FileName(prefix,
51                                        id,
52                                        step,
53                                        "vtk",
54                                        (CIO::E_CIO_OUTPUT_FNAME)fnameformat,
55                                        mio,
56                                        CIO::E_CIO_OFF);
57
58   //ファイルオープン
59   if( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_BINARY ) {
60     if( (fp = fopen(outfile.c_str(), "w")) == NULL ) {
61       printf("\tCan't open file.(%s)\n",outfile.c_str());
62       Exit(0);
63     }
64   } else if( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
65     if( (fp = fopen(outfile.c_str(), "wa")) == NULL ) {
66       printf("\tCan't open file.(%s)\n",outfile.c_str());
67       Exit(0);
68     }
69   }
70
71   return fp;
72
73 }
```

4.10.3.3 bool convOutput_VTK::WriteDataMarker (int *dmy*, FILE * *fp*, bool *out*) [virtual]

マーカーの書き込み

引数

in	*dmy*	マーカー
in	*fp*	出力ファイルポインタ
in	*out*	出力フラグ

convOutputを再定義しています。

convOutput_VTK.C の 194 行で定義されています。

```
195 {
196   if( !out ) return true;
197   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
198   return true;
199 }
```

4.10.3.4 bool convOutput_VTK::WriteFieldData (FILE * *fp*, cio_Array * *src*, size_t *dLen*) [virtual]

Field Datat 出力

引数

in	*fp*	出力ファイルポインタ
in	*src*	出力データ配列ポインタ
in	*dLen*	出力データサイズ

convOutputを再定義しています。

convOutput_VTK.C の 150 行で定義されています。

参照先 Exit, InputParam::Get_OutputFormatType(), と convOutput::m_InputCntl.

```
151 {
152
153   const int* sz = src->getArraySizeInt();
154   cio_Array *out = cio_Array::instanceArray
155                    (src->getDataType(),
156                     src->getArrayShape(),
157                     (int *)sz,
158                     0,
159                     src->getNcomp());
160
161   int ret = src->copyArray(out);
162
163   //バイナリー出力のとき
164   if( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_BINARY ) {
165
166     //出力タイプが float
167     if( out->getDataType() == CIO::E_CIO_FLOAT32 ) {
168       float *data = (float*)out->getData();
169       BSWAPVEC(data,dLen);
170       fwrite( data, sizeof(float), dLen, fp );
171     } else if( out->getDataType() == CIO::E_CIO_FLOAT64 ) {
172       double *data = (double*)out->getData();
173       DBSWAPVEC(data,dLen);
174       fwrite( data, sizeof(double), dLen, fp );
175     }
176
177   //アスキー出力のとき
178   } else if ( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
179
180     if( out->writeAscii(fp) != dLen ) {
181       delete out;
182       Exit(0);
183     }
184
185   }
186
187   delete out;
188   return true;
189 }
```

4.10.3.5 bool convOutput_VTK::WriteHeaderRecord (int *step*, int *dim*, int *d_type*, int *imax*, int *jmax*, int *kmax*, double *time*, double * *org*, double * *pit*, const std::string *prefix*, FILE * *fp*) [virtual]

vtk ファイルの header の書き込み

引数

in	*step*	ステップ数
in	*dim*	成分数
in	*d_type*	データ型タイプ
in	*imax*	x 方向ボクセルサイズ
in	*jmax*	y 方向ボクセルサイズ
in	*kmax*	z 方向ボクセルサイズ
in	*time*	時間
in	*org*	原点座標

in	*pit*	ピッチ
in	*prefix*	ファイル接頭文字
in	*fp*	出力ファイルポインタ

convOutputを再定義しています。

convOutput_VTK.C の 77 行で定義されています。

参照先 InputParam::Get_OutputFormatType(), と convOutput::m_InputCntl.

```
89 {
90   if( !fp ) return false;
91
92   fprintf( fp, "# vtk DataFile Version 2.0\n" );
93   fprintf( fp, "step=%d,time=%g\n", step, time );
94
95   if( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_BINARY ) {
96     fprintf( fp, "BINARY\n" );
97   } else if( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
98     fprintf( fp, "ASCII\n" );
99   }
100
101   fprintf( fp, "DATASET STRUCTURED_POINTS\n" );
102
103   fprintf( fp, "DIMENSIONS %d %d %d\n", imax+1, jmax+1, kmax+1 );
104
105   double t_org[3];
106   t_org[0]=org[0]-(pit[0]*0.5);
107   t_org[1]=org[1]-(pit[1]*0.5);
108   t_org[2]=org[2]-(pit[2]*0.5);
109   fprintf( fp, "ORIGIN %e %e %e\n", t_org[0], t_org[1], t_org[2] );
110
111   fprintf( fp, "ASPECT_RATIO %e %e %e\n", pit[0], pit[1], pit[2] );
112
113   //int nw = imax*jmax*kmax;
114   //fprintf( fp, "CELL_DATA %d\n", nw );
115   int nw = (imax+1)*(jmax+1)*(kmax+1);
116   fprintf( fp, "POINT_DATA %d\n", nw );
117
118   if( dim == 1 )
119   {
120     if( d_type == CIO::E_CIO_FLOAT32 ) {
121       fprintf( fp, "SCALARS %s float\n", prefix.c_str() );
122     } else if( d_type == CIO::E_CIO_FLOAT64 ) {
123       fprintf( fp, "SCALARS %s double\n", prefix.c_str() );
124     }
125     fprintf( fp, "LOOKUP_TABLE default\n" );
126   }
127   else if( dim == 3 )
128   {
129     if( d_type == CIO::E_CIO_FLOAT32 ) {
130       fprintf( fp, "VECTORS %s float\n", prefix.c_str() );
131     } else if( d_type == CIO::E_CIO_FLOAT64 ) {
132       fprintf( fp, "VECTORS %s double\n", prefix.c_str() );
133     }
134   }
135   else
136   {
137     fprintf( fp, "FIELD %s 1\n", prefix.c_str() );
138     if( d_type == CIO::E_CIO_FLOAT32 ) {
139       fprintf( fp, "%s %d %d float\n", prefix.c_str(), dim, nw );
140     } else if( d_type == CIO::E_CIO_FLOAT64 ) {
141       fprintf( fp, "%s %d %d doubel\n", prefix.c_str(), dim, nw );
142     }
143   }
144
145   return true;
146 }
```

このクラスの説明は次のファイルから生成されました:

- convOutput_VTK.h
- convOutput_VTK.C

4.11 構造体 CONV::dfi_MinMax

```
#include <conv.h>
```

Public メソッド

- dfi_MinMax (int nstep, int ncomp)
- ~dfi_MinMax ()

Public 変数

- cio_DFI * dfi
- double * Min
- double * Max

4.11.1 説明

conv.h の 61 行で定義されています。

4.11.2 コンストラクタとデストラクタ

4.11.2.1 CONV::dfi_MinMax::dfi_MinMax (int *nstep*, int *ncomp*) [inline]

conv.h の 65 行で定義されています。

```
65                                  {
66       Min = new double[ncomp*nstep];
67       Max = new double[ncomp*nstep];
68       for(int i=0; i<ncomp*nstep; i++ ) {
69         Min[i]=DBL_MAX;
70         Max[i]=-DBL_MAX;
71       }
72     };
```

4.11.2.2 CONV::dfi_MinMax::~dfi_MinMax ( ) [inline]

conv.h の 73 行で定義されています。

```
73                 {
74       if( Min ) delete [] Min;
75       if( Max ) delete [] Max;
76     };
```

4.11.3 変数

4.11.3.1 cio_DFI* CONV::dfi_MinMax::dfi

conv.h の 62 行で定義されています。

参照元 convMxM::exec(), と convMx1::exec().

4.11.3.2 double* CONV::dfi_MinMax::Max

conv.h の 64 行で定義されています。

4.11.3.3 double* CONV::dfi_MinMax::Min

conv.h の 63 行で定義されています。

この構造体の説明は次のファイルから生成されました:

- conv.h

4.12 クラス InputParam

`#include <InputParam.h>`

Public メソッド

- InputParam ()
- ~InputParam ()
- bool importCPM (cpm_ParaManager *paraMngr)
  CPM のポインタをコピー
- bool Read (std::string input_file_name)
  入力ファイルの読み込み
- vector< std::string > Get_IndfiNameList ()
  input dfi ファイル名リストの取り出し
- vector< std::string > Get_OutdfiNameList ()
  output dfi ファイル名リストの取り出し
- vector< std::string > Get_OutprocNameList ()
  output proc ファイル名リストの取り出し
- bool Set_OutprocNameList (vector< std::string > out_proc_name)
  output proc ファイル名リストをセットする
- int Get_ConvType ()
  コンバートタイプの取り出し
- int * Get_OutputDivision ()
  出力分割数の取り出し
- int Get_OutputFormat ()
  出力ファイルフォーマットの取り出し
- std::string Get_OutputFormat_string ()
- int Get_OutputDataType ()
  出力タイプの取り出し
- int Get_OutputFormatType ()
  出力形式の取り出し（*ascii,binary,FortranBinary)*
- std::string Get_OutputDir ()
  出力先ディレクトリ名の取り出し
- int Get_ThinOut ()
  間引き数の取り出し
- int Get_OutputArrayShape ()
  出力配列形状の取り出し
- void Set_OutputArrayShape (int outputArrayShape)
  出力配列形状のセット
- int Get_OutputFilenameFormat ()
  出力ファイル名命名順の取り出し
- int Get_OutputGuideCell ()
  出力ガイドセル数の取り出し
- int Get_MultiFileCasting ()
  並列処理時のファイル割振り方法の取り出し
- int * Get_CropOndexStart ()
  入力領域のスタート位置取り出し
- int * Get_CropOndexEnd ()
  入力領域のエンド位置取り出し

Public 変数

- cpm_ParaManager * m_paraMngr

Protected 変数

- vector< std::string > m_in_dfi_name
  読込み *dfi* **ファイルリスト**
- vector< std::string > m_out_dfi_name
  出力 *dfi* **ファイルリスト**
- vector< std::string > m_out_proc_name
  出力する *proc* **ファイルリスト**
- std::string m_outdir_name
  出力先ディレクトリー名
- int m_thin_count
  間引き数
- int m_output_data_type
  出力実数タイプ *byte,short,int,float,double*
- int m_out_format
  出力ファイルフォーマット *sph,bov,avs,plot3d,vtk*
- int m_out_format_type
  出力形式 *ascii,binary,FortranBinary*
- int m_conv_type
  convert **タイプ** *Mx1 MxN MxM*
- int m_outputDiv [3]
  出力分割数 *MxN* **で有効**
- int m_outputArrayShape
  出力配列形状
- int m_outputFilenameFormat
  出力ファイル名命名順
- int m_outputGuideCell
  出力するガイドセル数
- int m_multiFileCasting
  並列処理時のファイル割振り方法
- int m_cropIndexStart [3]
  入力領域のスタート位置（*2014* **対応予定***)*
- int m_cropIndexEnd [3]
  入力領域のエンド位置（*2014* **対応予定***)*

4.12.1 説明

InputParam.h の 44 行で定義されています。

4.12.2 コンストラクタとデストラクタ

4.12.2.1 InputParam::InputParam ( )

コンストラクタ

InputParam.C の 21 行で定義されています。

参照先 m_in_dfi_name, m_out_dfi_name, m_out_format_type, m_out_proc_name, m_outputArrayShape, m_outputDiv, m_outputFilenameFormat, と m_thin_count.

