FCONV (File Converter)

0.9

作成： Doxygen 1.8.4

Wed Feb 5 2014 15:36:33

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。

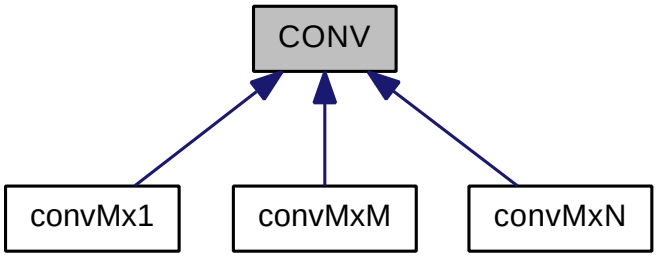
Chapter 4

クラス

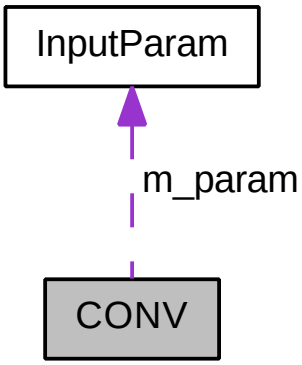
4.1 クラス CONV

`#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
- CIO::E_CIO_ERRORCODE ReadDfiFiles ()

  InputParam のポインタをコピー
- bool CheckDFIdata ()

  dfi 毎の成分数、出力ガイドセルのチェックと更新
- 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 PrintDFI (FILE *fp)

  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 T >
  bool copyArray (cio_TypeArray< T > *B, cio_Array *&src, int sta[3], int end[3], int n)

  配列のコピー
- 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 T >
  CONV_INLINE bool copyArray (cio_TypeArray< T > *B, cio_Array *&src, int sta[3], int end[3], int n)
- 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 *param)

  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_param

  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 LOG_OUT_, m_fplog, と 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   LOG_OUT_ fclose(m_fplog);
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 192 行で定義されています。

```
195 {
196
197   if( src == NULL ) return false;
198
199   double CompVal;
200
201   //size の取得
202   const int *sz = src->getArraySizeInt();
203   //配列形状の取得
204   CIO::E_CIO_ARRAYSHAPE shape = src->getArrayShape();
205   //成分数の取得
206   int nComp = src->getNcomp();
207
208
209   if( nComp > 1 ) {
210     //nijk の処理
211     if( shape == CIO::E_CIO_NIJK ) {
212       for(int k=0; k<sz[2]; k++) {
213       for(int j=0; j<sz[1]; j++) {
214       for(int i=0; i<sz[0]; i++) {
215         CompVal=(double)0.0;
216         for(int n=0; n<nComp; n++) {
217           if( min[n] > (double)src->val(n,i,j,k) ) min[n] = (double)src->val(n,i,j,k);
```

```
218           if( max[n] < (double)src->val(n,i,j,k) ) max[n] = (double)src->val(n,i,j,k);
219           CompVal = CompVal + (double)src->val(n,i,j,k)*(double)src->val(n,i,j,k);
220         }
221         CompVal = sqrt(CompVal);
222         if( min[nComp] > CompVal ) min[nComp]=CompVal;
223         if( max[nComp] < CompVal ) max[nComp]=CompVal;
224       }}}
225
226     }
227     else if( shape == CIO::E_CIO_IJKN )
228     //ijkn の処理
229     {
230       for(int k=0; k<sz[2]; k++) {
231       for(int j=0; j<sz[1]; j++) {
232       for(int i=0; i<sz[0]; i++) {
233         CompVal=(double)0.0;
234         for(int n=0; n<nComp; n++) {
235           if( min[n] > (double)src->val(i,j,k,n) ) min[n] = (double)src->val(i,j,k,n);
236           if( max[n] < (double)src->val(i,j,k,n) ) max[n] = (double)src->val(i,j,k,n);
237           CompVal = CompVal + (double)src->val(i,j,k,n)*(double)src->val(i,j,k,n);
238         }
239         CompVal = sqrt(CompVal);
240         if( min[nComp] > CompVal ) min[nComp]=CompVal;
241         if( max[nComp] < CompVal ) max[nComp]=CompVal;
242       }}}
243     } else return false;
244
245   } else {
246     //nijk の処理
247     if( shape == CIO::E_CIO_NIJK ) {
248       for(int k=0; k<sz[2]; k++) {
249       for(int j=0; j<sz[1]; j++) {
250       for(int i=0; i<sz[0]; i++) {
251         if( min[0] > (double)src->val(0,i,j,k) ) min[0] = (double)src->val(0,i,j,k);
252         if( max[0] < (double)src->val(0,i,j,k) ) max[0] = (double)src->val(0,i,j,k);
253       }}}
254     }
255     else if( shape == CIO::E_CIO_IJKN )
256     //ijkn の処理
257     {
258       for(int k=0; k<sz[2]; k++) {
259       for(int j=0; j<sz[1]; j++) {
260       for(int i=0; i<sz[0]; i++) {
261         if( min[0] > (double)src->val(i,j,k,0) ) min[0] = (double)src->val(i,j,k,0);
262         if( max[0] < (double)src->val(i,j,k,0) ) max[0] = (double)src->val(i,j,k,0);
263       }}}
264     } else return false;
265
266   }
267   return true;
268
269 }
```

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::CheckDFIdata ( )

dfi 毎の成分数、出力ガイドセルのチェックと更新

戻り値

エラーコード

conv.C の 123 行で定義されています。

参照先 E_CONV_OUTPUT_MxN, InputParam::Get_ConvType(), InputParam::Get_CropIndexEnd(), InputParam::Get_CropIndexEnd_on(), InputParam::Get_CropIndexStart(), InputParam::Get_CropIndexStart_on(), InputParam::Get_OutputFormat(), InputParam::Get_OutputGuideCell(), m_in_dfi, m_param, InputParam::Set_CropIndexEnd(), InputParam::Set_CropIndexStart(), と InputParam::Set_OutputGuideCell().

参照元 ReadDfiFiles().

```
124 {
125
126   bool ierr=true;
127   bool upsta=true;
128   bool upend=true;
129   int *sta;
130   int *end;
131
132   for( int i=0; i<m_in_dfi.size(); i++) {
133     //コンバート成分数のチェック
134     if( m_param->Get_OutputFormat() == CIO::E_CIO_FMT_SPH ) {
135       if( m_in_dfi[i]->GetNumComponent() > 3 ) {
136         printf("\tCan't Converter OutputFormat.\n");
137         ierr=false;
138       }
139     }
140
141     //ガイドセル数のチェックと更新
142     if( m_param->Get_OutputGuideCell() > 0 ) {
143       if( m_in_dfi[i]->GetNumGuideCell() < m_param->Get_OutputGuideCell() ) {
144         m_param->Set_OutputGuideCell(m_in_dfi[i]->GetNumGuideCell());
145         printf("\tupdate OutputGudeCell %d\n",m_param->Get_OutputGuideCell() );
146       }
147     }
148
149     //入力指示範囲のチェック
150     if( m_param->Get_CropIndexStart_on() ) {
151       sta = m_param->Get_CropIndexStart();
152       for(int j=0; j<3; j++) {
153         if( sta[j]<1 ) {
154           sta[j]=1;
155           upsta=false;
156         }
157       }
158     }
159     if( m_param->Get_CropIndexEnd_on() ) {
160       const cio_Domain* DFI_Domian = m_in_dfi[i]->GetcioDomain();
161       end = m_param->Get_CropIndexEnd();
162       for(int j=0; j<3; j++) {
163         if( end[j]>DFI_Domian->GlobalVoxel[j] ) {
164           end[j]=DFI_Domian->GlobalVoxel[j];
165           upend=false;
166         }
167       }
168     }
169
170     //Prefix の重複チェック
171     const cio_FileInfo* Finfo1 = m_in_dfi[i]->GetcioFileInfo();
172     for(int j=i+1; j<m_in_dfi.size(); j++) {
173       const cio_FileInfo* Finfo2 = m_in_dfi[j]->GetcioFileInfo();
174       if( Finfo1->Prefix == Finfo2->Prefix ) {
175         printf("\tCan't duplicate Prefix \"%s\" dfi : %s dfi : %s\n",
176                 Finfo1->Prefix.c_str(),m_in_dfi[i]->get_dfi_fname().c_str(),
177                 m_in_dfi[j]->get_dfi_fname().c_str());
178         ierr=false;
179       }
180     }
181
182     //voxel size のチェック
183     if( m_param->Get_ConvType() == E_CONV_OUTPUT_MxN ) {
184       const cio_Domain* domain1 = m_in_dfi[i]->GetcioDomain();
185       for(int j=i+1; j<m_in_dfi.size(); j++) {
186         const cio_Domain* domain2 = m_in_dfi[j]->GetcioDomain();
187         if( domain1->GlobalVoxel[0] != domain2->GlobalVoxel[0] ||
188             domain1->GlobalVoxel[1] != domain2->GlobalVoxel[1] ||
189             domain1->GlobalVoxel[2] != domain2->GlobalVoxel[2]   ) {
190           printf("\tCan't mismatch ClobalVoxel size %s (%d,%d,%d) \n"
191                  "\t                                %s (%d,%d,%d) \n",
192             m_in_dfi[i]->get_dfi_fname().c_str(),
193             domain1->GlobalVoxel[0],domain1->GlobalVoxel[1],domain1->GlobalVoxel[2],
194             m_in_dfi[j]->get_dfi_fname().c_str(),
195             domain2->GlobalVoxel[0],domain2->GlobalVoxel[1],domain2->GlobalVoxel[2]);
196
197           ierr=false;
198         }
199       }
200     }
201   }
```

```
202
203   //入力指示範囲の更新
204   if( !upsta ) {
205     m_param->Set_CropIndexStart(sta);
206     printf("\tupdate input CropIndexStart : %d %d %d\n",sta[0],sta[1],sta[2]);
207   }
208   if( !upend ) {
209     m_param->Set_CropIndexEnd(end);
210     printf("\tupdate input CropIndexEnd   : %d %d %d\n",end[0],end[1],end[2]);
211   }
212
213   return ierr;
214
215 }
```

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 219 行で定義されています。

参照先 Exit, と Hostonly_.

参照元 main().

```
220 {
221   Hostonly_
222   {
223
224 #ifndef _WIN32
225
226     if( dirstr.size() == 0 ) {
227       //printf("\toutput current directory\n");
228       return;
229     }
230
231     DIR* dir;
232     if( !(dir = opendir(dirstr.c_str())) ) {
233       if( errno == ENOENT ) {
234         mode_t mode = S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH;
235         if ( cio_DFI::MakeDirectorySub(dirstr) != 0 )
236         {
237           printf("\tCan't generate directory(%s).\n", dirstr.c_str());
238           Exit(0);
239         }
240       }
241       else {
242         printf("Directory open error.(%s)", dirstr.c_str());
243         Exit(0);
244       }
245     }
246     else {
247       if( closedir(dir) == -1 ) {
248         printf("Directory close error.(%s)", dirstr.c_str());
249         Exit(0);
250       }
251     }
252
253 #else // for windows
254
255     if( dirstr.size() == 0 ) {
256       printf("\toutput current directory\n");
257       return;
258     }
259
260     // check to exist directory
261     if (IsDirExsist(dirstr)) {
262       // exist directory
263       return;
264     }
265
266     // make directory
267     if(!CreateDirectory(dirstr.c_str(), NULL)){
268       printf("\tCan't generate directory(%s).\n", dirstr.c_str());
269       Exit(0);
270     }
```

```
271
272 #endif  // _WIN32
273
274   }
275
276   return;
277 }
```

4.1.3.5 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 624 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().

```
630 {
631
632   //debug
633   const int *tmp = src->getHeadIndex();
634
635   //copy
636   int gcB          = buf->getGcInt();
637   const int *headB = buf->getHeadIndex();
638   const int *tailB = buf->getTailIndex();
639   int       gcS    = src->getGcInt();
640   int sta[3],end[3];
641   for( int i=0;i<3;i++ )
642   {
643     sta[i] = (headB[i]-gcB>=headS[i]-gcS) ? headB[i]-gcB : headS[i]-gcS;
644     end[i] = (tailB[i]+gcB<=tailS[i]+gcS) ? tailB[i]+gcB : tailS[i]+gcS;
645   }
646
647   CIO::E_CIO_DTYPE buf_dtype =  buf->getDataType();
648
649   //uint8
650   if( buf_dtype == CIO::E_CIO_UINT8 ) {
651     cio_TypeArray<unsigned char> *B = dynamic_cast<cio_TypeArray<unsigned char>*>(buf);
652     return copyArray(B, src, sta, end, n);
653   }
654   //int8
655   else if( buf_dtype == CIO::E_CIO_INT8 ) {
656     cio_TypeArray<char> *B = dynamic_cast<cio_TypeArray<char>*>(buf);
657     return copyArray(B, src, sta, end, n);
658   }
659   //uint16
660   else if( buf_dtype == CIO::E_CIO_UINT16 ) {
661     cio_TypeArray<unsigned short> *B = dynamic_cast<cio_TypeArray<unsigned short>*>(buf);
662     return copyArray(B, src, sta, end, n);
663   }
664   //int16
665   else if( buf_dtype == CIO::E_CIO_INT16 ) {
666     cio_TypeArray<short> *B = dynamic_cast<cio_TypeArray<short>*>(buf);
667     return copyArray(B, src, sta, end, n);
668   }
669   //uint32
670   else if( buf_dtype == CIO::E_CIO_UINT32 ) {
671     cio_TypeArray<unsigned int> *B = dynamic_cast<cio_TypeArray<unsigned int>*>(buf);
672     return copyArray(B, src, sta, end, n);
673   }
674   //int32
675   else if( buf_dtype == CIO::E_CIO_INT32 ) {
676     cio_TypeArray<int> *B = dynamic_cast<cio_TypeArray<int>*>(buf);
677     return copyArray(B, src, sta, end, n);
678   }
679   //uint64
680   else if( buf_dtype == CIO::E_CIO_UINT64 ) {
681     cio_TypeArray<unsigned long long> *B = dynamic_cast<cio_TypeArray<unsigned long long>*>(buf);
```

```
682     return copyArray(B, src, sta, end, n);
683   }
684   //int64
685   else if( buf_dtype == CIO::E_CIO_INT64 ) {
686     cio_TypeArray<long long> *B = dynamic_cast<cio_TypeArray<long long>*>(buf);
687     return copyArray(B, src, sta, end, n);
688   }
689   //float32
690   else if( buf_dtype == CIO::E_CIO_FLOAT32 ) {
691     cio_TypeArray<float> *B = dynamic_cast<cio_TypeArray<float>*>(buf);
692     return copyArray(B, src, sta, end, n);
693   }
694   //float64
695   else if( buf_dtype == CIO::E_CIO_FLOAT64 ) {
696     cio_TypeArray<double> *B = dynamic_cast<cio_TypeArray<double>*>(buf);
697     return copyArray(B, src, sta, end, n);
698   }
699
700   return false;
701 }
```

4.1.3.6 CONV * CONV::ConvInit (InputParam * *param*) [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_CONV_OUTPUT_Mx1, E_CONV_OUTPUT_MxM, E_CONV_OUTPUT_MxN, InputParam::Get_ConvType(), InputParam::Get_OutputFormat(), m_bgrid_interp_flag, と m_param.

参照元 main().

```
56 {
57   CONV* conv = NULL;
58
59   if(param->Get_ConvType() == E_CONV_OUTPUT_Mx1 ) {
60     conv = new convMx1();
61   } else if( param->Get_ConvType() == E_CONV_OUTPUT_MxM ) {
62     conv = new convMxM();
63   } else if( param->Get_ConvType() == E_CONV_OUTPUT_MxN ) {
64     conv = new convMxN();
65   }
66
67   //InputParam のポインタをセット
68   conv->m_param = param;
69
70   //格子点補間フラグのセット
71   if( param->Get_OutputFormat() == CIO::E_CIO_FMT_PLOT3D ||
72       param->Get_OutputFormat() == CIO::E_CIO_FMT_AVS ||
73       param->Get_OutputFormat() == CIO::E_CIO_FMT_VTK ) {
74     conv->m_bgrid_interp_flag = true;
75   } else {
76     conv->m_bgrid_interp_flag = false;
77   }
78
79   return conv;
80 }
```

4.1.3.7 template<class T > CONV_INLINE bool CONV::copyArray (cio_TypeArray< T > * *B*, cio_Array *& *src*, int *sta[3]*, int *end[3]*, int *n*)

conv_inline.h の 30 行で定義されています。

参照先 copyArray().

```
35 {
36
37   CIO::E_CIO_DTYPE src_dtype =  src->getDataType();
38
39   //uint8
```

```
40   if( src_dtype == CIO::E_CIO_UINT8 ) {
41     cio_TypeArray<unsigned char> *S = dynamic_cast<cio_TypeArray<unsigned char>*>(src);
42     return copyArray(B,S,sta,end,n);
43   }
44   //int8
45   else if( src_dtype== CIO::E_CIO_INT8 ) {
46     cio_TypeArray<char> *S = dynamic_cast<cio_TypeArray<char>*>(src);
47     return copyArray(B,S,sta,end,n);
48   }
49   //uint16
50   else if( src_dtype== CIO::E_CIO_UINT16 ) {
51     cio_TypeArray<unsigned short> *S = dynamic_cast<cio_TypeArray<unsigned short>*>(src);
52     return copyArray(B,S,sta,end,n);
53   }
54   //int16
55   else if( src_dtype== CIO::E_CIO_INT16 ) {
56     cio_TypeArray<short> *S = dynamic_cast<cio_TypeArray<short>*>(src);
57     return copyArray(B,S,sta,end,n);
58   }
59   //uint32
60   else if( src_dtype== CIO::E_CIO_UINT32 ) {
61     cio_TypeArray<unsigned int> *S = dynamic_cast<cio_TypeArray<unsigned int>*>(src);
62     return copyArray(B,S,sta,end,n);
63   }
64   //int32
65   else if( src_dtype== CIO::E_CIO_INT32 ) {
66     cio_TypeArray<int> *S = dynamic_cast<cio_TypeArray<int>*>(src);
67     return copyArray(B,S,sta,end,n);
68   }
69   //uint64
70   else if( src_dtype== CIO::E_CIO_UINT64 ) {
71     cio_TypeArray<unsigned long long> *S = dynamic_cast<cio_TypeArray<unsigned long long>*>(src);
72     return copyArray(B,S,sta,end,n);
73   }
74   //int64
75   else if( src_dtype== CIO::E_CIO_INT64 ) {
76     cio_TypeArray<long long> *S = dynamic_cast<cio_TypeArray<long long>*>(src);
77     return copyArray(B,S,sta,end,n);
78   }
79   //float32
80   else if( src_dtype== CIO::E_CIO_FLOAT32 ) {
81     cio_TypeArray<float> *S = dynamic_cast<cio_TypeArray<float>*>(src);
82     return copyArray(B,S,sta,end,n);
83   }
84   else if( src_dtype== CIO::E_CIO_FLOAT64 ) {
85     cio_TypeArray<double> *S = dynamic_cast<cio_TypeArray<double>*>(src);
86     return copyArray(B,S,sta,end,n);
87   }
88
89   return false;
90
91 }
```

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 97 行で定義されています。

参照先 InputParam::Get_CropIndexStart(), InputParam::Get_CropIndexStart_on(), InputParam::Get_ThinOut(), と m_param.

```
102 {
103
104   //配列形状
105   CIO::E_CIO_ARRAYSHAPE buf_shape = buf->getArrayShape();
106
107   CIO::E_CIO_ARRAYSHAPE src_shape = src->getArrayShape();
108
109   //入力指示領域の取得
110   int IndexStart[3];
111   if( m_param->Get_CropIndexStart_on() ) {
112     const int *cropIndexStart = m_param->Get_CropIndexStart();
113     for(int i=0; i<3; i++) IndexStart[i]=cropIndexStart[i];
114   } else {
115     //for(int i=0; i<3; i++) IndexStart[i]=sta[i]+1;
116     for(int i=0; i<3; i++) IndexStart[i]=1;
117   }
118
119   const int* headS = src->getHeadIndex();
120
121   //間引き数の取得
```

```
122   int thin_count = m_param->Get_ThinOut();
123
124   //IJKN&IJKN
125   if( buf_shape == CIO::E_CIO_IJKN && src_shape == CIO::E_CIO_IJKN )
126   {
127     for( int k=sta[2];k<=end[2];k++ ){
128     if( (k-(IndexStart[2]-1))%thin_count != 0 ) continue;
129
130     for( int j=sta[1];j<=end[1];j++ ){
131     if( (j-(IndexStart[1]-1))%thin_count != 0 ) continue;
132
133     for( int i=sta[0];i<=end[0];i++ ){
134       if( (i-(IndexStart[0]-1))%thin_count != 0 ) continue;
135
136       src->hval(i/thin_count,j/thin_count,k/thin_count,n) = buf->hval(i,j,k,n);
137     }}}
138   }
139   else if( buf_shape == CIO::E_CIO_NIJK && src_shape == CIO::E_CIO_NIJK )
140   //NIJK&NIJK
141   {
142     for( int k=sta[2];k<=end[2];k++ ){
143     if( (k-(IndexStart[2]-1))%thin_count != 0 ) continue;
144
145     for( int j=sta[1];j<=end[1];j++ ){
146     if( (j-(IndexStart[1]-1))%thin_count != 0 ) continue;
147
148     for( int i=sta[0];i<=end[0];i++ ){
149       if( (i-(IndexStart[0]-1))%thin_count != 0 ) continue;
150
151       src->hval(n,i/thin_count,j/thin_count,k/thin_count) = buf->hval(n,i,j,k);
152     }}}
153   }
154   else if( buf_shape == CIO::E_CIO_IJKN && src_shape == CIO::E_CIO_NIJK )
155   //IJNK&NIJK
156   {
157
158     for( int k=sta[2];k<=end[2];k++ ){
159     if( (k-(IndexStart[2]-1))%thin_count != 0 ) continue;
160     for( int j=sta[1];j<=end[1];j++ ){
161     if( (j-(IndexStart[1]-1))%thin_count != 0 ) continue;
162     for( int i=sta[0];i<=end[0];i++ ){
163       if( (i-(IndexStart[0]-1))%thin_count != 0 ) continue;
164
165       src->hval(n,i/thin_count,j/thin_count,k/thin_count) = buf->hval(i,j,k,n);
166
167     }}}
168   }
169   else if( buf_shape == CIO::E_CIO_NIJK && src_shape == CIO::E_CIO_IJKN )
170   //NIJK&IJKN
171   {
172     for( int k=sta[2];k<=end[2];k++ ){
173     if( (k-(IndexStart[2]-1))%thin_count != 0 ) continue;
174
175     for( int j=sta[1];j<=end[1];j++ ){
176     if( (j-(IndexStart[1]-1))%thin_count != 0 ) continue;
177
178     for( int i=sta[0];i<=end[0];i++ ){
179       if( (i-(IndexStart[0]-1))%thin_count != 0 ) continue;
180
181       src->hval(i/thin_count,j/thin_count,k/thin_count,n) = buf->hval(n,i,j,k);
182     }}}
183   }
184
185   return true;
186 }
```

4.1.3.9 template<class T > bool CONV::copyArray (cio_TypeArray< T > * *B*, cio_Array *& *src*, int *sta[3]*, int *end[3]*, int *n*)

配列のコピー

引数

in	*B*	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(), と copyArray().

4.1.3.10 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

4.1.3.11 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 833 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().

```
836 {
837   CIO::E_CIO_DTYPE d_type = src->getDataType();
838   if( d_type == CIO::E_CIO_UINT8 ) {
839     cio_TypeArray<unsigned char> *data = dynamic_cast<cio_TypeArray<unsigned char>*>(src);
840     if( !calcMinMax(data,min,max) ) return false;
841   } else if( d_type == CIO::E_CIO_INT8 ) {
842     cio_TypeArray<char> *data = dynamic_cast<cio_TypeArray<char>*>(src);
843     if( !calcMinMax(data,min,max) ) return false;
844   } else if( d_type == CIO::E_CIO_UINT16 ) {
845     cio_TypeArray<unsigned short> *data = dynamic_cast<cio_TypeArray<unsigned short>*>(src);
846     if( !calcMinMax(data,min,max) ) return false;
847   } else if( d_type == CIO::E_CIO_INT16 ) {
848     cio_TypeArray<short> *data = dynamic_cast<cio_TypeArray<short>*>(src);
849     if( !calcMinMax(data,min,max) ) return false;
850   } else if( d_type == CIO::E_CIO_UINT32 ) {
851     cio_TypeArray<unsigned int> *data = dynamic_cast<cio_TypeArray<unsigned int>*>(src);
852     if( !calcMinMax(data,min,max) ) return false;
853   } else if( d_type == CIO::E_CIO_INT32 ) {
854     cio_TypeArray<int> *data = dynamic_cast<cio_TypeArray<int>*>(src);
855     if( !calcMinMax(data,min,max) ) return false;
856   } else if( d_type == CIO::E_CIO_UINT64 ) {
857     cio_TypeArray<unsigned long long> *data = dynamic_cast<cio_TypeArray<unsigned long long>*>(src);
858     if( !calcMinMax(data,min,max) ) return false;
859   } else if( d_type == CIO::E_CIO_INT64 ) {
860     cio_TypeArray<long long> *data = dynamic_cast<cio_TypeArray<long long>*>(src);
861     if( !calcMinMax(data,min,max) ) return false;
862   } else if( d_type == CIO::E_CIO_FLOAT32 ) {
863     cio_TypeArray<float> *data = dynamic_cast<cio_TypeArray<float>*>(src);
864     if( !calcMinMax(data,min,max) ) return false;
865   } else if( d_type == CIO::E_CIO_FLOAT64 ) {
866     cio_TypeArray<double> *data = dynamic_cast<cio_TypeArray<double>*>(src);
867     if( !calcMinMax(data,min,max) ) return false;
868   }
869
870   return true;
871
872 }
```

4.1.3.12 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().