```
22 {
23   m_thin_count=1;
24   m_out_format_type=CIO::E_CIO_OUTPUT_TYPE_BINARY;
25   m_outputDiv[0]=-1;m_outputDiv[1]=-1;m_outputDiv[2]=-1;
26   m_in_dfi_name.clear();
27   m_outputArrayShape=CIO::E_CIO_ARRAYSHAPE_UNKNOWN;
28   m_outputFilenameFormat=CIO::E_CIO_FNAME_STEP_RANK;
29
30   m_in_dfi_name.clear();
31   m_out_dfi_name.clear();
32   m_out_proc_name.clear();
33
34 }
```

4.12.2.2 InputParam::~InputParam ( )

デストラクタ

InputParam.C の 38 行で定義されています。

```
39 {
40
41 }
```

4.12.3 関数

4.12.3.1 int InputParam::Get_ConvType ( ) [inline]

コンバートタイプの取り出し

InputParam.h の 122 行で定義されています。

参照元 CONV::CheckConvData(), CONV::ConvInit(), main(), と convOutput_AVS::output_avs().

```
122 { return m_conv_type; };
```

4.12.3.2 int* InputParam::Get_CropOndexEnd ( ) [inline]

入力領域のエンド位置取り出し

InputParam.h の 202 行で定義されています。

```
202 { return m_cropIndexEnd; };
```

4.12.3.3 int* InputParam::Get_CropOndexStart ( ) [inline]

入力領域のスタート位置取り出し

InputParam.h の 197 行で定義されています。

```
197 { return m_cropIndexStart; };
```

4.12.3.4 vector<std::string> InputParam::Get_IndfiNameList ( ) [inline]

input dfi ファイル名リストの取り出し

InputParam.h の 94 行で定義されています。

参照元 convMx1::exec(), と CONV::ReadDfiFiles().

```
94 { return m_in_dfi_name; };
```

4.12.3.5 int InputParam::Get_MultiFileCasting ( ) [inline]

並列処理時のファイル割振り方法の取り出し

InputParam.h の 192 行で定義されています。

参照元 CONV::CheckConvData(), と convMxM::exec().

```
192 { return m_multiFileCasting; };
```

4.12.3.6 vector<std::string> InputParam::Get_OutdfiNameList ( ) [inline]

output dfi ファイル名リストの取り出し

InputParam.h の 99 行で定義されています。

参照元 CONV::CheckConvData(), convMxM::exec(), convMx1::exec(), convMxN::exec(), convMxN::VoxelInit(), と CONV::WriteIndexDfiFile().

```
99 { return m_out_dfi_name; };
```

4.12.3.7 vector<std::string> InputParam::Get_OutprocNameList ( ) [inline]

output proc ファイル名リストの取り出し

InputParam.h の 104 行で定義されています。

参照元 CONV::CheckConvData(), convMxM::exec(), convMx1::exec(), convMxN::VoxelInit(), と CONV::WriteIndexDfiFile().

```
104 { return m_out_proc_name; };
```

4.12.3.8 int InputParam::Get_OutputArrayShape ( ) [inline]

出力配列形状の取り出し

InputParam.h の 171 行で定義されています。

参照元 CONV::CheckConvData(), convMx1::exec(), convMxN::exec(), convMxM::mxmsolv(), convMxN::VoxelInit(), と CONV::WriteIndexDfiFile().

```
171 { return m_outputArrayShape; };
```

4.12.3.9 int InputParam::Get_OutputDataType ( ) [inline]

出力タイプの取り出し

InputParam.h の 151 行で定義されています。

参照元 CONV::CheckConvData(), convMx1::exec(), convMxN::exec(), convMxM::mxmsolv(), convMxN::VoxelInit(), convOutput_PLOT3D::WriteGridData(), と CONV::WriteIndexDfiFile().

```
151 { return m_output_data_type; };
```

4.12.3.10 std::string InputParam::Get_OutputDir ( ) [inline]

出力先ディレクトリ名の取り出し

InputParam.h の 161 行で定義されています。

参照元 main(), convMxM::mxmsolv(), convOutput_AVS::output_avs_coord(), convOutput_AVS::output_avs_header(), convOutput_AVS::OutputFile_Open(), convOutput_BOV::OutputFile_Open(), convOutput_SPH::OutputFile_Open(), convOutput_VTK::OutputFile_Open(), convOutput_PLOT3D::OutputFile_Open(), convOutput_PLOT3D::OutputPlot3D_xyz(), convMxN::VoxelInit(), と CONV::WriteIndexDfiFile().

```
161 { return m_outdir_name; };
```

4.12.3.11 int* InputParam::Get_OutputDivision ( ) [inline]

出力分割数の取り出し

InputParam.h の 127 行で定義されています。

参照元 convMxN::VoxelInit().

```
127 { return m_outputDiv; };
```

4.12.3.12 int InputParam::Get_OutputFilenameFormat ( ) [inline]

出力ファイル名命名順の取り出し

InputParam.h の 182 行で定義されています。

参照元 convMxN::exec(), convMxM::mxmsolv(), convOutput_AVS::output_avs_coord(), convOutput_AVS::output_avs_header(), convOutput_AVS::OutputFile_Open(), convOutput_BOV::OutputFile_Open(), convOutput_SPH::OutputFile_Open(), convOutput_VTK::OutputFile_Open(), convOutput_PLOT3D::OutputFile_Open(), と convOutput_PLOT3D::OutputPlot3D_xyz().

```
182 { return m_outputFilenameFormat; };
```

4.12.3.13 int InputParam::Get_OutputFormat ( ) [inline]

出力ファイルフォーマットの取り出し

InputParam.h の 132 行で定義されています。

参照元 CONV::CheckConvData(), CONV::ConvInit(), convMx1::exec(), convMxN::exec(), convMxM::mxmsolv(), convMxN::VoxelInit(), と CONV::WriteIndexDfiFile().

```
132 { return m_out_format; };
```

4.12.3.14 std::string InputParam::Get_OutputFormat_string ( ) [inline]

出力ファイルフォーマットの取り出し (文字列)

InputParam.h の 137 行で定義されています。

参照元 CONV::CheckConvData().

```
138   {
139     if( Get_OutputFormat() == (int)CIO::E_CIO_FMT_SPH ) return "sph";
140     if( Get_OutputFormat() == (int)CIO::E_CIO_FMT_BOV ) return "bov";
141     if( Get_OutputFormat() == (int)CIO::E_CIO_FMT_AVS ) return "avs";
142     if( Get_OutputFormat() == (int)CIO::E_CIO_FMT_VTK ) return "vtk";
143     if( Get_OutputFormat() == (int)CIO::E_CIO_FMT_PLOT3D ) return "plot3d";
144     return "";
145   };
```

4.12.3.15 int InputParam::Get_OutputFormatType ( ) [inline]

出力形式の取り出し（ascii,binary,FortranBinary)

InputParam.h の 156 行で定義されています。

参照元 CONV::CheckConvData(), convMxM::mxmsolv(), convOutput_VTK::OutputFile_Open(), convOutput_PLOT3D::OutputFile_Open(), convOutput_PLOT3D::OutputPlot3D_xyz(), convMxN::VoxelInit(), convOutput_PLOT3D::WriteBlockData(), convOutput_PLOT3D::WriteDataMarker(), convOutput_VTK::WriteFieldData(), convOutput_PLOT3D::WriteFuncBlockData(), convOutput_PLOT3D::WriteFuncData(), convOutput_VTK::WriteHeaderRecord(), convOutput_PLOT3D::WriteNgrid(), と convOutput_PLOT3D::WriteXYZData().

```
156 { return m_out_format_type; };
```

4.12.3.16 int InputParam::Get_OutputGuideCell ( ) [inline]

出力ガイドセル数の取り出し

InputParam.h の 187 行で定義されています。

参照元 CONV::CheckConvData(), convMx1::convMx1_out_ijkn(), convMx1::convMx1_out_nijk(), convMx1::exec(), convMxN::exec(), convMxM::mxmsolv(), convMxN::VoxelInit(), と CONV::WriteIndexDfiFile().

```
187 { return m_outputGuideCell; };
```

4.12.3.17 int InputParam::Get_ThinOut ( ) [inline]

間引き数の取り出し

InputParam.h の 166 行で定義されています。

参照元 convMx1::convMx1_out_ijkn(), convMx1::convMx1_out_nijk(), CONV::copyArray(), convMx1::exec(), convMxN::exec(), CONV::makeProcInfo(), convMxM::mxmsolv(), convOutput_AVS::output_avs_Mx1(), convOutput_AVS::output_avs_MxM(), convOutput_AVS::output_avs_MxN(), convOutput_PLOT3D::OutputPlot3D_xyz(), と convMxN::VoxelInit().

```
166 { return m_thin_count; };
```

4.12.3.18 bool InputParam::importCPM (cpm_ParaManager * *paraMngr*)

CPM のポインタをコピー

引数

in	*paraMngr*	cpm_ParaManager クラス

戻り値

エラーコード

InputParam.C の 45 行で定義されています。

参照先 m_paraMngr.

参照元 main().