4.1.3.13 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 705 行で定義されています。

```
706 {
707
708   if     ( file_format_type == CIO::E_CIO_FMT_SPH ) return D_CIO_EXT_SPH;
709   else if( file_format_type == CIO::E_CIO_FMT_BOV ) return D_CIO_EXT_BOV;
710   else if( file_format_type == CIO::E_CIO_FMT_AVS ) return D_CIO_EXT_SPH;
711   else if( file_format_type == CIO::E_CIO_FMT_PLOT3D ) return D_CIO_EXT_FUNC;
712   else if( file_format_type == CIO::E_CIO_FMT_VTK ) return D_CIO_EXT_VTK;
713
714   return "";
715 }
```

4.1.3.14 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 610 行で定義されています。

```
611 {
612
613   const cio_TimeSlice* Tslice = dfi->GetcioTimeSlice();
614   for(int i=0; i<Tslice->SliceList.size(); i++) {
615     if( Tslice->SliceList[i].step == step ) return Tslice->SliceList[i].time;
616   }
617
618   return 0.0;
619
620 }
```

4.1.3.15 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   }
```

4.1.3.16 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 1004 行で定義されています。

参照先 InputParam::Get_CropIndexEnd(), InputParam::Get_CropIndexEnd_on(), InputParam::Get_CropIndexStart(), InputParam::Get_CropIndexStart_on(), InputParam::Get_ThinOut(), と m_param.

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

```
1009 {
1010
1011   //間引き数の取得
1012   int thin_count = m_param->Get_ThinOut();
1013
1014   //MPI 情報の生成
1015   out_mpi = new cio_MPI(numProc,0);
1016
1017   //Domain 情報の生成
1018   cio_Domain* dfi_domain = (cio_Domain *)dfi->GetcioDomain();
1019   double Gorigin[3];
1020   double Gregion[3];
1021   int Gvoxel[3];
1022   int Gdiv[3];
1023   int IndexStart[3];
1024   int IndexEnd[3];
1025   for(int i=0; i<3; i++) {
1026     Gorigin[i] = dfi_domain->GlobalOrigin[i];
1027     Gregion[i] = dfi_domain->GlobalRegion[i];
1028     Gvoxel[i]  = dfi_domain->GlobalVoxel[i];
1029     Gdiv[i]    = dfi_domain->GlobalDivision[i];
1030     IndexStart[i]=1;
1031     IndexEnd[i]=Gvoxel[i];
1032   }
1033
1034   //pit を計算
1035   double pit[3];
1036   for(int i=0; i<3; i++) pit[i]=Gregion[i]/(double)Gvoxel[i];
1037
1038   //入力領域指示ありのときボクセルサイズを更新
1039   if( m_param->Get_CropIndexStart_on() ) {
1040     const int* Start=m_param->Get_CropIndexStart();
1041     for(int i=0; i<3; i++) IndexStart[i]=Start[i];
1042     Gvoxel[0]=Gvoxel[0]-IndexStart[0]+1;
1043     Gvoxel[1]=Gvoxel[1]-IndexStart[1]+1;
1044     Gvoxel[2]=Gvoxel[2]-IndexStart[2]+1;
1045   }
1046   if( m_param->Get_CropIndexEnd_on() ) {
1047     const int* End=m_param->Get_CropIndexEnd();
1048     for(int i=0; i<3; i++) IndexEnd[i]=End[i];
1049     Gvoxel[0]=Gvoxel[0]-(dfi_domain->GlobalVoxel[0]-IndexEnd[0]);
1050     Gvoxel[1]=Gvoxel[1]-(dfi_domain->GlobalVoxel[1]-IndexEnd[1]);
1051     Gvoxel[2]=Gvoxel[2]-(dfi_domain->GlobalVoxel[2]-IndexEnd[2]);
1052   }
```

```
1053
1054   //Gregion の更新
1055   for(int i=0; i<3; i++) Gregion[i]=(double)Gvoxel[i]*pit[i];
1056
1057   //間引きありのときボクセルサイズを更新
1058   if( thin_count > 1 ) {
1059     for(int i=0; i<3; i++) {
1060       if( Gvoxel[i]%thin_count != 0 ) Gvoxel[i]=Gvoxel[i]/thin_count+1;
1061       else                            Gvoxel[i]=Gvoxel[i]/thin_count;
1062     }
1063   }
1064   //numProc が 1 のとき（Mx1 のとき）GlobalDivision を 1 にする
1065   if( numProc == 1 ) for(int i=0; i<3; i++) Gdiv[i]=1;
1066
1067   //out_domain の生成
1068   out_domain = new cio_Domain(Gorigin,Gregion,Gvoxel,Gdiv);
1069
1070   //Process 情報の生成
1071   const cio_Process* dfi_Process = dfi->GetcioProcess();
1072   out_process = new cio_Process();
1073   cio_Rank rank;
1074   if( numProc == dfi_Process->RankList.size() ) {
1075     for(int i=0; i<numProc; i++) {
1076       rank.RankID = dfi_Process->RankList[i].RankID;
1077       for(int j=0; j<3; j++) {
1078         rank.VoxelSize[j]=dfi_Process->RankList[i].VoxelSize[j];
1079         rank.HeadIndex[j]=dfi_Process->RankList[i].HeadIndex[j];
1080         rank.TailIndex[j]=dfi_Process->RankList[i].TailIndex[j];
1081       }
1082       if( thin_count > 1 ) {
1083         for(int j=0; j<3; j++) {
1084           if( rank.VoxelSize[j]%thin_count != 0 ) rank.VoxelSize[j]=rank.VoxelSize[j]/thin_count+1;
1085           else                                    rank.VoxelSize[j]=rank.VoxelSize[j]/thin_count;
1086           if( (rank.HeadIndex[j]-1)%thin_count != 0 ) {
1087              rank.HeadIndex[j]=(rank.HeadIndex[j]-1)/thin_count+1;
1088           } else {
1089              rank.HeadIndex[j]=(rank.HeadIndex[j]-1)/thin_count;
1090           }
1091           rank.HeadIndex[j]=rank.HeadIndex[j]+1;
1092           rank.TailIndex[j]=rank.HeadIndex[j]+rank.VoxelSize[j]-1;
1093         }
1094       }
1095       out_process->RankList.push_back(rank);
1096     }
1097   } else if( numProc == 1 ) {
1098     rank.RankID=0;
1099     for(int i=0; i<3; i++) {
1100       rank.VoxelSize[i]=Gvoxel[i];
1101       //rank.HeadIndex[i]=IndexStart[i];
1102       rank.HeadIndex[i]=1;
1103       rank.TailIndex[i]=rank.HeadIndex[i]+Gvoxel[i]-1;
1104     }
1105     out_process->RankList.push_back(rank);
1106   }
1107
1108   return true;
1109 }
```

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*	rank 基準のリスト

conv.C の 776 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().

```
777 {
778
779   //総 rank 数を求める
780   int Total_rank = 0;
781   for(int i=0; i<m_in_dfi.size(); i++){
782     const cio_Process* DFI_Process = m_in_dfi[i]->GetcioProcess();
783     Total_rank+=DFI_Process->RankList.size();
```

```
784   }
785
786   //自ランクで担当する rank 数を求める
787   int nRank = Total_rank/m_numProc;
788   if( Total_rank%m_numProc != 0 ) {
789     for(int i=0; i<Total_rank%m_numProc; i++) {
790       if( m_myRank == i ) nRank++;
791     }
792   }
793
794   //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
795   int sta,end;
796   sta = m_myRank * nRank;
797   if( Total_rank%m_numProc != 0 ) {
798     if( m_myRank >= Total_rank%m_numProc ) sta = sta+Total_rank%m_numProc;
799   }
800   end = sta+nRank-1;
801
802   //処理 rank リストの生成
803   int cnt=0;
804   for(int i=0; i<m_in_dfi.size(); i++) {
805     step_rank_info info;
806     info.rankStart = -1;
807     const cio_Process* DFI_Process = m_in_dfi[i]->GetcioProcess();
808     for(int j=0; j<DFI_Process->RankList.size(); j++) {
809       if( sta > cnt ) { cnt++; continue; }
810       if( info.rankStart == -1 ) {
811         info.dfi = m_in_dfi[i];
812         info.rankStart = j;
813       }
814       info.rankEnd = j;
815       cnt++;
816       if( end < cnt ) break;
817     }
818     if( info.rankStart > -1 ) StepRankList.push_back(info);
819     if( end < cnt ) break;
820   }
821
822   //stepStart,stepEnd のセット
823   for(int i=0; i<StepRankList.size(); i++) {
824     const cio_TimeSlice* TSlice = StepRankList[i].dfi->GetcioTimeSlice();
825     StepRankList[i].stepStart=0;
826     StepRankList[i].stepEnd=TSlice->SliceList.size()-1;
827   }
828
829 }
```

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 719 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().

```
720 {
721
722   //総ステップ数を求める
723   int Total_step = 0;
724   for(int i=0; i<m_in_dfi.size(); i++){
725     const cio_TimeSlice* TSlice = m_in_dfi[i]->GetcioTimeSlice();
726     Total_step+=TSlice->SliceList.size();
727   }
728
729   //自ランクで担当するステップ数を求める
730   int nStep = Total_step/m_numProc;
731   if( Total_step%m_numProc != 0 ) {
732     for(int i=0; i<Total_step%m_numProc; i++) {
733       if( m_myRank == i ) nStep++;
734     }
735   }
736
737   //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
```

```
738   int sta,end;
739   sta = m_myRank * nStep;
740   if( Total_step%m_numProc != 0 ) {
741     if( m_myRank >= Total_step%m_numProc ) sta = sta+Total_step%m_numProc;
742   }
743   end = sta+nStep-1;
744
745   //処理ステップリストの生成
746   int cnt=0;
747   for(int i=0; i<m_in_dfi.size(); i++){
748     step_rank_info info;
749     info.stepStart = -1;
750     const cio_TimeSlice* TSlice = m_in_dfi[i]->GetcioTimeSlice();
751     for( int j=0; j<TSlice->SliceList.size(); j++) {
752
753       if( sta > cnt ) { cnt++; continue; }
754       if( info.stepStart == -1 ) {
755         info.dfi=m_in_dfi[i];
756         info.stepStart=j;
757       }
758       info.stepEnd = j;
759       cnt++;
760       if( end < cnt ) break;
761     }
762     if( info.stepStart > -1 ) StepRankList.push_back(info);
763     if( end < cnt ) break;
764   }
765
766   //rantStart,rankEnd のセット
767   for(int i=0; i<StepRankList.size(); i++) {
768     const cio_Process* DFI_Process = StepRankList[i].dfi->GetcioProcess();
769     StepRankList[i].rankStart=0;
770     StepRankList[i].rankEnd=DFI_Process->RankList.size()-1;
771   }
772 }
```

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 454 行で定義されています。

参照元 convMx1::exec().