```
46 {
47   if( !paraMngr ) return false;
48   m_paraMngr = paraMngr;
49   return true;
50 }
```

4.12.3.19 bool InputParam::Read (std::string *input_file_name*)

入力ファイルの読み込み

引数

in	*input_file_name*	入力TP ファイル名

InputParam.C の 55 行で定義されています。

参照先 E_OUTPUT_Mx1, E_OUTPUT_MxM, E_OUTPUT_MxN, E_OUTPUT_RANK, E_OUTPUT_STEP, Exit, Hostonly_, m_conv_type, m_cropIndexEnd, m_cropIndexStart, m_in_dfi_name, m_multiFileCasting, m_out_dfi_name, m_out_format, m_out_format_type, m_out_proc_name, m_outdir_name, m_output_data_type, m_outputArrayShape, m_outputDiv, m_outputFilenameFormat, m_outputGuideCell, m_thin_count, と stamped_printf.

参照元 main().

```
56 {
57   FILE* fp=NULL;
58   string str;
59   string label,label_base,label_leaf;
60
61   // TextParser のインスタンス
62   TextParser tpCntl;
63
64   // 入力ファイルをセット
65   int err = tpCntl.read(input_file_name);
66
67   // node 数の取得
68   int nnode=0;
69   label_base = "/ConvData";
70   if( tpCntl.chkNode(label_base) ) {
71     nnode = tpCntl.countLabels(label_base);
72   }
73
74   // 読込み dfi ファイル名の読込み
75   label_base = "/ConvData";
76   for(int i=0; i<nnode; i++) {
77     if( !tpCntl.getNodeStr(label_base, i+1, str) ) {
78       printf("\tParsing error : No Elem name\n");
79       Exit(0);
80     }
81     if( strcasecmp(str.substr(0,8).c_str(), "InputDFI") ) continue;
82     label = label_base+"/"+str;
83     if ( !(tpCntl.getInspectedValue(label, str )) ) {
84       printf("\tParsing error : fail to get '%s'\n", label.c_str());
85       Exit(0);
86     }
87     //dfi ファイル名を格納
88     m_in_dfi_name.push_back(str);
89   }
90
91   //出力 dfi ファイル名の読込み
92   label_base = "/ConvData";
93   for(int i=0; i<nnode; i++) {
94     if( tpCntl.getNodeStr(label_base, i+1, str) ) {
95       if( strcasecmp(str.substr(0,9).c_str(), "OutputDFI") ) continue;
96       label = label_base+"/"+str;
97       if ( !(tpCntl.getInspectedValue(label, str )) ) {
98         printf("\tParsing error : fail to get '%s'\n", label.c_str());
99         Exit(0);
100       }
101     }
102     //dfi ファイル名を格納
103     printf("OutputDFI : %s\n",str.c_str());
104     m_out_dfi_name.push_back(str);
105   }
106
107   //出力 proc ファイル名の読込み
108   label_base = "/ConvData";
109   for(int i=0; i<nnode; i++) {
110     if( tpCntl.getNodeStr(label_base, i+1, str) ) {
111       if( strcasecmp(str.substr(0,13).c_str(), "OutputProcDFI") ) continue;
112       label = label_base+"/"+str;
113       if ( !(tpCntl.getInspectedValue(label, str )) ) {
114         printf("\tParsing error : fail to get '%s'\n", label.c_str());
115         Exit(0);
116       }
117     }
118     //dfi ファイル名を格納
119     printf("OutputProcDFI : %s\n",str.c_str());
120     m_out_proc_name.push_back(str);
121   }
122
123   // コンバートタイプの読込み
124   label = "/ConvData/ConvType";
125   if( (tpCntl.getInspectedValue(label,str)) ) {
126     if     ( !strcasecmp(str.c_str(), "Mx1") )  m_conv_type = E_OUTPUT_Mx1;
```

```
127     else if( !strcasecmp(str.c_str(), "MxN") )  m_conv_type = E_OUTPUT_MxN;
128     else if( !strcasecmp(str.c_str(), "MxM") )  m_conv_type = E_OUTPUT_MxM;
129     else
130     {
131       Hostonly_ stamped_printf("\tInvalid keyword is described for '%s'\n", label.c_str());
132       Exit(0);
133     }
134   }
135
136   // 出力分割数の読込み
137   int vec[3];
138   label = "/ConvData/OutputDivision";
139   if( (tpCntl.getInspectedVector(label, vec, 3)) ) {
140     m_outputDiv[0]=vec[0];
141     m_outputDiv[1]=vec[1];
142     m_outputDiv[2]=vec[2];
143   }
144
145   // 出力ファイルフォーマットの読込み
146   label = "/ConvData/OutputFormat";
147   if ( !(tpCntl.getInspectedValue(label, str )) )
148   {
149     Hostonly_ stamped_printf("\tParsing error : fail to get '%s'\n", label.c_str());
150     Exit(0);
151   }
152   if      ( !strcasecmp(str.c_str(), "sph" ) )    m_out_format = CIO::E_CIO_FMT_SPH;
153   else if( !strcasecmp(str.c_str(), "plot3d" ) ) m_out_format = CIO::E_CIO_FMT_PLOT3D;
154   else if( !strcasecmp(str.c_str(), "avs" ) )    m_out_format = CIO::E_CIO_FMT_AVS;
155   else if( !strcasecmp(str.c_str(), "bov" ) )    m_out_format = CIO::E_CIO_FMT_BOV;
156   else if( !strcasecmp(str.c_str(), "vtk" ) )    m_out_format = CIO::E_CIO_FMT_VTK;
157   else
158   {
159     Hostonly_ stamped_printf("\tInvalid keyword is described for  '%s'\n", label.c_str());
160     Exit(0);
161   }
162
163   // 出力データタイプの読込み
164   label = "/ConvData/OutputDataType";
165   if( !(tpCntl.getInspectedValue(label, str )) ) {
166     m_output_data_type = CIO::E_CIO_DTYPE_UNKNOWN;
167   } else {
168     if      ( !strcasecmp(str.c_str(), "UInt8"  ) ) m_output_data_type = CIO::E_CIO_UINT8;
169     else if( !strcasecmp(str.c_str(),  "Int8"  ) ) m_output_data_type = CIO::E_CIO_INT8;
170     else if( !strcasecmp(str.c_str(), "UInt16" ) ) m_output_data_type = CIO::E_CIO_UINT16;
171     else if( !strcasecmp(str.c_str(),  "Int16" ) ) m_output_data_type = CIO::E_CIO_INT16;
172     else if( !strcasecmp(str.c_str(), "UInt32" ) ) m_output_data_type = CIO::E_CIO_UINT32;
173     else if( !strcasecmp(str.c_str(),  "Int32" ) ) m_output_data_type = CIO::E_CIO_INT32;
174     else if( !strcasecmp(str.c_str(), "UInt64" ) ) m_output_data_type = CIO::E_CIO_UINT64;
175     else if( !strcasecmp(str.c_str(),  "Int64" ) ) m_output_data_type = CIO::E_CIO_INT64;
176     else if( !strcasecmp(str.c_str(), "Float32") ) m_output_data_type = CIO::E_CIO_FLOAT32;
177     else if( !strcasecmp(str.c_str(), "Float64") ) m_output_data_type = CIO::E_CIO_FLOAT64;
178     else
179     {
180       printf("\tInvalid keyword is described for '%s'\n", label.c_str());
181       Exit(0);
182     }
183   }
184
185   // 出力形式の読込み
186   label = "/ConvData/OutputFormatType";
187   if( !(tpCntl.getInspectedValue(label, str )) ) {
188     m_out_format_type = CIO::E_CIO_OUTPUT_TYPE_BINARY;
189   } else {
190     if      ( !strcasecmp(str.c_str(), "binary") )        m_out_format_type=CIO::E_CIO_OUTPUT_TYPE_BINARY;
191     else if( !strcasecmp(str.c_str(), "ascii") )         m_out_format_type=CIO::E_CIO_OUTPUT_TYPE_ASCII;
192     else if( !strcasecmp(str.c_str(), "FortranBinary") ) m_out_format_type=CIO::E_CIO_OUTPUT_TYPE_FBINARY;
193     else
194     {
195       printf("\tInvalid keyword is described for '%s'\n", label.c_str());
196       Exit(0);
197     }
198   }
199
200   // 出力先ディレクトリの読込み
201   label = "/ConvData/OutputDir";
202   if( !(tpCntl.getInspectedValue(label, str )) ) {
203     Hostonly_ stamped_printf("\tParsing error : fail to get '%s'\n", label.c_str());
204     Exit(0);
205   } else {
206     m_outdir_name = str;
207     //if( m_outdir_name.size() != 0 ) m_outdir_name=m_outdir_name+"/";
208   }
209
210   //間引き数の読込み
211   int ict;
212   label = "/ConvData/ThinningOut";
213   if( !(tpCntl.getInspectedValue(label, ict )) ) {
```

```
214     m_thin_count=1;
215   } else {
216     if( ict < 0 ) {
217       printf("\tInvalid keyword is described for '%s'\n", label.c_str());
218       Exit(0);
219     }
220     m_thin_count = ict;
221   }
222
223   //出力配列形状の読込み
224   label = "/ConvData/OutputArrayShape";
225   if( !(tpCntl.getInspectedValue(label, str )) ) {
226     m_outputArrayShape = CIO::E_CIO_ARRAYSHAPE_UNKNOWN;
227   } else {
228     if      ( !strcasecmp(str.c_str(), "ijkn") ) m_outputArrayShape = CIO::E_CIO_IJKN;
229     else if( !strcasecmp(str.c_str(), "nijk") ) m_outputArrayShape = CIO::E_CIO_NIJK;
230     else {
231       printf("\tInvalid keyword is described for '%s'\n", label.c_str());
232       Exit(0);
233     }
234   }
235
236   //出力ファイル名命名順の読込み
237   label = "/ConvData/OutputFilenameFormat";
238   if( !(tpCntl.getInspectedValue(label, str )) ) {
239     m_outputFilenameFormat = CIO::E_CIO_FNAME_STEP_RANK;
240   } else {
241     if      ( !strcasecmp(str.c_str(), "step_rank") ) m_outputFilenameFormat = CIO::E_CIO_FNAME_STEP_RANK;
242     else if( !strcasecmp(str.c_str(), "rank_step") ) m_outputFilenameFormat = CIO::E_CIO_FNAME_RANK_STEP;
243     else {
244       printf("\tInvalid keyword is described for '%s'\n", label.c_str());
245       Exit(0);
246     }
247   }
248
249   //出力ガイドセル数
250   label = "/ConvData/OutputGuideCell";
251   if( !(tpCntl.getInspectedValue(label, ict )) ) {
252     m_outputGuideCell=0;
253   } else {
254     if( ict < 0 ) {
255       printf("\tInvalid keyword is described for '%s'\n", label.c_str());
256       Exit(0);
257     }
258     m_outputGuideCell=ict;
259   }
260   printf("m_outputGuideCell : %d\n",m_outputGuideCell);
261
262   //並列処理時のファイル割振り方法
263   label = "/ConvData/MultiFileCasting";
264   if( !(tpCntl.getInspectedValue(label, str )) ) {
265     m_multiFileCasting = E_OUTPUT_STEP;
266   } else {
267     if      ( !strcasecmp(str.c_str(), "step") ) m_multiFileCasting =
   E_OUTPUT_STEP;
268     else if( !strcasecmp(str.c_str(), "rank") ) m_multiFileCasting =
   E_OUTPUT_RANK;
269     else {
270       printf("\tInvalid keyword is described for '%s'\n", label.c_str());
271       Exit(0);
272     }
273   }
274
275   printf("m_multiFileCasting : %d\n",m_multiFileCasting);
276
277   //入力領域のスタート位置（2014 対応予定）
278   vec[3];
279   label = "/ConvData/CropOndexStart";
280   if( (tpCntl.getInspectedVector(label, vec, 3)) ) {
281     m_cropIndexStart[0]=vec[0];
282     m_cropIndexStart[1]=vec[1];
283     m_cropIndexStart[2]=vec[2];
284   }
285
286   //入力領域のエンド位置（2014 対応予定）
287   vec[3];
288   label = "/ConvData/CropOndexEnd";
289   if( (tpCntl.getInspectedVector(label, vec, 3)) ) {
290     m_cropIndexEnd[0]=vec[0];
291     m_cropIndexEnd[1]=vec[1];
292     m_cropIndexEnd[2]=vec[2];
293   }
294
295   // TextParser の破棄
296   tpCntl.remove();
297
298   return true;
```

```
299
300 }
```

4.12.3.20 bool InputParam::Set_OutprocNameList (vector< std::string > *out_proc_name*) [inline]

output proc ファイル名リストをセットする

引数

in	*out_proc_name*	セットする proc ファイル名リスト

InputParam.h の 110 行で定義されています。

参照元 CONV::CheckConvData().

```
111   { if( out_proc_name.size() < 1 ) return false;
112     for(int i=0; i<out_proc_name.size(); i++) {
113       m_out_proc_name.push_back(out_proc_name[i]);
114     }
115     return true;
116   }
```

4.12.3.21 void InputParam::Set_OutputArrayShape (int *outputArrayShape*) [inline]

出力配列形状のセット

InputParam.h の 176 行で定義されています。

参照元 CONV::CheckConvData().

```
177       { m_outputArrayShape = outputArrayShape; };
```

4.12.4 変数

4.12.4.1 int InputParam::m_conv_type [protected]

convert タイプ Mx1 MxN MxM

InputParam.h の 60 行で定義されています。

参照元 Read().

4.12.4.2 int InputParam::m_cropIndexEnd[3] [protected]

入力領域のエンド位置（2014 対応予定)

InputParam.h の 67 行で定義されています。

参照元 Read().

4.12.4.3 int InputParam::m_cropIndexStart[3] [protected]

入力領域のスタート位置（2014 対応予定)

InputParam.h の 66 行で定義されています。

参照元 Read().

4.12.4.4 vector<std::string> InputParam::m_in_dfi_name [protected]

読込み dfi ファイルリスト

InputParam.h の 52 行で定義されています。

参照元 InputParam(), と Read().

4.12.4.5 int InputParam::m_multiFileCasting [protected]

並列処理時のファイル割振り方法

InputParam.h の 65 行で定義されています。

参照元 Read().

4.12.4.6 vector<std::string> InputParam::m_out_dfi_name [protected]

出力 dfi ファイルリスト

InputParam.h の 53 行で定義されています。

参照元 InputParam(), と Read().

4.12.4.7 int InputParam::m_out_format [protected]

出力ファイルフォーマット sph,bov,avs,plot3d,vtk

InputParam.h の 58 行で定義されています。

参照元 Read().

4.12.4.8 int InputParam::m_out_format_type [protected]

出力形式 ascii,binary,FortranBinary

InputParam.h の 59 行で定義されています。

参照元 InputParam(), と Read().

4.12.4.9 vector<std::string> InputParam::m_out_proc_name [protected]

出力する proc ファイルリスト

InputParam.h の 54 行で定義されています。

参照元 InputParam(), と Read().

4.12.4.10 std::string InputParam::m_outdir_name [protected]

出力先ディレクトリー名

InputParam.h の 55 行で定義されています。

参照元 Read().

4.12.4.11 int InputParam::m_output_data_type [protected]

出力実数タイプ byte,short,int,float,double

InputParam.h の 57 行で定義されています。

参照元 Read().

4.12.4.12 int InputParam::m_outputArrayShape [protected]

出力配列形状

InputParam.h の 62 行で定義されています。

参照元 InputParam(), と Read().

4.12.4.13 int InputParam::m_outputDiv[3] [protected]

出力分割数 MxN で有効

InputParam.h の 61 行で定義されています。

参照元 InputParam(), と Read().

4.12.4.14 int InputParam::m_outputFilenameFormat [protected]

出力ファイル名命名順

InputParam.h の 63 行で定義されています。

参照元 InputParam(), と Read().

4.12.4.15 int InputParam::m_outputGuideCell [protected]

出力するガイドセル数

InputParam.h の 64 行で定義されています。

参照元 Read().

4.12.4.16 cpm_ParaManager* InputParam::m_paraMngr

InputParam.h の 48 行で定義されています。

参照元 importCPM().

4.12.4.17 int InputParam::m_thin_count [protected]

間引き数

InputParam.h の 56 行で定義されています。

参照元 InputParam(), と Read().

このクラスの説明は次のファイルから生成されました:

- InputParam.h
- InputParam.C

4.13 構造体 CONV::step_rank_info

```
#include <conv.h>
```

Public 変数

- cio_DFI * dfi
- int stepStart
- int stepEnd
- int rankStart
- int rankEnd

4.13.1 説明

conv.h の 52 行で定義されています。

4.13.2 変数

4.13.2.1 cio_DFI* CONV::step_rank_info::dfi

conv.h の 53 行で定義されています。

参照元 CONV::makeRankList(), と CONV::makeStepList().

4.13.2.2 int CONV::step_rank_info::rankEnd

conv.h の 57 行で定義されています。

参照元 CONV::makeRankList().

4.13.2.3 int CONV::step_rank_info::rankStart

conv.h の 56 行で定義されています。

参照元 CONV::makeRankList().

4.13.2.4 int CONV::step_rank_info::stepEnd

conv.h の 55 行で定義されています。

参照元 CONV::makeStepList().

4.13.2.5 int CONV::step_rank_info::stepStart

conv.h の 54 行で定義されています。

参照元 CONV::makeStepList().

この構造体の説明は次のファイルから生成されました:

- conv.h

Chapter 5

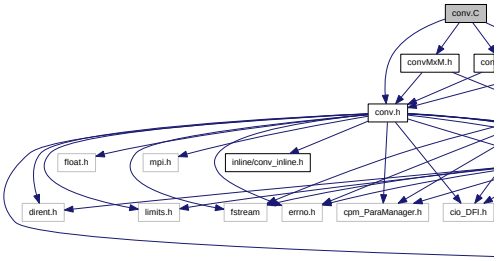
ファイル

5.1 conv.C

CONV Class.

```
#include "conv.h"
#include "convMx1.h"
#include "convMxM.h"
#include "convMxN.h"
```

conv.C のインクルード依存関係図



5.1.1 説明

CONV Class.

作者

kero

conv.C で定義されています。

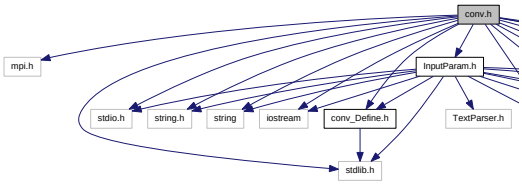
5.2 conv.h

CONV Class Header.

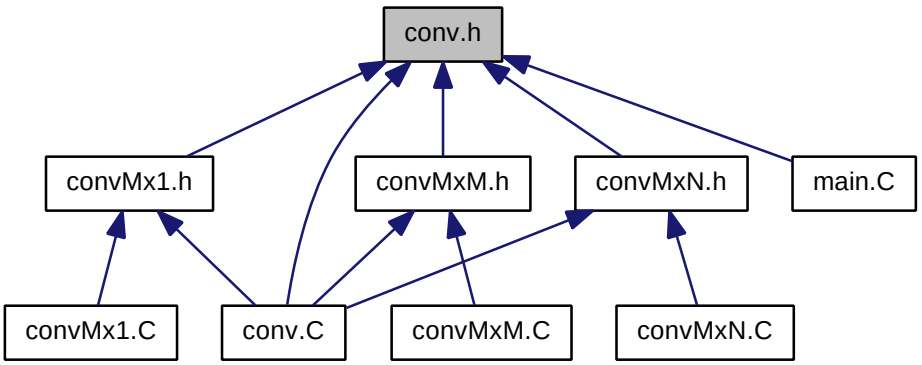
```
#include "mpi.h"
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <iostream>
#include <fstream>
#include <errno.h>
#include <float.h>
#include <dirent.h>
#include "cpm_ParaManager.h"
#include "cio_DFI.h"
#include "limits.h"
#include "conv_Define.h"
#include "InputParam.h"
#include "inline/conv_inline.h"
```

conv.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



構成

- class CONV
- struct CONV::step_rank_info
- struct CONV::dfi_MinMax

5.2.1 説明

CONV Class Header.

作者

kero

日付

2013/11/7

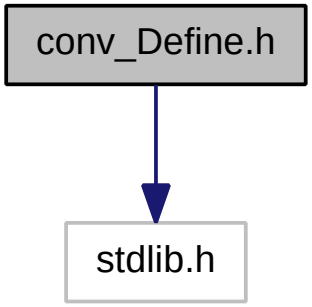
conv.h で定義されています。

5.3 conv_Define.h

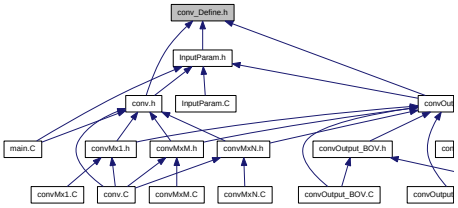
CONV Definition Header.