```
455 {
456   const double mem = Memory;
457   const double KB = 1024.0;
458   const double MB = 1024.0*KB;
459   const double GB = 1024.0*MB;
460   const double TB = 1024.0*GB;
461   const double PB = 1024.0*TB;
462   const double factor = 1.05; // estimate 5% for addtional
463
464   // Global memory
465   fprintf (fp," MemorySize = ");
466   if ( mem > PB ) {
467     fprintf (fp,"%6.2f (PB)\n", mem / PB *factor);
468   }
469   else if ( mem > TB ) {
470     fprintf (fp,"%6.2f (TB)\n", mem / TB *factor);
471   }
472   else if ( mem > GB ) {
473     fprintf (fp,"%6.2f (GB)\n", mem / GB *factor);
474   }
475   else if ( mem > MB ) {
476     fprintf (fp,"%6.2f (MB)\n", mem / MB *factor);
477   }
478   else if ( mem > KB ) {
479     fprintf (fp,"%6.2f (KB)\n", mem / KB *factor);
480   }
481   else if ( mem <= KB ){
482     fprintf (fp,"%6.2f (B)\n", mem *factor);
483   }
484   else {
485     fprintf (fp,"Caution! Memory required : %d (Byte)", (int)(mem *factor) );
```

```
486   }
487
488   fflush(fp);
489 }
```

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 493 行で定義されています。

```
494 {
495   double mem;
496   const double KB = 1024.0;
497   const double MB = 1024.0*KB;
498   const double GB = 1024.0*MB;
499   const double TB = 1024.0*GB;
500   const double PB = 1024.0*TB;
501   const double factor = 1.05; // estimate 5% for addtional
502
503   fprintf (fp,"*** Required MemorySize ***");
504   fprintf (fp,"\n");
505
506   mem = sphMemory;
507   fprintf (fp,"  read SPH MemorySize = ");
508   if ( mem > PB ) {
509     fprintf (fp,"%6.2f (PB)", mem / PB *factor);
510   }
511   else if ( mem > TB ) {
512     fprintf (fp,"%6.2f (TB)", mem / TB *factor);
513   }
514   else if ( mem > GB ) {
515     fprintf (fp,"%6.2f (GB)", mem / GB *factor);
516   }
517   else if ( mem > MB ) {
518     fprintf (fp,"%6.2f (MB)", mem / MB *factor);
519   }
520   else if ( mem > KB ) {
521     fprintf (fp,"%6.2f (KB)", mem / KB *factor);
522   }
523   else if ( mem <= KB ){
524     fprintf (fp,"%6.2f (B)", mem *factor);
525   }
526   else {
527     fprintf (fp,"Caution! Memory required : %d (Byte)", (int)(mem *factor) );
528   }
529
530   mem = plot3dMemory;
531   fprintf (fp,"  write PLOT3D MemorySize = ");
532   if ( mem > PB ) {
533     fprintf (fp,"%6.2f (PB)", mem / PB *factor);
534   }
535   else if ( mem > TB ) {
536     fprintf (fp,"%6.2f (TB)", mem / TB *factor);
537   }
538   else if ( mem > GB ) {
539     fprintf (fp,"%6.2f (GB)", mem / GB *factor);
540   }
541   else if ( mem > MB ) {
542     fprintf (fp,"%6.2f (MB)", mem / MB *factor);
543   }
544   else if ( mem > KB ) {
545     fprintf (fp,"%6.2f (KB)", mem / KB *factor);
546   }
547   else if ( mem <= KB ){
548     fprintf (fp,"%6.2f (B)", mem *factor);
549   }
550   else {
551     fprintf (fp,"Caution! Memory required : %d (Byte)", (int)(mem *factor) );
```

```
552   }
553
554   mem = thinMemory;
555   fprintf (fp,"  write thin out MemorySize = ");
556   if ( mem > PB ) {
557     fprintf (fp,"%6.2f (PB)", mem / PB *factor);
558   }
559   else if ( mem > TB ) {
560     fprintf (fp,"%6.2f (TB)", mem / TB *factor);
561   }
562   else if ( mem > GB ) {
563     fprintf (fp,"%6.2f (GB)", mem / GB *factor);
564   }
565   else if ( mem > MB ) {
566     fprintf (fp,"%6.2f (MB)", mem / MB *factor);
567   }
568   else if ( mem > KB ) {
569     fprintf (fp,"%6.2f (KB)", mem / KB *factor);
570   }
571   else if ( mem <= KB ){
572     fprintf (fp,"%6.2f (B)", mem *factor);
573   }
574   else {
575     fprintf (fp,"Caution! Memory required : %d (Byte)", (int)(mem *factor) );
576   }
577
578   mem = TotalMemory;
579   fprintf (fp,"  TotalMemorySize = ");
580   if ( mem > PB ) {
581     fprintf (fp,"%6.2f (PB)", mem / PB *factor);
582   }
583   else if ( mem > TB ) {
584     fprintf (fp,"%6.2f (TB)", mem / TB *factor);
585   }
586   else if ( mem > GB ) {
587     fprintf (fp,"%6.2f (GB)", mem / GB *factor);
588   }
589   else if ( mem > MB ) {
590     fprintf (fp,"%6.2f (MB)", mem / MB *factor);
591   }
592   else if ( mem > KB ) {
593     fprintf (fp,"%6.2f (KB)", mem / KB *factor);
594   }
595   else if ( mem <= KB ){
596     fprintf (fp,"%6.2f (B)", mem *factor);
597   }
598   else {
599     fprintf (fp,"Caution! Memory required : %d (Byte)", (int)(mem *factor) );
600   }
601
602   fprintf (fp,"\n");
603   fprintf (fp,"\n");
604
605   fflush(fp);
606 }
```

4.1.3.21 void CONV::OpenLogFile ( )

ログファイルのオープン

conv.C の 281 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().

```
282 {
283   //log file open
284   string prefix="log_comb_id";
285   int len = prefix.size()+10;//+6+4
286   char* tmp = new char[len];
287   memset(tmp, 0, sizeof(char)*len);
288   sprintf(tmp, "%s_%06d.%s", prefix.c_str(), m_myRank, "txt");
289
290   std::string logname(tmp);
291   if ( tmp ) delete [] tmp;
292
293   if ( !(m_fplog = fopen(logname.c_str(), "w")) ){
294     printf("\tFile Open Error : '%s'\n",logname.c_str());
295     Exit(0);
296   }
297   fprintf(m_fplog,"####################\n",logname.c_str());
```

```
298   fprintf(m_fplog,"### log_comb.txt ###\n",logname.c_str());
299   fprintf(m_fplog,"####################\n",logname.c_str());
300   fprintf(m_fplog,"\n");
301
302   fprintf(m_fplog,"procGrp  = %d\n", m_procGrp);
303   fprintf(m_fplog,"myRank   = %d\n", m_myRank);
304   fprintf(m_fplog,"numProc  = %d\n", m_numProc);
305   fprintf(m_fplog,"HostName = %s\n", m_HostName.c_str());
306   fprintf(m_fplog,"\n");
307
308 }
```

4.1.3.22 void CONV::PrintDFI (FILE * *fp*)

ログファイルのクローズ

dfi のログ出力

conv.C の 321 行で定義されています。

参照先 InputParam::m_dfiList, と m_param.

参照元 ReadDfiFiles().

```
322 {
323
324   fprintf(fp,"\n");
325   fprintf(fp,"*** dfi file info ***\n");
326   fprintf(fp,"\n");
327   for(int i=0; i<m_param->m_dfiList.size(); i++) {
328     fprintf(fp,"\tDFI File Name : %s\n",m_param->m_dfiList[i].in_dfi_name.c_str());
329     cio_DFI *dfi = m_param->m_dfiList[i].in_dfi;
330     const cio_FileInfo *DFI_Info = dfi->GetcioFileInfo();
331     if( DFI_Info->DFIType == CIO::E_CIO_DFITYPE_CARTESIAN ) {
332       fprintf(fp,"\tDFI_Info->DFIType                  = \"Cartesian\"\n");
333     } else {
334       fprintf(fp,"\tDFI_Info->DFIType                  = \"\"\n");
335     }
336     fprintf(fp,"\tDFI_Info->DirectoryPath            = \"%s\"\n",DFI_Info->DirectoryPath.c_str());
337     if( DFI_Info->TimeSliceDirFlag == CIO::E_CIO_ON ) {
338       fprintf(fp,"\tDFI_Info->TimeSliceDirFlag         = \"on\"\n");
339     } else {
340       fprintf(fp,"\tDFI_Info->TimeSliceDirFlag         = \"off\"\n");
341     }
342     fprintf(fp,"\tDFI_Info->Prefix                   = \"%s\"\n",DFI_Info->Prefix.c_str());
343     fprintf(fp,"\tDFI_Info->FileFormat               = \"%s\2\n",
344             dfi->GetFileFormatString().c_str());
345     if( DFI_Info->FieldFilenameFormat == CIO::E_CIO_FNAME_RANK_STEP ) {
346       fprintf(fp,"\tDFI_Info->FieldFilenameFormat      = \"rank_step\"\n");
347     }else {
348       fprintf(fp,"\tDFI_Info->FieldFilenameFormat      = \"step_rank\"\n");
349     }
350     fprintf(fp,"\tDFI_Info->GuideCell                = %d\n",DFI_Info->GuideCell);
351     fprintf(fp,"\tDFI_Info->DataType                 = \"%s\"\n",
352             dfi->GetDataTypeString().c_str());
353     if( DFI_Info->Endian == CIO::E_CIO_LITTLE ) {
354       fprintf(fp,"\tDFI_Info->Endian                   = \"little\"\n");
355     }else if( DFI_Info->Endian == CIO::E_CIO_BIG ) {
356       fprintf(fp,"\tDFI_Info->Endian                   = \"big\"\n");
357     }else {
358       fprintf(fp,"\tDFI_Info->Endian                   = \"\"\n");
359     }
360     fprintf(fp,"\tDFI_Info->ArrayShape               = \"%s\"\n",
361             dfi->GetArrayShapeString().c_str());
362     fprintf(fp,"\tDFI_Info->Component                = %d\n",DFI_Info->Component);
363     for(int j=0; j<DFI_Info->ComponentVariable.size(); j++ ) {
364       fprintf(fp,"\t  DFI_Info->ComponentVariable[%d]        = %s\n",j,
365               DFI_Info->ComponentVariable[j].c_str());
366     }
367
368     const cio_MPI *DFI_MPI = dfi->GetcioMPI();
369     fprintf(fp,"\tDFI_MPI->NumberOfRank              = %d\n",DFI_MPI->NumberOfRank);
370     fprintf(fp,"\tDFI_MPI->NumberOfGroup             = %d\n",DFI_MPI->NumberOfGroup);
371
372     const cio_Domain *DFI_Domain = dfi->GetcioDomain();
373     fprintf(fp,"\tDFI_Domain->GlobalVoxel[0]         = %d\n",DFI_Domain->GlobalVoxel[0]);
374     fprintf(fp,"\tDFI_Domain->GlobalVoxel[1]         = %d\n",DFI_Domain->GlobalVoxel[1]);
375     fprintf(fp,"\tDFI_Domain->GlobalVoxel[2]         = %d\n",DFI_Domain->GlobalVoxel[2]);
376     fprintf(fp,"\tDFI_Domain->GlobalDivision[0]      = %d\n",DFI_Domain->GlobalDivision[0]);
377     fprintf(fp,"\tDFI_Domain->GlobalDivision[1]      = %d\n",DFI_Domain->GlobalDivision[1]);
378     fprintf(fp,"\tDFI_Domain->GlobalDivision[2]      = %d\n",DFI_Domain->GlobalDivision[2]);
```