```
#include <stdlib.h>
```

conv_Define.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



マクロ定義

- #define Exit(x) ((void)printf("exit at %s:%u\n", __FILE__, __LINE__), exit((x)))
- #define message() printf("\t%s (%d):\n",__FILE__, __LINE__)
- #define mark() printf("%s (%d):\n",__FILE__, __LINE__)
- #define stamped_printf printf("%s (%d): ",__FILE__, __LINE__), printf
- #define stamped_fprintf fprintf(fp, "%s (%d): ",__FILE__, __LINE__), fprintf
- #define Hostonly_ if(m_paraMngr->GetMyRankID()==0)
- #define LOG_OUT_ if(m_lflag)
- #define LOG_OUTV_ if(m_lflagv)
- #define STD_OUT_ if(m_pflag)
- #define STD_OUTV_ if(m_pflagv)
- #define ON 1
- #define OFF 0
- #define REAL_UNKNOWN 0
- #define SPH_FLOAT 1
- #define SPH_DOUBLE 2
- #define SPH_DATA_UNKNOWN 0
- #define SPH_SCALAR 1
- #define SPH_VECTOR 2
- #define _F_IDX_S3D(_I, _J, _K, _NI, _NJ, _NK, _VC)

列挙型

- enum E_OUTPUT_CONV { E_OUTPUT_Mx1 = 0, E_OUTPUT_MxN, E_OUTPUT_MxM }
- enum E_OUTPUT_MULTI_FILE_CAST { E_OUTPUT_CAST_UNKNOWN = -1, E_OUTPUT_STEP = 0, E_OUTPUT_RANK }

5.3.1 説明

CONV Definition Header.

作者

kero

日付

2013/11/7

conv_Define.h で定義されています。

5.3.2 マクロ定義

5.3.2.1 #define _F_IDX_S3D(_I, _J, _K, _NI, _NJ, _NK, _VC)

値:

```
( (size_t)(_K+_VC-1) * (size_t)(_NI+2*_VC) * (size_t)(_NJ+2*_VC) \
+ (size_t)(_J+_VC-1) * (size_t)(_NI+2*_VC) \
+ (size_t)(_I+_VC-1) \
)
```

3 次元インデクス (i,j,k) -> 1 次元インデクス変換マクロ

覚え書き

i,j,k インデクスはF 表記

引数

in	_I	i 方向インデクス
in	_J	j 方向インデクス
in	_K	k 方向インデクス
in	_NI	i 方向インデクスサイズ
in	_NJ	j 方向インデクスサイズ
in	_NK	k 方向インデクスサイズ
in	_VC	仮想セル数

戻り値

1 次元インデクス

conv_Define.h の 81 行で定義されています。

参照元 convOutput_PLOT3D::OutputPlot3D_xyz().

5.3.2.2 #define Exit(x) ((void)printf("exit at %s:%u\n", __FILE__, __LINE__), exit((x)))

conv_Define.h の 24 行で定義されています。

参照元 CONV::CheckDir(), CONV::OpenLogFile(), convOutput_AVS::output_avs_coord(), convOutput_AVS::output_avs_header(), convOutput_AVS::OutputFile_Open(), convOutput_VTK::OutputFile_Open(), convOutput_BOV::OutputFile_Open(), convOutput_SPH::OutputFile_Open(), convOutput_PLOT3D::OutputFile_Open(), convOutput_PLOT3D::OutputPlot3D_xyz(), InputParam::Read(), convMxN::VoxelInit(), convOutput_AVS::WriteFieldData(), convOutput_VTK::WriteFieldData(), と convOutput::WriteFieldData().

5.3.2.3 #define Hostonly_ if(m_paraMngr->GetMyRankID()==0)

conv_Define.h の 33 行で定義されています。

参照元 CONV::CheckDir(), と InputParam::Read().

5.3.2.4 #define LOG_OUT_ if(m_lflag)

conv_Define.h の 36 行で定義されています。

参照元 convMx1::exec(), と main().

5.3.2.5 #define LOG_OUTV_ if(m_lflagv)

conv_Define.h の 37 行で定義されています。

参照元 convMx1::exec(), と CONV::ReadDfiFiles().

5.3.2.6 #define mark( ) printf("%s (%d):\n",__FILE__, __LINE__)

conv_Define.h の 28 行で定義されています。

5.3.2.7 #define message( ) printf("\t%s (%d):\n",__FILE__, __LINE__)

conv_Define.h の 27 行で定義されています。

5.3.2.8 #define OFF 0

conv_Define.h の 60 行で定義されています。

5.3.2.9 #define ON 1

conv_Define.h の 59 行で定義されています。

5.3.2.10 #define REAL_UNKNOWN 0

conv_Define.h の 62 行で定義されています。

5.3.2.11 #define SPH_DATA_UNKNOWN 0

conv_Define.h の 66 行で定義されています。

5.3.2.12 #define SPH_DOUBLE 2

conv_Define.h の 64 行で定義されています。
参照元 convMx1::exec(), と convOutput_SPH::WriteHeaderRecord().

5.3.2.13 #define SPH_FLOAT 1

conv_Define.h の 63 行で定義されています。
参照元 convMx1::exec(), と convOutput_SPH::WriteHeaderRecord().

5.3.2.14 #define SPH_SCALAR 1

conv_Define.h の 67 行で定義されています。
参照元 convOutput_SPH::WriteHeaderRecord().

5.3.2.15 #define SPH_VECTOR 2

conv_Define.h の 68 行で定義されています。
参照元 convOutput_SPH::WriteHeaderRecord().

5.3.2.16 #define stamped_fprintf fprintf(fp, "%s (%d): ",__FILE__, __LINE__), fprintf

conv_Define.h の 31 行で定義されています。

5.3.2.17 #define stamped_printf printf("%s (%d): ",__FILE__, __LINE__), printf

conv_Define.h の 30 行で定義されています。
参照元 InputParam::Read().

5.3.2.18 #define STD_OUT_ if(m_pflag)

conv_Define.h の 38 行で定義されています。
参照元 convMx1::exec().

5.3.2.19 #define STD_OUTV_ if(m_pflagv)

conv_Define.h の 39 行で定義されています。
参照元 convMx1::exec(), と CONV::ReadDfiFiles().

5.3.3 列挙型

5.3.3.1 enum E_OUTPUT_CONV

コンバート形式

列挙型の値

E_OUTPUT_Mx1
E_OUTPUT_MxN M 対 1.

E_OUTPUT_MxM M 対N. M 対M

conv_Define.h の 43 行で定義されています。

```
44 {
45   E_OUTPUT_Mx1 = 0,
46   E_OUTPUT_MxN,
47   E_OUTPUT_MxM
48 };
```

5.3.3.2 enum E_OUTPUT_MULTI_FILE_CAST

並列処理時のファイル割振り方法

列挙型の値

E_OUTPUT_CAST_UNKNOWN

E_OUTPUT_STEP

E_OUTPUT_RANK

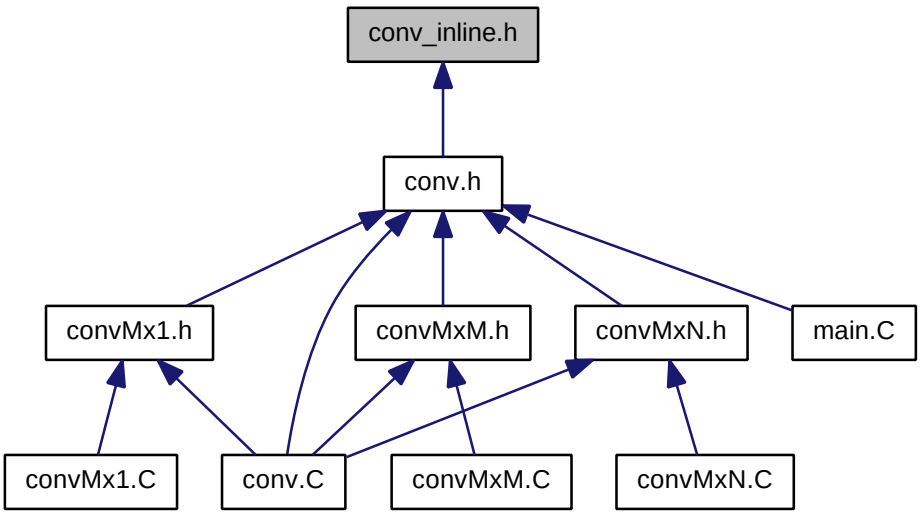
conv_Define.h の 51 行で定義されています。

```
52 {
53   E_OUTPUT_CAST_UNKNOWN = -1, //未定義
54   E_OUTPUT_STEP          = 0,  //step 基準
55   E_OUTPUT_RANK                //rank 基準
56 };
```

5.4 conv_inline.h

CONV クラスの inline 関数ヘッダーファイル

このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



マクロ定義

- #define CONV_INLINE inline

5.4.1 説明

CONV クラスの inline 関数ヘッダーファイル

作者

kero

日付

2013/11/7

conv_inline.h で定義されています。

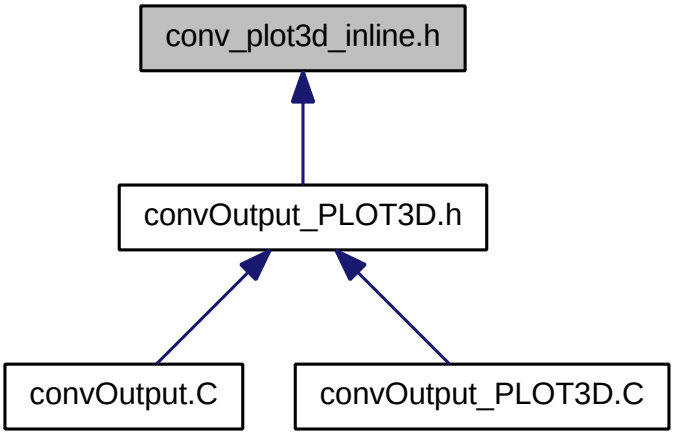
5.4.2 マクロ定義

5.4.2.1 #define CONV_INLINE inline

conv_inline.h の 22 行で定義されています。

5.5 conv_plot3d_inline.h

convOutput_PLOT3D クラスの inline 関数ヘッダーファイル
このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



マクロ定義

- #define CONV_INLINE inline

5.5.1 説明

convOutput_PLOT3D クラスの inline 関数ヘッダーファイル

作者

kero

日付

2013/11/7

conv_plot3d_inline.h で定義されています。

5.5.2 マクロ定義

5.5.2.1 #define CONV_INLINE inline

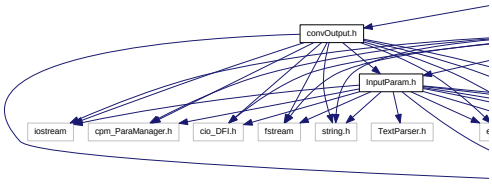
conv_plot3d_inline.h の 22 行で定義されています。

5.6 convMx1.C

convMx1 Class

```
#include "convMx1.h"
```

convMx1.C のインクルード依存関係図



5.6.1 説明

convMx1 Class

作者

kero

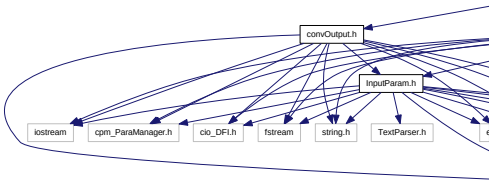
convMx1.C で定義されています。

5.7 convMx1.h

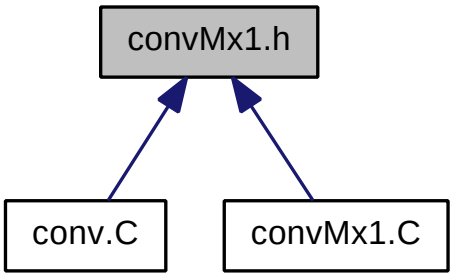
convMx1 Class Header

```
#include "conv.h"
#include "convOutput.h"
#include "inline/convMx1_inline.h"
```

convMx1.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



構成

- class convMx1

5.7.1 説明

convMx1 Class Header

作者

kero

日付

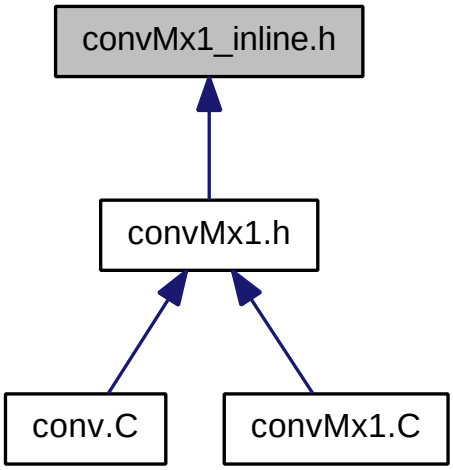
2013/11/14

convMx1.h で定義されています。

5.8 convMx1_inline.h

convMx1 クラスの inline 関数ヘッダーファイル

このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



マクロ定義

- #define CONV_INLINE inline

5.8.1 説明

convMx1 クラスの inline 関数ヘッダーファイル

作者

kero

日付

2013/11/7

convMx1_inline.h で定義されています。

5.8.2 マクロ定義

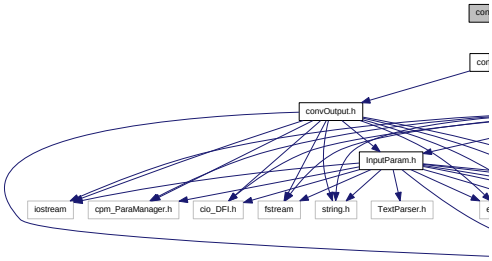
5.8.2.1 #define CONV_INLINE inline

convMx1_inline.h の 22 行で定義されています。

5.9 convMxM.C

convMxM Class

```
#include "convMxM.h"
```
convMxM.C のインクルード依存関係図



5.9.1 説明

convMxM Class

作者

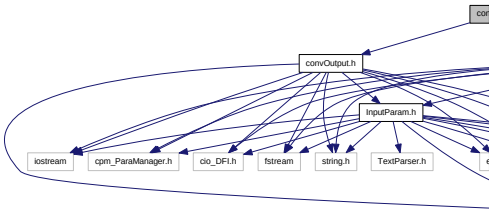
kero

convMxM.C で定義されています。

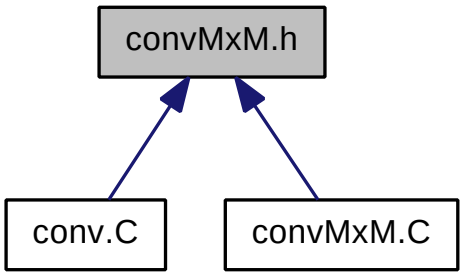
5.10 convMxM.h

convMxM Class Header

```
#include "conv.h"
#include "convOutput.h"
```
convMxM.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



構成

- class convMxM

5.10.1 説明

convMxM Class Header

作者

kero

日付

2013/11/14

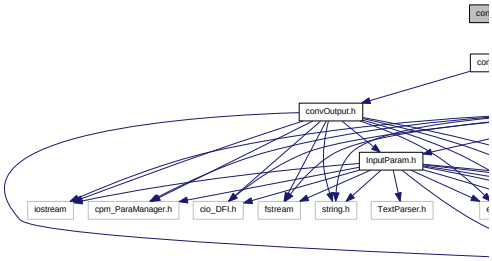
convMxM.h で定義されています。

5.11 convMxN.C

convMxN Class

```
#include "convMxN.h"
```

convMxN.C のインクルード依存関係図



5.11.1 説明

convMxN Class

作者

kero

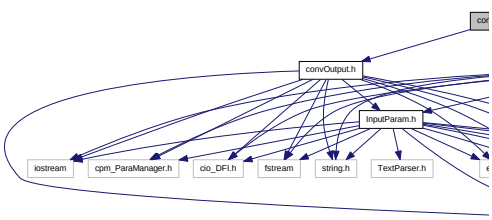
convMxN.C で定義されています。

5.12 convMxN.h

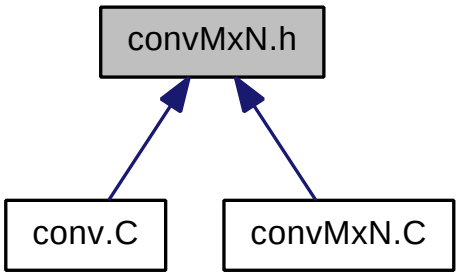
convMxN Class Header

```
#include "conv.h"
#include "convOutput.h"
```

convMxN.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



構成

- class convMxN

5.12.1 説明

convMxN Class Header

作者

kero

日付

2013/11/14

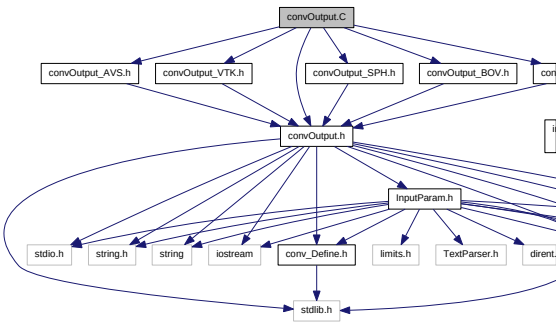
convMxN.h で定義されています。

5.13 convOutput.C

convOutput Class

```
#include "convOutput.h"
#include "convOutput_SPH.h"
#include "convOutput_BOV.h"
#include "convOutput_AVS.h"
#include "convOutput_VTK.h"
#include "convOutput_PLOT3D.h"
```

convOutput.C のインクルード依存関係図



5.13.1 説明

convOutput Class

作者

kero

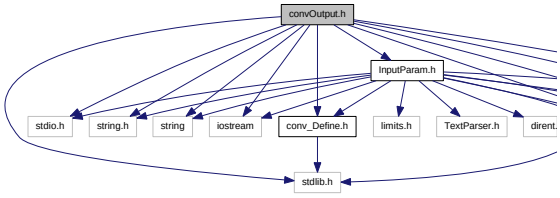
convOutput.C で定義されています。

5.14 convOutput.h

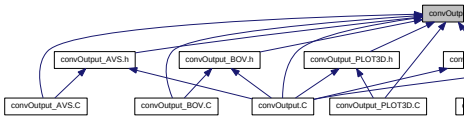
convOutput Class Header

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <iostream>
#include <fstream>
#include <errno.h>
#include "cpm_ParaManager.h"
#include "cio_DFI.h"
#include "conv_Define.h"
#include "InputParam.h"
```

convOutput.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



構成

- class convOutput

5.14.1 説明

convOutput Class Header

作者

kero

日付

2013/11/7

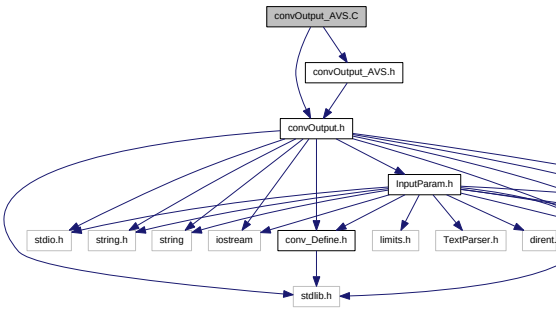
convOutput.h で定義されています。

5.15 convOutput_AVS.C

convOutput_AVS Class

```
#include "convOutput.h"
#include "convOutput_AVS.h"
```