```
379
380     const cio_Process *DFI_Process = dfi->GetcioProcess();
381     fprintf(fp,"\tDFI_Process->RankList.size()       = %d\n",DFI_Process->RankList.size());
382     for(int j=0; j<DFI_Process->RankList.size(); j++) {
383       fprintf(fp,"\t  DFI_Process->RankList[%d].RankID       = %d\n",j,
384               DFI_Process->RankList[j].RankID);
385       fprintf(fp,"\t  DFI_Process->RankList[%d].HostName     = %s\n",j,
386               DFI_Process->RankList[j].HostName.c_str());
387       fprintf(fp,"\t  DFI_Process->RankList[%d].VoxelSize[0] = %d\n",j,
388               DFI_Process->RankList[j].VoxelSize[0]);
389       fprintf(fp,"\t  DFI_Process->RankList[%d].VoxelSize[1] = %d\n",j,
390               DFI_Process->RankList[j].VoxelSize[1]);
391       fprintf(fp,"\t  DFI_Process->RankList[%d].VoxelSize[2] = %d\n",j,
392               DFI_Process->RankList[j].VoxelSize[2]);
393       fprintf(fp,"\t  DFI_Process->RankList[%d].HeadIndex[0] = %d\n",j,
394               DFI_Process->RankList[j].HeadIndex[0]);
395       fprintf(fp,"\t  DFI_Process->RankList[%d].HeadIndex[1] = %d\n",j,
396               DFI_Process->RankList[j].HeadIndex[1]);
397       fprintf(fp,"\t  DFI_Process->RankList[%d].HeadIndex[2] = %d\n",j,
398               DFI_Process->RankList[j].HeadIndex[2]);
399       fprintf(fp,"\t  DFI_Process->RankList[%d].TailIndex[0] = %d\n",j,
400               DFI_Process->RankList[j].TailIndex[0]);
401       fprintf(fp,"\t  DFI_Process->RankList[%d].TailIndex[1] = %d\n",j,
402               DFI_Process->RankList[j].TailIndex[1]);
403       fprintf(fp,"\t  DFI_Process->RankList[%d].TailIndex[2] = %d\n",j,
404               DFI_Process->RankList[j].TailIndex[2]);
405     }
406
407     const cio_TimeSlice* DFI_TSlice = dfi->GetcioTimeSlice();
408     fprintf(fp,"\tDFI_TSlice->SliceList.size()       = %d\n",DFI_TSlice->SliceList.size());
409     for(int j=0; j<DFI_TSlice->SliceList.size(); j++ ) {
410       fprintf(fp,"\t  DFI_TSlice->SliceList[%d].step         = %d\n",j,
411               DFI_TSlice->SliceList[j].step);
412       fprintf(fp,"\t  DFI_TSlice->SliceList[%d].time         = %f\n",j,
413               DFI_TSlice->SliceList[j].time);
414       if( !DFI_TSlice->SliceList[j].avr_mode ) {
415         fprintf(fp,"\t  DFI_TSlice->SliceList[%d].AveragedStep = %d\n",j,
416               DFI_TSlice->SliceList[j].AveragedStep);
417         fprintf(fp,"\t  DFI_TSlice->SliceList[%d].AveragedTime = %f\n",j,
418               DFI_TSlice->SliceList[j].AveragedTime);
419       }
420       if( DFI_Info->Component > 1 ) {
421         fprintf(fp,"\t  DFI_TSlice->SliceList[%d].VectorMin    = %f\n",j,
422               DFI_TSlice->SliceList[j].VectorMin);
423         fprintf(fp,"\t  DFI_TSlice->SliceList[%d].VectorMax    = %f\n",j,
424               DFI_TSlice->SliceList[j].VectorMax);
425         for(int k=0; k<DFI_TSlice->SliceList[j].Min.size(); k++) {
426           fprintf(fp,"\t    DFI_TSlice->SliceList[%d].Min[%d]      = %f\n",j,k,
427               DFI_TSlice->SliceList[j].Min[k]);
428         }
429         for(int k=0; k<DFI_TSlice->SliceList[j].Max.size(); k++) {
430           fprintf(fp,"\t    DFI_TSlice->SliceList[%d].Max[%d]      = %f\n",j,k,
431               DFI_TSlice->SliceList[j].Max[k]);
432         }
433       }
434     }
435
436     fprintf(fp,"\n");
437   }
438
439 }
```

4.1.3.23 CIO::E_CIO_ERRORCODE CONV::ReadDfiFiles ( )

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 dfi ファイルの読み込みとDfiInfo クラスデータの作成

conv.C の 84 行で定義されています。

参照先 CheckDFIdata(), LOG_OUTV_, InputParam::m_dfiList, m_fplog, m_in_dfi, m_myRank, m_param, PrintDFI(), InputParam::PrintParam(), と STD_OUTV_.

参照元 main().

```
85 {
86
87   // dfi ファイルの読込み
88   int tempg[3];
89   int tempd[3];
90   CIO::E_CIO_ERRORCODE ret = CIO::E_CIO_SUCCESS;
91   for(int i=0; i<m_param->m_dfiList.size(); i++) {
92     cio_DFI* dfi_in = cio_DFI::ReadInit(MPI_COMM_WORLD,
93                                         m_param->m_dfiList[i].in_dfi_name,
94                                         tempg,
95                                         tempd,
96                                         ret);
97     if( dfi_in == NULL ) return ret;
98     if( ret != CIO::E_CIO_SUCCESS && ret != CIO::E_CIO_ERROR_INVALID_DIVNUM ) return ret;
99     m_in_dfi.push_back(dfi_in);
100     m_param->m_dfiList[i].in_dfi = dfi_in;
101   }
102
103   LOG_OUTV_ {
104     PrintDFI(m_fplog);
105     m_param->PrintParam(m_fplog);
106   }
107
108   STD_OUTV_ {
109     if( m_myRank == 0 ) {
110       PrintDFI(stdout);
111       m_param->PrintParam(stdout);
112     }
113   }
114
115   // dfi 毎の成分数、出力ガイドセルのチェックと更新、等
116   if( !CheckDFIdata() ) return CIO::E_CIO_ERROR;
117
118   return CIO::E_CIO_SUCCESS;
119 }
```

4.1.3.24 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   }
```

4.1.3.25 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 228 行で定義されています。

参照元 main().

```
228 { return; }
```

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

index.dfi の出力

引数

minmaxList	minmax のリスト

conv.C の 876 行で定義されています。

参照先 InputParam::Get_OutputArrayShape(), InputParam::Get_OutputDataType(), InputParam::Get_Outputdfi_on(), InputParam::Get_OutputDir(), InputParam::Get_OutputFilenameFormat(), InputParam::Get_OutputFormat(), InputParam::Get_OutputGuideCell(), InputParam::m_dfiList, と m_param.

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

```
877 {
878
879   if( !m_param->Get_Outputdfi_on() ) return false;
880
881   //出力 dfi ファイル名の取得
882   /*
883   vector<std::string> out_dfi_name  = m_InputCntl->Get_OutdfiNameList();
884   vector<std::string> out_proc_name = m_InputCntl->Get_OutprocNameList();
885
886   if( minmaxList.size() != out_dfi_name.size() &&
887       minmaxList.size() != out_proc_name.size() ) return false;
888   */
889   for(int i=0; i<minmaxList.size(); i++) {
890
891     cio_DFI* dfi = minmaxList[i]->dfi;
892
893     std::string out_dfi_name  = m_param->m_dfiList[i].out_dfi_name;
894     std::string out_proc_name = m_param->m_dfiList[i].out_proc_name;
895
896     FILE* fp=NULL;
897     if( !(fp = fopen(out_dfi_name.c_str(), "w")) )
898     {
899       printf("Can't open file.(%s)\n", out_dfi_name.c_str());
900       return false;
901     }
902
903     //成分数の取得
904     int nComp = dfi->GetNumComponent();
905
906     cio_FileInfo *dfi_Finfo = (cio_FileInfo *)dfi->GetcioFileInfo();
907
908     CIO::E_CIO_ARRAYSHAPE shape = dfi_Finfo->ArrayShape;
909     //if( dfi_Finfo->FileFormat == (CIO::E_CIO_FMT_BOV) ) {
910     if( m_param->Get_OutputFormat() == (CIO::E_CIO_FMT_BOV) ) {
911       shape = m_param->Get_OutputArrayShape();
912     }
913
914     int outGc = m_param->Get_OutputGuideCell();
915     if( outGc > 0 ) {
916       if( outGc > dfi_Finfo->GuideCell ) outGc=dfi_Finfo->GuideCell;
917     }
918
919     CIO::E_CIO_DTYPE out_d_type = m_param->Get_OutputDataType();
920     if( out_d_type == CIO::E_CIO_DTYPE_UNKNOWN ) out_d_type=dfi->GetDataType();
921
922     CIO::E_CIO_OUTPUT_FNAME FieldFilenameFormat;
923     FieldFilenameFormat=m_param->Get_OutputFilenameFormat();
924
925     //FileInfo の出力
926     cio_FileInfo *Finfo = new cio_FileInfo(CIO::E_CIO_DFITYPE_CARTESIAN,
927                                FieldFilenameFormat,
928                                m_param->Get_OutputDir(),
929                                dfi_Finfo->TimeSliceDirFlag,
930                                dfi_Finfo->Prefix,
931                                m_param->Get_OutputFormat(),
932                                outGc,
933                                //(CIO::E_CIO_DTYPE)m_InputCntl->Get_OutputDataType(),
934                                out_d_type,
935                                dfi_Finfo->Endian,
936                                //(CIO::E_CIO_ARRAYSHAPE)m_InputCntl->Get_OutputArrayShape(),
937                                shape,
938                                nComp);
939
940     for(int n=0; n<nComp; n++) {
941        std::string variable = dfi->getComponentVariable(n);
942        if( variable != "" ) Finfo->setComponentVariable(n,variable);
943     }
944
945     if( Finfo->Write(fp, 0) != CIO::E_CIO_SUCCESS ) {
946       fclose(fp);
947       return false;
948     }
```

```
949     delete Finfo;
950
951     //FilePath の出力
952     cio_FilePath *dfi_Fpath = (cio_FilePath *)dfi->GetcioFilePath();
953     cio_FilePath *Fpath = new cio_FilePath(out_proc_name);
954     if( Fpath->Write(fp, 1) != CIO::E_CIO_SUCCESS )
955     {
956       fclose(fp);
957       return false;
958     }
959     delete Fpath;
960
961     //Unit の出力
962     cio_Unit *dfi_Unit = (cio_Unit *)dfi->GetcioUnit();
963     if( dfi_Unit->Write(fp, 0) != CIO::E_CIO_SUCCESS )
964     {
965       fclose(fp);
966       return false;
967     }
968
969     //TimeSlice の出力
970     const cio_TimeSlice *dfi_TSlice = dfi->GetcioTimeSlice();
971     cio_TimeSlice *TSlice = new cio_TimeSlice();
972     int nsize = nComp;
973     if( nComp > 1 ) nsize++;
974     double* minmax = new double[nsize*2];
975     for(int j=0; j<dfi_TSlice->SliceList.size(); j++) {
976       for(int n=0; n<nsize; n++) {
977         minmax[n*2+0] = minmaxList[i]->Min[j*nsize+n];
978         minmax[n*2+1] = minmaxList[i]->Max[j*nsize+n];
979       }
980       TSlice->AddSlice(dfi_TSlice->SliceList[j].step,
981                        dfi_TSlice->SliceList[j].time,
982                        minmax,
983                        nComp,
984                        true,
985                        0,
986                        0.0);
987
988     }
989
990     if( TSlice->Write(fp, 1) != CIO::E_CIO_SUCCESS )
991     {
992       fclose(fp);
993       return false;
994     }
995
996     //fclose(fp);
997
998   }
999
1000   return true;
1001 }
```

4.1.3.27 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 1114 行で定義されています。

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

```
1118 {
1119
1120   FILE* fp = NULL;
1121
1122   if( out_domain == NULL || out_mpi == NULL || out_process == NULL ) return false;
1123
1124   //proc.dfi ファイルオープン
1125   if( !(fp = fopen(proc_name.c_str(), "w")) )
1126   {
1127     printf("Can't open file.(%s)\n", proc_name.c_str());
1128     return false;
1129   }
1130
1131   //Domain {} の出力
1132   if( out_domain->Write(fp, 0) != CIO::E_CIO_SUCCESS )
1133   {
1134     fclose(fp);
1135     return false;
```

```
1136   }
1137
1138   //MPI {} の出力
1139   if( out_mpi->Write(fp, 0) != CIO::E_CIO_SUCCESS )
1140   {
1141     fclose(fp);
1142     return false;
1143   }
1144
1145   //Process {} の出力
1146   if( out_process->Write(fp, 0) != CIO::E_CIO_SUCCESS )
1147   {
1148     fclose(fp);
1149     return false;
1150   }
1151
1152   fclose(fp);
1153
1154   return true;
1155 }
```

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

所要時間の記述

引数

in	*tt*	所要時間

conv.C の 443 行で定義されています。

参照先 m_fplog.

参照元 main().

```
444 {
445   fprintf(m_fplog,"\n\n");
446   fprintf(m_fplog,"TIME : ReadInit      %10.3f sec.\n", tt[0]);
447   fprintf(m_fplog,"TIME : ReadDfiFiles  %10.3f sec.\n", tt[1]);
448   fprintf(m_fplog,"TIME : Converter     %10.3f sec.\n", tt[2]);
449   fprintf(m_fplog,"TIME : Total Time    %10.3f sec.\n", tt[3]);
450 }
```

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 convMx1::exec(), OpenLogFile(), ReadDfiFiles(), WriteTime(), と ~CONV().

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 CheckDFIdata(), CONV(), convMxM::exec(), convMx1::exec(), convMxN::exec(), makeRankList(), makeStepList(), ReadDfiFiles(), convMxN::VoxelInit(), と ~CONV().

4.1.4.5 int CONV::m_lflag

conv.h の 101 行で定義されています。

参照元 CONV(), と main().

4.1.4.6 int CONV::m_lflagv

conv.h の 102 行で定義されています。

参照元 CONV(), と main().

4.1.4.7 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(), convMxN::exec(), makeRankList(), makeStepList(), OpenLogFile(), と ReadDfiFiles().

4.1.4.8 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().

4.1.4.9 InputParam* CONV::m_param

InputParam Class.

conv.h の 82 行で定義されています。

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

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().