convOutput_AVS.C のインクルード依存関係図



5.15.1 説明

convOutput_AVS Class

作者

kero

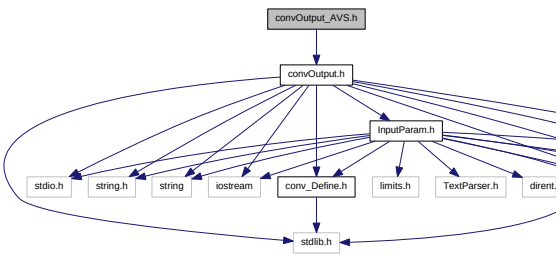
convOutput_AVS.C で定義されています。

5.16 convOutput_AVS.h

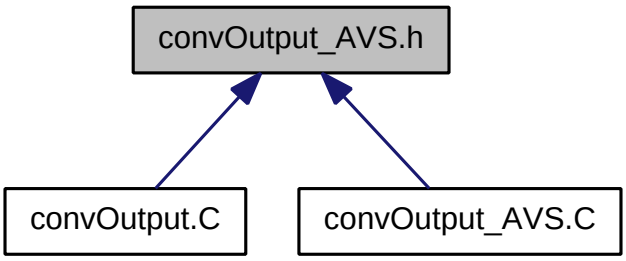
convOutput_AVS Class Header

```
#include "convOutput.h"
```

convOutput_AVS.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



構成

- class convOutput_AVS

5.16.1 説明

convOutput_AVS Class Header

作者

kero

日付

2013/11/7

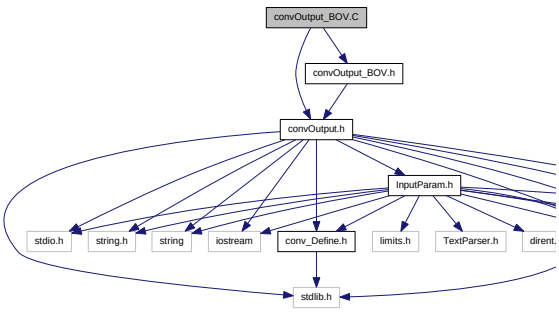
convOutput_AVS.h で定義されています。

5.17 convOutput_BOV.C

convOutput_BOV Class

```
#include "convOutput.h"
#include "convOutput_BOV.h"
```

convOutput_BOV.C のインクルード依存関係図



5.17.1 説明

convOutput_BOV Class

作者

kero

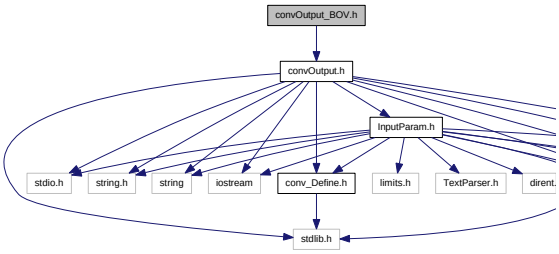
convOutput_BOV.C で定義されています。

5.18 convOutput_BOV.h

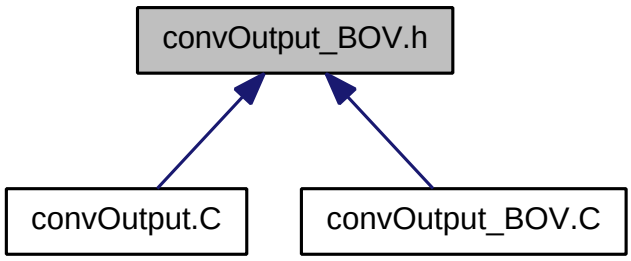
convOutput_BOV Class Header

```
#include "convOutput.h"
```

convOutput_BOV.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



構成

- class convOutput_BOV

5.18.1 説明

convOutput_BOV Class Header

作者

kero

日付

2013/11/7

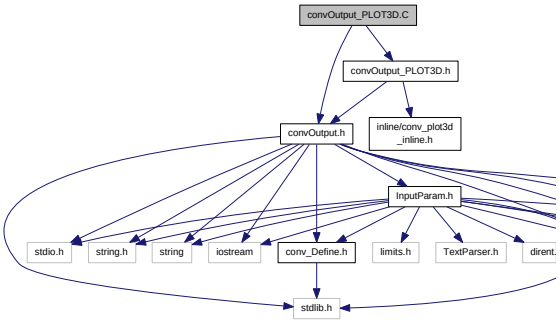
convOutput_BOV.h で定義されています。

5.19 convOutput_PLOT3D.C

convOutput_PLOT3D Class

```
#include "convOutput.h"
#include "convOutput_PLOT3D.h"
```

convOutput_PLOT3D.C のインクルード依存関係図



5.19.1 説明

convOutput_PLOT3D Class

作者

kero

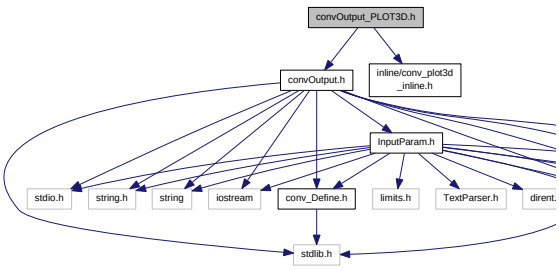
convOutput_PLOT3D.C で定義されています。

5.20 convOutput_PLOT3D.h

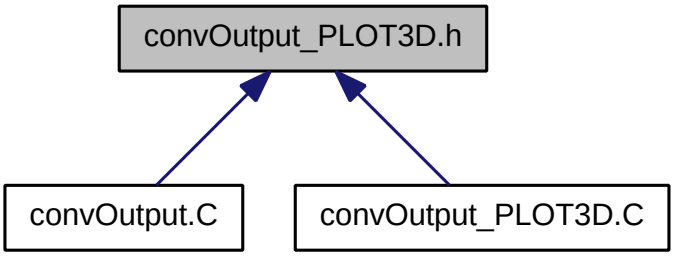
convOutput_PLOT3D Class Header

```
#include "convOutput.h"
#include "inline/conv_plot3d_inline.h"
```

convOutput_PLOT3D.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



構成

- class convOutput_PLOT3D

5.20.1 説明

convOutput_PLOT3D Class Header

作者

kero

日付

2013/11/7

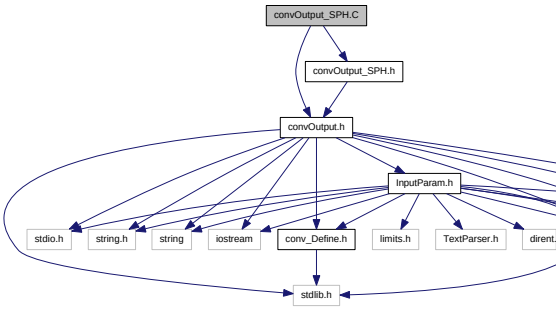
convOutput_PLOT3D.h で定義されています。

5.21 convOutput_SPH.C

convOutput_SPH Class

```
#include "convOutput.h"
#include "convOutput_SPH.h"
```

convOutput_SPH.C のインクルード依存関係図



5.21.1 説明

convOutput_SPH Class

作者

kero

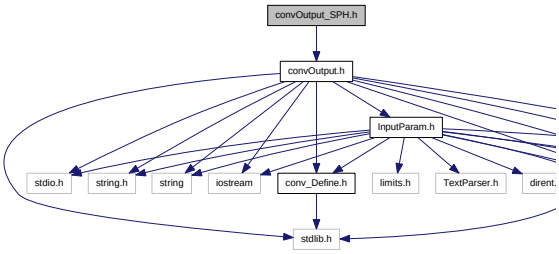
convOutput_SPH.C で定義されています。

5.22 convOutput_SPH.h

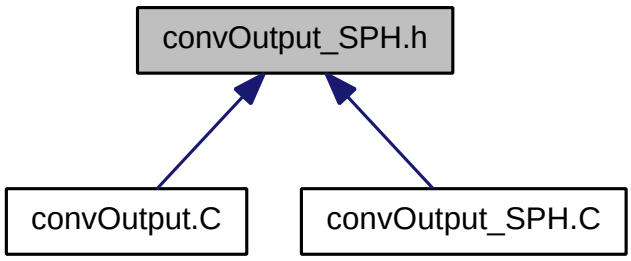
convOutput_SPH Class Header

```
#include "convOutput.h"
```

convOutput_SPH.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



構成

- class convOutput_SPH

5.22.1 説明

convOutput_SPH Class Header

作者

kero

日付

2013/11/7

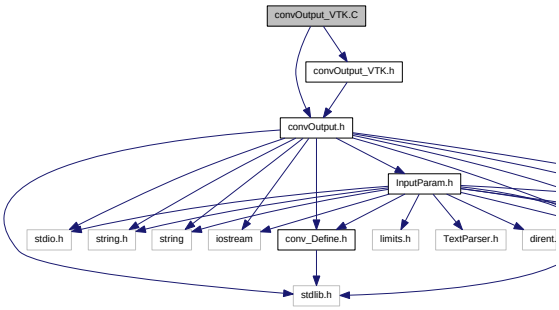
convOutput_SPH.h で定義されています。

5.23 convOutput_VTK.C

convOutput_VTK Class

```
#include "convOutput.h"
#include "convOutput_VTK.h"
```

convOutput_VTK.C のインクルード依存関係図



5.23.1 説明

convOutput_VTK Class

作者

kero

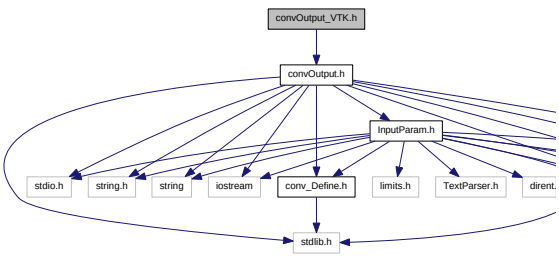
convOutput_VTK.C で定義されています。

5.24 convOutput_VTK.h

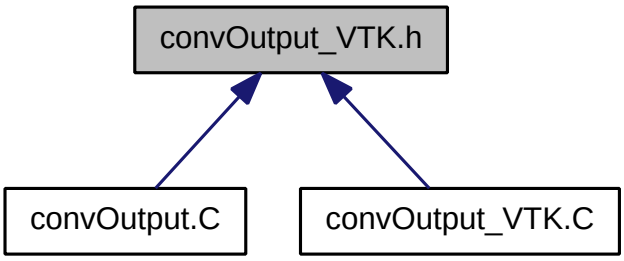
convOutput_VTK Class Header

```
#include "convOutput.h"
```

convOutput_VTK.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



構成

- class convOutput_VTK

5.24.1 説明

convOutput_VTK Class Header

作者

kero

日付

2013/11/12

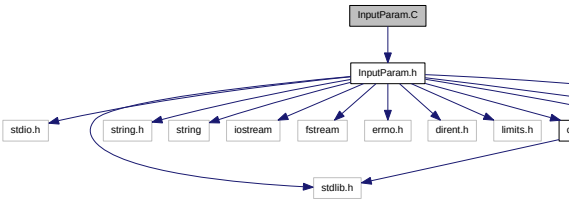
convOutput_VTK.h で定義されています。

5.25 InputParam.C

InputParam Class.

```
#include "InputParam.h"
```

InputParam.C のインクルード依存関係図



5.25.1 説明

InputParam Class.

作者

kero

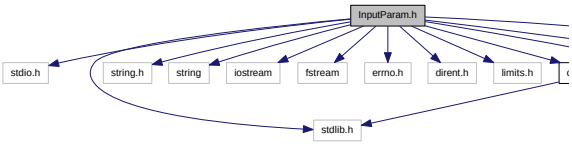
InputParam.C で定義されています。

5.26 InputParam.h

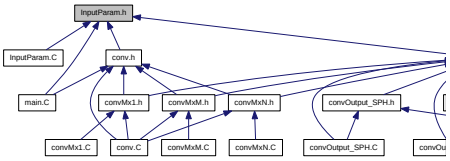
InputParam Class Header.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <iostream>
#include <fstream>
#include <errno.h>
#include <dirent.h>
#include "limits.h"
#include "conv_Define.h"
#include "TextParser.h"
#include "cpm_ParaManager.h"
#include "cio_DFI.h"
```

InputParam.h のインクルード依存関係図



このグラフは、どのファイルから直接、間接的にインクルードされているかを示しています。



構成

- class InputParam

5.26.1 説明

InputParam Class Header.

作者

kero

日付

2013/11/13

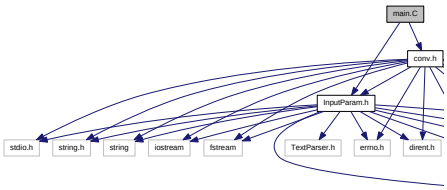
InputParam.h で定義されています。

5.27 main.C

conv の main 関数

```
#include "InputParam.h"
#include "conv.h"
```

main.C のインクルード依存関係図



関数

- void usage (const char *progname)
- int main (int argc, char **argv)

5.27.1 説明

conv の main 関数

作者

kero

main.C で定義されています。

5.27.2 関数

5.27.2.1 int main (int *argc*, char ** *argv*)

main.C の 36 行で定義されています。

参照先 CONV::CheckConvData(), CONV::CheckDir(), CONV::CloseLogFile(), CONV::ConvInit(), E_OUTPUT_Mx1, E_OUTPUT_MxM, E_OUTPUT_MxN, CONV::exec(), InputParam::Get_ConvType(), InputParam::Get_OutputDir(), InputParam::importCPM(), CONV::importCPM(), CONV::importInputParam(), LOG_OUT_, CONV::m_lflag, CONV::m_lflagv, CONV::m_pflag, CONV::m_pflagv, CONV::OpenLogFile(), InputParam::Read(), CONV::ReadDfiFiles(), usage(), CONV::VoxelInit(), と CONV::WriteTime().

```
37 {
38   char *progname = argv[0];
39   bool out_comb = false;
40   bool out_log  = false;
41   bool thin_out = false;
42   int thin_count=1;
43   string fname;
44   string dname="";
45   //int pflag=0;//出力しない
46   int pflag=1;//出力する
47   int pflagv=0;
48   int lflag;
49   int lflagv;
50
51   // タイミング用変数
52   double t0, t1, t2, t3, t4, t5;
53
54   // ################################################################
55   // COMB class のインスタンス
56   //CONV conv;
57
58   // ################################################################
59   // 初期処理
60
61   // 並列管理クラスのインスタンスと初期化
```

```
62   // ここで MPI_Init も行う
63   cpm_ParaManager* paraMngr = cpm_ParaManager::get_instance(argc, argv);
64   if( !paraMngr ) {
65     return CPM_ERROR_PM_INSTANCE;
66   }
67
68   // ####################################################################
69   // 入力オプション処理
70
71   int opt;
72   while ((opt = getopt(argc, argv, "avlf:d:hs:")) != -1) {
73     switch (opt) {
74     case 'f':
75       fname = optarg;
76       out_comb = true;
77       break;
78     case 'd':
79       dname = optarg;
80       break;
81     case 'v':
82       pflagv = 1;
83       break;
84     case 'l':
85       out_log = true;
86       break;
87     case 's':
88       thin_out = true;
89       thin_count = atoi(optarg);
90       break;
91     case 'h': // Show usage and exit
92       usage(progname);
93       return 0;
94       break;
95     case ':': // not find argument
96       usage(progname);
97       return 0;
98       break;
99     case '?': // unknown option
100       usage(progname);
101       return 0;
102       break;
103     }
104   }
105
106   // 入力ファイルが存在するかどうか
107   if( !(out_comb) ){
108     usage(progname);
109     return 0;
110   }
111
112   // 画面出力、ログ出力の整理
113   if(pflagv==1) pflag =1;
114   if(pflag ==0) pflagv=0;
115   lflag=0;
116   lflagv=0;
117   if(out_log){
118     lflag=pflag;
119     lflagv=pflagv;
120   }
121
122   // ####################################################################
123   // InputParam インスタンス
124   InputParam InputCntl;
125   // InputParam に cpm_ParaManager をセット
126   if( !InputCntl.importCPM(paraMngr) ) {
127    return CPM_ERROR_PM_INSTANCE;
128   }
129
130   // ####################################################################
131   // 入力ファイルの読込み
132   cout << endl;
133   cout << "Input Parameter File Read" << endl;
134   t0 = cpm_Base::GetWTime();
135   if( !InputCntl.Read(fname) ) {
136     cout << "Input Parameter File Read Error" << endl;
137     return 0;
138   };
139
140   // ####################################################################
141   // conv のインスタンス
142   CONV* conv = CONV::ConvInit(&InputCntl);
143   if( conv == NULL ) return 0;
144   if ( !conv->importCPM(paraMngr) )
145   {
146     return CPM_ERROR_PM_INSTANCE;
147   }
148
```

```
149   // ####################################################################
150   // conv に InputParam クラスポインタをコピー
151   if( !conv->importInputParam(&InputCntl) ) {
152     cout << "InputParam pointer is NULL" << endl;
153   }
154
155   // ####################################################################
156   // ログファイルのオープン
157   int m_lflag=lflag;
158   LOG_OUT_ conv->OpenLogFile();
159
160   // ####################################################################
161   // 出力指定ディレクトリのチェック
162   if( dname.size() > 0 ) conv->CheckDir(dname);
163   conv->CheckDir(InputCntl.Get_OutputDir());
164   //if( dname.size() != 0 ) dname = dname + "/";
165
166   // ####################################################################
167   // 引数のセット
168   conv->m_pflag=pflag;
169   conv->m_pflagv=pflagv;
170   conv->m_lflag=lflag;
171   conv->m_lflagv=lflagv;
172
173   // ####################################################################
174   // dfi ファイルの読み込み
175   cout << endl;
176   cout << "ReadDfiFiles" << endl;
177   t1 = cpm_Base::GetWTime();
178   conv->ReadDfiFiles();
179
180   // ####################################################################
181   // コンバートデータのエラーチェック
182   if( !conv->CheckConvData() ) return 0;
183
184   // ####################################################################
185   // Converter
186   //
187   // Mx1 の処理
188   if( InputCntl.Get_ConvType() == E_OUTPUT_Mx1 ) {
189     cout << endl;
190     cout << "Convert M x 1 " << endl;
191   // MxM の処理
192   } else if ( InputCntl.Get_ConvType() == E_OUTPUT_MxM ) {
193     cout << endl;
194     cout << "Convert M x M " << endl;
195   // MxN の処理
196   } else if ( InputCntl.Get_ConvType() == E_OUTPUT_MxN ) {
197     cout << endl;
198     cout << "Convert M x N " << endl;
199   }
200
201   t2 = cpm_Base::GetWTime();
202   // VoxelInit
203   conv->VoxelInit();
204   // 実行
205   if( !conv->exec() ) return 0;
206
207   // ####################################################################
208   // 終了処理
209   cout << endl;
210   cout << "converter finish" << endl;
211   t3 = cpm_Base::GetWTime();
212
213   double tt[4];
214   tt[0]=t1-t0;
215   tt[1]=t2-t1;
216   tt[2]=t3-t2;
217   tt[3]=t3-t0;
218
219   printf("\n\n");
220   printf("TIME : ReadInit      %10.3f sec.\n", tt[0]);
221   printf("TIME : ReadDfiFiles  %10.3f sec.\n", tt[1]);
222   printf("TIME : ConvertFiles  %10.3f sec.\n", tt[2]);
223   printf("TIME : Total Time    %10.3f sec.\n", tt[3]);
224   LOG_OUT_ conv->WriteTime(tt);
225
226   // ####################################################################
227   // ログファイルのクローズ
228   LOG_OUT_ conv->CloseLogFile();
229
230   // ####################################################################
231   // 並列環境の終了
232
233   return 0;
234 }
```

5.27.2.2 void usage (const char * *progname*)

main.C の 21 行で定義されています。

参照元 main().

```
22 {
23   std::cerr
24     << "Usage: " << progname << " <option> filename <options>\n"
25     << " filename: file name when -f specified\n"
26     << " Options:\n"
27     << "  -f filename  : input file name for combine (ex : comb.tp)\n"
28     << "  -d dirname   : output directory name for combine (this option is given priority over input file)
     \n"
29     << "  -v verbose   : print more info\n"
30     << "  -l log out   : print out logfile\n"
31     << "  -s thin out  : thin out option\n"
32     << "  -h           : Show usage and exit\n"
33     << std::endl;
34 }
```

Index