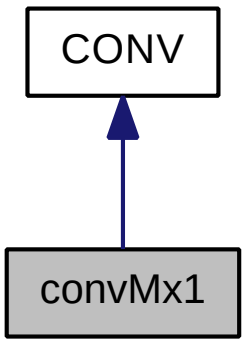
4.1.4.14 unsigned CONV::m_staging [private]

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:

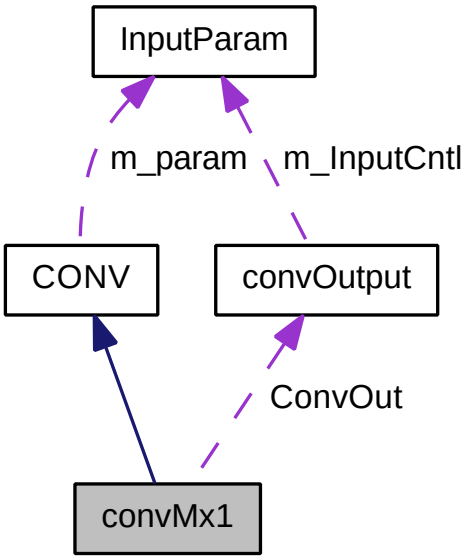
- conv.h
- conv.C
- conv_inline.h

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, CIO::E_CIO_DTYPE 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, CIO::E_CIO_DTYPE 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*, CIO::E_CIO_DTYPE *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 730 行で定義されています。

参照先 CONV::convertXY(), ConvOut, CONV::DtypeMinMax(), InputParam::Get_CropIndexEnd(), InputParam::Get_CropIndexEnd_on(), InputParam::Get_CropIndexStart(), InputParam::Get_CropIndexStart_on(), InputParam::Get_OutputGuideCell(), InputParam::Get_ThinOut(), InterPolate(), CONV::m_bgrid_interp_flag, CONV::m_param, nijk_to_ijk(), と convOutput::WriteFieldData().

参照元 exec().

```
742 {
743
744   //cio_Domain* DFI_Domian = (cio_Domain *)m_in_dfi[0]->GetcioDomain();
745   cio_Domain* DFI_Domian = (cio_Domain *)dfi->GetcioDomain();
746
747   int thin_count = m_param->Get_ThinOut();
748
749   int outGc=0;
750   int interp_Gc=0;
751
752   //出力ガイドセルの設定
753   if( m_param->Get_OutputGuideCell() > 1 ) outGc = m_param->Get_OutputGuideCell();
754   if( outGc > 1 ) {
755     const cio_FileInfo* DFI_FInfo = dfi->GetcioFileInfo();
756     if( outGc > DFI_FInfo->GuideCell ) outGc=DFI_FInfo->GuideCell;
757   }
758
759   //間引きありのとき、出力ガイドセルを 0 に設定
760   if( thin_count > 1 ) outGc=0;
761   interp_Gc = outGc;
762
```

```
763   //格子点出力のときガイドセルが 0 のとき 1 にセット
764   if( m_bgrid_interp_flag && outGc==0 ) interp_Gc=1;
765
766   //cell 出力のとき、出力ガイドセルを 0 に設定
767   if( !m_bgrid_interp_flag ) interp_Gc=0;
768
769   //出力のヘッダー、フッターをセット
770   int headS[3],tailS[3];
771   const int* CorpIndexStart = m_param->Get_CropIndexStart();
772   const int* CorpIndexEnd = m_param->Get_CropIndexEnd();
773   int IndexStart[3],IndexEnd[3];
774   for(int i=0;i<3; i++) {
775     IndexStart[i]=CorpIndexStart[i]-outGc;
776     IndexEnd[i]=CorpIndexEnd[i]+outGc;
777   }
778   if( !m_param->Get_CropIndexStart_on() ) {
779     IndexStart[0]=1-outGc;
780     IndexStart[1]=1-outGc;
781     IndexStart[2]=1-outGc;
782   }
783   if( !m_param->Get_CropIndexEnd_on() ) {
784     IndexEnd[0]=DFI_Domian->GlobalVoxel[0]+outGc;
785     IndexEnd[1]=DFI_Domian->GlobalVoxel[1]+outGc;
786     IndexEnd[2]=DFI_Domian->GlobalVoxel[2]+outGc;
787   }
788
789   headS[0]=IndexStart[0]-1;
790   headS[1]=IndexStart[1]-1;
791   tailS[0]=IndexEnd[0]-1;
792   tailS[1]=IndexEnd[1]-1;
793
794   //成分数の取り出し
795   int nComp = dfi->GetNumComponent();
796
797   //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
798   if( !m_bgrid_interp_flag ) {
799     sz[0]=sz[0]+2*outGc;
800     sz[1]=sz[1]+2*outGc;
801   }
802
803   //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）
804   cio_Array* src = cio_Array::instanceArray
805                    ( d_type
806                    , dfi->GetArrayShape()
807                    , sz
808                    , interp_Gc
809                    , nComp );
810
811   //補間用バッファ（読込み配列形状での DFI でインスタンス）
812   cio_Array* src_old = NULL;
813   cio_Array* outArray = NULL;
814   if( m_bgrid_interp_flag ) {
815     src_old = cio_Array::instanceArray
816               ( d_type
817               , dfi->GetArrayShape()
818               , sz
819               , interp_Gc
820               , nComp );
821
822     int szOut[3];
823     for(int i=0; i<2; i++) szOut[i]=sz[i]+1;
824     szOut[2]=sz[2];
825     outArray = cio_Array::instanceArray
826              ( d_type
827               , dfi->GetArrayShape()
828               //, CIO::E_CIO_IJKN,
829               , szOut
830               , interp_Gc
831               , 1 );
832   }
833
834   int kdiv,jdiv,idiv;
835   int l_rank;
836   std::string infile;
837
838   //成分数のループ
839   for(int n=0; n<nComp; n++) {
840
841     //z 方向の分割数回のループ
842     for( headT::iterator itz=mapHeadZ.begin(); itz!= mapHeadZ.end(); itz++ ) {
843
844       //z 層のスタートエンドを設定
845       kdiv = itz->second;
846       int kp_sta,kp_end;
847       kp_sta = itz->first;
848       int nrank = _CIO_IDX_IJK(0,0,kdiv,div[0],div[1],div[2],0);
849       kp_end = kp_sta + DFI_Process->RankList[nrank].VoxelSize[2];
```

```
850
851       //z 層のスタートエンドをガイドセルの考慮
852       if( kdiv == 0 ) kp_sta = kp_sta-outGc;
853       if( kdiv == div[2]-1 ) kp_end = kp_end+outGc;
854
855       //同一 Z 面のループ
856       for(int kp=kp_sta; kp< kp_end; kp++) {
857
858         //入力領域外のときスキップ
859         if( kp < IndexStart[2] || kp > IndexEnd[2] ) continue;
860
861         int kk = kp-1;
862         //間引きの層のときスキップ
863         //if( kk%thin_count != 0 ) continue;
864         if( (kp-IndexStart[2])%thin_count != 0 ) continue;
865
866         //y 方向の分割数のループ
867         for( headT::iterator ity=mapHeadY.begin(); ity!= mapHeadY.end(); ity++ ) {
868
869           //y のスタートエンドの設定
870           jdiv = ity->second;
871           int jp_sta,jp_end;
872           jp_sta = ity->first;
873           int nrank = _CIO_IDX_IJK(0,jdiv,kdiv,div[0],div[1],div[2],0);
874           jp_end = jp_sta + DFI_Process->RankList[nrank].VoxelSize[1];
875
876           //x 方向の分割数のループ
877           for( headT::iterator itx=mapHeadX.begin(); itx!= mapHeadX.end(); itx++ ) {
878
879             //x のスタートエンドの設定
880             idiv = itx->second;
881             int ip_sta,ip_end;
882             ip_sta = itx->first;
883             int nrank = _CIO_IDX_IJK(idiv,jdiv,kdiv,div[0],div[1],div[2],0);
884             ip_end = ip_sta + DFI_Process->RankList[nrank].VoxelSize[0];
885
886             int RankID = _CIO_IDX_IJK(idiv,jdiv,kdiv,div[0],div[1],div[2],0);
887
888             //読込み範囲の設定
889             if( IndexStart[0] > ip_end || IndexEnd[0] < ip_sta ) continue;
890             if( IndexStart[1] > jp_end || IndexEnd[1] < jp_sta ) continue;
891
892             int read_sta[3],read_end[3];
893             read_sta[0]=ip_sta;
894             if( IndexStart[0] > ip_sta ) read_sta[0] = IndexStart[0];
895             read_sta[1]=jp_sta;
896             if( IndexStart[1] > jp_sta ) read_sta[1] = IndexStart[1];
897             read_sta[2]=kp;
898             read_end[0]=ip_end-1;
899             if( IndexEnd[0] < ip_end ) read_end[0] = IndexEnd[0];
900             read_end[1]=jp_end-1;
901             if( IndexEnd[1] < jp_end ) read_end[1] = IndexEnd[1];
902             read_end[2]=kp;
903
904             //ガイドセルを考慮して読込み範囲を更新
905             if( idiv == 0 ) read_sta[0] = read_sta[0]-outGc;
906             if( idiv == div[0]-1 ) read_end[0] = read_end[0]+outGc;
907             if( jdiv == 0 ) read_sta[1] = read_sta[1]-outGc;
908             if( jdiv == div[1]-1 ) read_end[1] = read_end[1]+outGc;
909
910             l_rank=DFI_Process->RankList[RankID].RankID;
911             //連結対象ファイル名の生成
912             infile = CIO::cioPath_ConnectPath(inPath,dfi->Generate_FieldFileName(l_rank,l_step,mio));
913             unsigned int avr_step;
914             double avr_time;
915             CIO::E_CIO_ERRORCODE ret;
916             //連結対象ファイルの読込み
917             cio_Array* buf = dfi->ReadFieldData(infile, l_step, l_dtime,
918                                                 read_sta, read_end,
919                                                 DFI_Process->RankList[RankID].HeadIndex,
920                                                 DFI_Process->RankList[RankID].TailIndex,
921                                                 true, avr_step, avr_time, ret);
922
923             if( ret != CIO::E_CIO_SUCCESS ) {
924               printf("\tCan't Read Field Data Record %s\n",infile.c_str());
925               return false;
926             }
927             //headIndex を 0 スタートにしてセット
928             int headB[3];
929             headB[0]=read_sta[0]-1;
930             headB[1]=read_sta[1]-1;
931             headB[2]=read_sta[2]-1;
932             buf->setHeadIndex( headB );
933
934             int headS0[3];
935             headS0[0]=headS[0];
936             headS0[1]=headS[1];
```

```
937             headS0[2]=kk/thin_count;
938             src->setHeadIndex( headS0 );
939
940             headS0[2]=kk;
941             tailS[2]=headS0[2];
942
943             //出力配列へのコンバイン
944             convertXY(buf,src,headS0,tailS,n);
945
946             //minmax を求める
947             if( n==0 ) if( !DtypeMinMax(buf,min,max) ) return false;
948
949             delete buf;
950
951           }
952         }
953         //補間処理
954         if( m_bgrid_interp_flag ) {
955           if( kp == kp_sta ) {
956             if( !InterPolate(src,src,outArray,n,0) ) return false;
957           } else {
958             if( !InterPolate(src_old,src,outArray,n,0) ) return false;
959           }
960         } else {
961           //NIJK レコードを IJK にコピー
962           outArray = nijk_to_ijk(src,n);
963
964         }
965
966         //一層分出力
967         if( outArray ) {
968           const int* szOutArray = outArray->getArraySizeInt();
969           size_t dLen = szOutArray[0]*szOutArray[1]*szOutArray[2]*outArray->getNcomp();
970           if( ConvOut->WriteFieldData(fp,
971                                       outArray,
972                                       dLen ) != true ) return false;
973         }
974         //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
975         if( m_bgrid_interp_flag ) {
976           cio_Array* tmp = src;
977           src = src_old;
978           src_old = tmp;
979         }
980       }
981     }
982
983     if( m_bgrid_interp_flag ) {
984       for(int n=0; n<nComp; n++) {
985         if( !InterPolate(src_old,src_old,outArray,n,0) ) return false;
986       }
987       if( outArray ) {
988         const int* szOutArray = outArray->getArraySizeInt();
989         size_t dLen = szOutArray[0]*szOutArray[1]*szOutArray[2]*outArray->getNcomp();
990         if( ConvOut->WriteFieldData(fp,
991                                     outArray,
992                                     dLen ) != true ) return false;
993       } else return false;
994     }
995   }
996
997   delete src;
998   if( m_bgrid_interp_flag ) {
999     delete src_old;
1000     delete outArray;
1001   }
1002
1003   return true;
1004
1005 }
```

4.2.4.2 bool convMx1::convMx1_out_nijk (FILE * *fp*, std::string *inPath*, int *l_step*, double *l_dtime*, CIO::E_CIO_DTYPE *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 438 行で定義されています。

参照先 CONV::convertXY(), ConvOut, CONV::DtypeMinMax(), InputParam::Get_CropIndexEnd(), InputParam::Get_CropIndexEnd_on(), InputParam::Get_CropIndexStart(), InputParam::Get_CropIndexStart_on(), InputParam::Get_OutputGuideCell(), InputParam::Get_ThinOut(), InterPolate(), CONV::m_bgrid_interp_flag, CONV::m_param, と convOutput::WriteFieldData().

参照元 exec().

```
450 {
451
452   //cio_Domain* DFI_Domian = (cio_Domain *)m_in_dfi[0]->GetcioDomain();
453   cio_Domain* DFI_Domian = (cio_Domain *)dfi->GetcioDomain();
454
455   int thin_count = m_param->Get_ThinOut();
456
457   int outGc=0;
458   int interp_Gc=0;
459
460   //出力ガイドセルの設定
461   if( m_param->Get_OutputGuideCell() > 1 ) outGc = m_param->Get_OutputGuideCell();
462   if( outGc > 0 ) {
463     const cio_FileInfo* DFI_FInfo = dfi->GetcioFileInfo();
464     if( outGc > DFI_FInfo->GuideCell ) outGc=DFI_FInfo->GuideCell;
465   }
466
467   //間引きありのとき、出力ガイドセルを 0 に設定
468   if( thin_count > 1 ) outGc=0;
469   interp_Gc = outGc;
470
471   //格子点出力のときガイドセルが 0 のとき 1 にセット
472   if( m_bgrid_interp_flag && outGc==0 ) interp_Gc=1;
473
474   //cell 出力のとき、出力ガイドセルを 0 に設定
475   if( !m_bgrid_interp_flag ) interp_Gc=0;
476
477   //出力のヘッダー、フッターをセット
478   int headS[3],tailS[3];
479   const int* CorpIndexStart = m_param->Get_CropIndexStart();
480   const int* CorpIndexEnd = m_param->Get_CropIndexEnd();
481   int IndexStart[3],IndexEnd[3];
482   for(int i=0;i<3; i++) {
483     IndexStart[i]=CorpIndexStart[i]-outGc;
484     IndexEnd[i]=CorpIndexEnd[i]+outGc;
485   }
486   if( !m_param->Get_CropIndexStart_on() ) {
487     IndexStart[0]=1-outGc;
488     IndexStart[1]=1-outGc;
489     IndexStart[2]=1-outGc;
490   }
```

```
491   if( !m_param->Get_CropIndexEnd_on() ) {
492     IndexEnd[0]=DFI_Domian->GlobalVoxel[0]+outGc;
493     IndexEnd[1]=DFI_Domian->GlobalVoxel[1]+outGc;
494     IndexEnd[2]=DFI_Domian->GlobalVoxel[2]+outGc;
495   }
496
497   headS[0]=IndexStart[0]-1;
498   headS[1]=IndexStart[1]-1;
499   tailS[0]=IndexEnd[0]-1;
500   tailS[1]=IndexEnd[1]-1;
501
502   //成分数の取り出し
503   int nComp = dfi->GetNumComponent();
504
505   //配列形状の設定
506   CIO::E_CIO_ARRAYSHAPE out_shape;
507   if( nComp == 1 ) out_shape = CIO::E_CIO_IJKN;
508   else if( nComp > 1 ) out_shape = CIO::E_CIO_NIJK;
509
510   //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
511   if( !m_bgrid_interp_flag ) {
512     sz[0]=sz[0]+2*outGc;
513     sz[1]=sz[1]+2*outGc;
514   }
515
516   //出力バッファのインスタンス
517   cio_Array* src = cio_Array::instanceArray
518                    ( d_type
519                    //, dfi->GetArrayShape()
520                    , out_shape
521                    , sz
522                    , interp_Gc
523                    , nComp );
524
525   //補間用バッファ，格子点出力バッファのインスタンス
526   cio_Array* src_old = NULL;
527   cio_Array* outArray = NULL;
528   if( m_bgrid_interp_flag ) {
529     src_old = cio_Array::instanceArray
530               ( d_type
531               //, dfi->GetArrayShape()
532               , out_shape
533               , sz
534               , interp_Gc
535               , nComp );
536
537     int szOut[3];
538     for(int i=0; i<2; i++) szOut[i]=sz[i]+1;
539     szOut[2]=sz[2];
540     outArray = cio_Array::instanceArray
541              ( d_type
542               //, dfi->GetArrayShape()
543               , out_shape
544               , szOut
545               , interp_Gc
546               , nComp );
547   }
548
549   int kdiv,jdiv,idiv;
550   int l_rank;
551   std::string infile;
552
553   //z 方向の分割数回のループ
554   for( headT::iterator itz=mapHeadZ.begin(); itz!= mapHeadZ.end(); itz++ ) {
555
556     //z 層のスタートエンドを設定
557     kdiv = itz->second;
558     int kp_sta,kp_end;
559     kp_sta = itz->first;
560     int nrank = _CIO_IDX_IJK(0,0,kdiv,div[0],div[1],div[2],0);
561     kp_end = kp_sta + DFI_Process->RankList[nrank].VoxelSize[2];
562
563     //z 層のスタートエンドをガイドセルの考慮
564     if( kdiv == 0 ) kp_sta = kp_sta-outGc;
565     if( kdiv == div[2]-1 ) kp_end = kp_end+outGc;
566
567     //同一 z 面のループ
568     for(int kp=kp_sta; kp< kp_end; kp++) {
569
570       //入力領域外のときスキップ
571       if( kp < IndexStart[2] || kp > IndexEnd[2] ) continue;
572
573       int kk = kp-1;
574       //間引きの層のときスキップ
575       //if( kk%thin_count != 0 ) continue;
576       if( (kp-IndexStart[2])%thin_count != 0 ) continue;
577
```

```
578       //y 方向の分割数のループ
579       for( headT::iterator ity=mapHeadY.begin(); ity!= mapHeadY.end(); ity++ ) {
580
581         //y のスタートエンドの設定
582         jdiv = ity->second;
583         int jp_sta,jp_end;
584         jp_sta = ity->first;
585         int nrank = _CIO_IDX_IJK(0,jdiv,kdiv,div[0],div[1],div[2],0);
586         jp_end = jp_sta + DFI_Process->RankList[nrank].VoxelSize[1];
587
588         //x 方向の分割数のループ
589         for( headT::iterator itx=mapHeadX.begin(); itx!= mapHeadX.end(); itx++ ) {
590
591           //x のスタートエンドの設定
592           idiv = itx->second;
593           int ip_sta,ip_end;
594           ip_sta = itx->first;
595           int nrank = _CIO_IDX_IJK(idiv,jdiv,kdiv,div[0],div[1],div[2],0);
596           ip_end = ip_sta + DFI_Process->RankList[nrank].VoxelSize[0];
597
598           int RankID = _CIO_IDX_IJK(idiv,jdiv,kdiv,div[0],div[1],div[2],0);
599
600           //読込み範囲の設定
601           if( IndexStart[0] > ip_end || IndexEnd[0] < ip_sta ) continue;
602           if( IndexStart[1] > jp_end || IndexEnd[1] < jp_sta ) continue;
603
604           int read_sta[3],read_end[3];
605           read_sta[0]=ip_sta;
606           if( IndexStart[0] > ip_sta ) read_sta[0] = IndexStart[0];
607           read_sta[1]=jp_sta;
608           if( IndexStart[1] > jp_sta ) read_sta[1] = IndexStart[1];
609           read_sta[2]=kp;
610           read_end[0]=ip_end-1;
611           if( IndexEnd[0] < ip_end ) read_end[0] = IndexEnd[0];
612           read_end[1]=jp_end-1;
613           if( IndexEnd[1] < jp_end ) read_end[1] = IndexEnd[1];
614           read_end[2]=kp;
615
616           //ガイドセルを考慮して読込み範囲を更新
617           if( idiv == 0 ) read_sta[0] = read_sta[0]-outGc;
618           if( idiv == div[0]-1 ) read_end[0] = read_end[0]+outGc;
619           if( jdiv == 0 ) read_sta[1] = read_sta[1]-outGc;
620           if( jdiv == div[1]-1 ) read_end[1] = read_end[1]+outGc;
621
622           l_rank=DFI_Process->RankList[RankID].RankID;
623           //連結対象ファイル名の生成
624           infile = CIO::cioPath_ConnectPath(inPath,dfi->Generate_FieldFileName(l_rank,l_step,mio));
625
626           unsigned int avr_step;
627           double avr_time;
628           CIO::E_CIO_ERRORCODE ret;
629           //連結対象ファイルの読込み
630           cio_Array* buf = dfi->ReadFieldData(infile, l_step, l_dtime,
631                                               read_sta, read_end,
632                                               DFI_Process->RankList[RankID].HeadIndex,
633                                               DFI_Process->RankList[RankID].TailIndex,
634                                               true, avr_step, avr_time, ret);
635           if( ret != CIO::E_CIO_SUCCESS ) {
636             printf("\tCan't Read Field Data Record %s\n",infile.c_str());
637             return false;
638           }
639           //headIndex を0スタートにしてセット
640           int headB[3];
641           headB[0]=read_sta[0]-1;
642           headB[1]=read_sta[1]-1;
643           headB[2]=read_sta[2]-1;
644           buf->setHeadIndex( headB );
645
646           int headS0[3];
647           //headS0[0]=headS[0];
648           //headS0[1]=headS[1];
649           headS0[0]=headS[0]/thin_count;
650           headS0[1]=headS[1]/thin_count;
651           headS0[2]=kk/thin_count;
652           src->setHeadIndex( headS0 );
653
654           headS0[0]=headS[0];
655           headS0[1]=headS[1];
656           headS0[2]=kk;
657           tailS[2]=headS0[2];
658
659           //出力配列へのコンバイン
660           for(int n=0; n<nComp; n++) convertXY(buf,src,headS0,tailS,n);
661           delete buf;
662
663         }
664       }
```

```
665       //補間処理
666       if( m_bgrid_interp_flag ) {
667         if( kp == kp_sta ) {
668           for(int n=0; n<nComp; n++) {
669             if( !InterPolate(src,src,outArray,n,n) ) return false;
670           }
671         } else {
672           for(int n=0; n<nComp; n++) {
673             if( !InterPolate(src_old,src,outArray,n,n) ) return false;
674           }
675         }
676       } else outArray = src;
677
678       //一層分出力
679       if( outArray ) {
680         const int* szOutArray = outArray->getArraySizeInt();
681         size_t dLen = szOutArray[0]*szOutArray[1]*szOutArray[2]*outArray->getNcomp();
682         if( ConvOut->WriteFieldData(fp,
683                                     outArray,
684                                     dLen ) != true ) return false;
685
686       }
687
688       //minmax を求める
689       if( !DtypeMinMax(outArray,min,max) ) return false;
690
691       //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
692       if( m_bgrid_interp_flag ) {
693         cio_Array* tmp = src;
694         src = src_old;
695         src_old = tmp;
696       }
697     }
698   }
699
700   if( m_bgrid_interp_flag ) {
701     for(int n=0; n<nComp; n++) {
702       if( !InterPolate(src_old,src_old,outArray,n,n) ) return false;
703     }
704     if( outArray ) {
705       const int* szOutArray = outArray->getArraySizeInt();
706       size_t dLen = szOutArray[0]*szOutArray[1]*szOutArray[2]*outArray->getNcomp();
707       if( ConvOut->WriteFieldData(fp,
708                                   outArray,
709                                   dLen ) != true ) return false;
710
711        //minmax を求める
712        if( !DtypeMinMax(src,min,max) ) return false;
713
714     } else return false;
715   }
716   delete src;
717
718   if( m_bgrid_interp_flag ) {
719     delete src_old;
720     delete outArray;
721   }
722
723   return true;
724
725 }
```

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_CropIndexEnd(), InputParam::Get_CropIndexEnd_on(), InputParam::Get_CropIndexStart(), InputParam::Get_CropIndexStart_on(), InputParam::Get_OutputArrayShape(), InputParam::Get_OutputDataType(), InputParam::Get_Outputdfi_on(), InputParam::Get_OutputFormat(), InputParam::Get_OutputGuideCell(), InputParam::Get_ThinOut(), convOutput::importInputParam(), LOG_OUT_, LOG_OUTV_, CONV::m_bgrid_interp_flag, InputParam::m_dfiList, CONV::m_fplog, CONV::m_in_dfi, CONV::m_myRank, CONV::m_param, 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   if( m_myRank == 0 ) {
41     printf("Convert M x 1\n");
42   }
43
44   // 出力ファイル形式クラスのインスタンス
45   ConvOut = convOutput::OutputInit(m_param->Get_OutputFormat());
46
47   // InputParam のインスタンス
48   if( !ConvOut->importInputParam(m_param) ) {
49     return false;
50   }
51
52   string prefix,outfile,infile,inPath;
53   FILE *fp;
54   int dummy;
55
56   int l_rank;
57   int l_d_type;
58   CIO::E_CIO_DTYPE d_type;
59   int l_step, l_imax, l_jmax, l_kmax;
60   float l_time;
61   double l_dorg[3], l_dpit[3];
62   double l_dtime;
63   int xsize,ysize,zsize,asize,vsize;
64   int dim;
65
66   int div[3];
67   int l_imax_th, l_jmax_th, l_kmax_th;
```

```
68
69   //間引き数のセット
70   int thin_count = m_param->Get_ThinOut();
71
72   // 入力モード
73   bool mio = false;
74   //const cio_MPI* DFI_MPI = m_in_dfi[0]->GetcioMPI();
75   //if( DFI_MPI->NumberOfRank > 1) mio=true;
76
77   headT mapHeadX;
78   headT mapHeadY;
79   headT mapHeadZ;
80
81   //step 基準のリスト生成
82   makeStepList(m_StepRankList);
83
84   //minmax の格納構造体のインスタンス
85   vector<dfi_MinMax*> minmaxList;
86
87   for(int i=0; i<m_in_dfi.size(); i++){
88     const cio_TimeSlice* TSlice = m_in_dfi[i]->GetcioTimeSlice();
89     int nComp = m_in_dfi[i]->GetNumComponent();
90
91     dfi_MinMax *MinMax;
92     if( nComp == 1 ) MinMax = new dfi_MinMax(TSlice->SliceList.size(),nComp);
93     else             MinMax = new dfi_MinMax(TSlice->SliceList.size(),nComp+1);
94
95     MinMax->dfi = m_in_dfi[i];
96     minmaxList.push_back(MinMax);
97   }
98
99   //入力領域指示を考慮
100   int IndexStart[3];
101   int IndexEnd[3];
102   const int *cropIndexStart = m_param->Get_CropIndexStart();
103   const int *cropIndexEnd =   m_param->Get_CropIndexEnd();
104   for(int i=0; i<3; i++ ) {
105     IndexStart[i]=cropIndexStart[i];
106     IndexEnd[i]=cropIndexEnd[i];
107   }
108   //dfi*step のループ
109   for (int i=0;i<m_StepRankList.size();i++) {
110
111     cio_Domain* DFI_Domian = (cio_Domain *)m_StepRankList[i].dfi->GetcioDomain();
112     cio_Process* DFI_Process = (cio_Process *)m_StepRankList[i].dfi->GetcioProcess();
113     //全体サイズのキープ
114     l_imax= DFI_Domian->GlobalVoxel[0];
115     l_jmax= DFI_Domian->GlobalVoxel[1];
116     l_kmax= DFI_Domian->GlobalVoxel[2];
117
118     //オリジナルのピッチを計算
119     double o_pit[3];
120     for(int j=0; j<3; j++) o_pit[j]=DFI_Domian->GlobalRegion[j]/DFI_Domian->GlobalVoxel[j];
121
122     //入力領域指示を考慮
123     if( !m_param->Get_CropIndexStart_on() ) {
124       IndexStart[0] = 1;
125       IndexStart[1] = 1;
126       IndexStart[2] = 1;
127     }
128     if( !m_param->Get_CropIndexEnd_on() ) {
129       IndexEnd[0] = l_imax;
130       IndexEnd[1] = l_jmax;
131       IndexEnd[2] = l_kmax;
132     }
133     l_imax = IndexEnd[0]-IndexStart[0]+1;
134     l_jmax = IndexEnd[1]-IndexStart[1]+1;
135     l_kmax = IndexEnd[2]-IndexStart[2]+1;
136
137     //入力領域を考慮したリージョンの作成
138     double region[3];
139     region[0]=l_imax*o_pit[0];
140     region[1]=l_jmax*o_pit[1];
141     region[2]=l_kmax*o_pit[2];
142
143     //間引きを考慮
144     l_imax_th=l_imax/thin_count;//間引き後の x サイズ
145     l_jmax_th=l_jmax/thin_count;//間引き後の y サイズ
146     l_kmax_th=l_kmax/thin_count;//間引き後の z サイズ
147     if(l_imax%thin_count != 0) l_imax_th++;
148     if(l_jmax%thin_count != 0) l_jmax_th++;
149     if(l_kmax%thin_count != 0) l_kmax_th++;
150
151     //sph のオリジンとピッチを作成
152     l_dpit[0]=region[0]/(double)l_imax_th;
153     l_dpit[1]=region[1]/(double)l_jmax_th;
154     l_dpit[2]=region[2]/(double)l_kmax_th;
```

```
155     l_dorg[0]=DFI_Domian->GlobalOrigin[0]+0.5*l_dpit[0];
156     l_dorg[1]=DFI_Domian->GlobalOrigin[1]+0.5*l_dpit[1];
157     l_dorg[2]=DFI_Domian->GlobalOrigin[2]+0.5*l_dpit[2];
158
159     //GRID データ 出力
160     const cio_FileInfo* DFI_FInfo = m_StepRankList[i].dfi->GetcioFileInfo();
161     int sz[3];
162     sz[0]=l_imax;
163     sz[1]=l_jmax;
164     sz[2]=l_kmax;
165     ConvOut->WriteGridData(DFI_FInfo->Prefix,0, m_myRank, m_in_dfi[0]->GetDataType(),
166                            DFI_FInfo->GuideCell, l_dorg, l_dpit, sz);
167
168     //dfi ファイルのディレクトリの取得
169     inPath = CIO::cioPath_DirName(m_StepRankList[i].dfi->get_dfi_fname());
170
171     //const cio_FileInfo* DFI_FInfo = m_StepRankList[i].dfi->GetcioFileInfo();
172     prefix=DFI_FInfo->Prefix;
173     LOG_OUTV_ fprintf(m_fplog,"  COMBINE SPH START : %s\n", prefix.c_str());
174     STD_OUTV_ printf("  COMBINE SPH START : %s\n", prefix.c_str());
175
176     //Scalar or Vector
177     dim=m_StepRankList[i].dfi->GetNumComponent();
178
179     const cio_TimeSlice* TSlice = m_StepRankList[i].dfi->GetcioTimeSlice();
180
181     div[0]=DFI_Domian->GlobalDivision[0];
182     div[1]=DFI_Domian->GlobalDivision[1];
183     div[2]=DFI_Domian->GlobalDivision[2];
184     //mapHeadX,Y,Z の生成
185     DFI_Process->CreateRankList(*DFI_Domian,
186                                 mapHeadX,
187                                 mapHeadY,
188                                 mapHeadZ);
189
190     //入力ファイルの並列フラグセット
191     const cio_MPI* DFI_MPI = m_StepRankList[i].dfi->GetcioMPI();
192     if( DFI_MPI->NumberOfRank > 1) mio=true;
193     else mio=false;
194
195     //step Loop
196     for(int j=m_StepRankList[i].stepStart; j<=m_StepRankList[i].stepEnd; j++) {
197
198       //minmax の初期化
199       int nsize = dim;
200       if( dim > 1 ) nsize++;
201       double *min = new double[nsize];
202       double *max = new double[nsize];
203       for(int n=0; n<nsize; n++) {
204          min[n]=DBL_MAX;
205          max[n]=-DBL_MAX;
206       }
207
208       l_step=TSlice->SliceList[j].step;
209       l_time=(float)TSlice->SliceList[j].time;
210
211       LOG_OUTV_ fprintf(m_fplog,"\tstep = %d\n", l_step);
212       STD_OUTV_ printf("\tstep = %d\n", l_step);
213
214       //連結出力ファイルオープン
215       fp = ConvOut->OutputFile_Open(prefix, l_step, 0, false);
216
217       //m_d_type のセット (float or double)
218       if( m_StepRankList[i].dfi->GetDataType() == CIO::E_CIO_FLOAT32 ) {
219         l_d_type = SPH_FLOAT;
220       } else if( m_StepRankList[i].dfi->GetDataType() == CIO::E_CIO_FLOAT64 ) {
221         l_d_type = SPH_DOUBLE;
222       }
223
224       if( m_param->Get_OutputDataType() == CIO::E_CIO_DTYPE_UNKNOWN )
225       {
226         d_type = m_StepRankList[i].dfi->GetDataType();
227       } else {
228         d_type = m_param->Get_OutputDataType();
229       }
230
231       //ヘッダーレコードを出力
232       int outGc=0;
233       if( m_param->Get_OutputGuideCell() > 1 ) outGc = m_param->
    Get_OutputGuideCell();
234       double t_org[3];
235       for(int n=0; n<3; n++) t_org[n]=l_dorg[n];
236
237       //出力ガイドセルによるオリジンの更新
238       if( outGc > 0 ) {
239         if( outGc > DFI_FInfo->GuideCell ) outGc=DFI_FInfo->GuideCell;
240         for(int n=0; n<3; n++) t_org[n]=t_org[n]-(double)outGc*l_dpit[n];
```

```
241       }
242       if( thin_count > 1 || m_bgrid_interp_flag ) outGc-0;
243
244       if( !(ConvOut->WriteHeaderRecord(l_step, dim, d_type,
245                                        l_imax_th+2*outGc, l_jmax_th+2*outGc, l_kmax_th+2*outGc,
246                                        l_time, t_org, l_dpit, prefix, fp)) ) {
247         printf("\twrite header error\n");
248         return false;
249       }
250       //全体の大きさの計算とデータのヘッダ書き込み
251       size_t dLen;
252
253       //dLen = size_t(l_imax_th) * size_t(l_jmax_th) * size_t(l_kmax_th);
254       dLen = size_t(l_imax_th+2*outGc) * size_t(l_jmax_th+2*outGc) * size_t(l_kmax_th+2*outGc);
255       if( dim == 3 ) dLen *= 3;
256       //if( m_param->Get_OutputDataType() == CIO::E_CIO_FLOAT32 ) {
257       if( d_type == CIO::E_CIO_FLOAT32 ) {
258          dummy = dLen * sizeof(float);
259       } else {
260          dummy = dLen * sizeof(double);
261       }
262       if( !(ConvOut->WriteDataMarker(dummy, fp)) ) {
263         printf("\twrite data header error\n");
264         return false;
265       }
266
267       //書き込み workarea のサイズ決め
268       xsize=l_imax_th;
269       ysize=l_jmax_th;
270       asize=xsize*ysize;
271       vsize=0;
272       for(int n=0; n< DFI_Process->RankList.size(); n++ ) {
273         int szx,szy,szz;
274         szx=DFI_Process->RankList[n].VoxelSize[0];
275         szy=DFI_Process->RankList[n].VoxelSize[1];
276         szz=DFI_Process->RankList[n].VoxelSize[2];
277         int vdum=szx*szy*szz;
278         if(vsize < vdum) vsize=vdum;
279       }
280       // メモリチェック
281       LOG_OUTV_ fprintf(m_fplog,"\tNode %4d - Node %4d\n", 0,
282                         DFI_Domian->GlobalVoxel[2] -1);
283       STD_OUTV_ printf("\tNode %4d - Node %4d\n", 0,
284                        DFI_Domian->GlobalVoxel[2] -1);
285       double mc1 = (double)asize*(double)dim;
286       double mc2 = (double)vsize*(double)dim;
287       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
288         printf("\tsize error : mc1>INT_MAX\n");
289         return false;
290       }
291       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
292         printf("\tsize error : mc2>INT_MAX\n");
293         return false;
294       }
295       double TotalMemory=0.0; // = mc * (double)sizeof(REAL_TYPE);
296       if( m_param->Get_OutputDataType() == CIO::E_CIO_FLOAT32 ) {
297         TotalMemory = TotalMemory + mc1 * (double)sizeof(float);
298       } else {
299         TotalMemory = TotalMemory + mc1 * (double)sizeof(double);
300       }
301
302       if( l_d_type == SPH_FLOAT ) {
303         TotalMemory = TotalMemory + mc2 * (double)sizeof(float);
304       } else {
305         TotalMemory = TotalMemory + mc2 * (double)sizeof(double);
306       }
307       LOG_OUT_ MemoryRequirement(TotalMemory,m_fplog);
308       STD_OUT_ MemoryRequirement(TotalMemory,stdout);
309
310       int szS[3];
311       szS[0]=l_imax_th;
312       szS[1]=l_jmax_th;
313       szS[2]=1;
314
315       CIO::E_CIO_ARRAYSHAPE output_AShape = m_param->Get_OutputArrayShape();
316
317       if( output_AShape == CIO::E_CIO_NIJK ||
318           DFI_FInfo->Component == 1 ){
319
320         //output nijk
321         if( !convMx1_out_nijk(fp,
322                               inPath,
323                               l_step,
324                               l_dtime,
325                               d_type,
326                               mio,
327                               div,
```

```
328                             szS,
329                             //m_stepList[i].dfi,
330                             m_StepRankList[i].dfi,
331                             DFI_Process,
332                             mapHeadX,mapHeadY,mapHeadZ,
333                             min,max
334                             ) ) return false;
335
336       } else {
337         //output IJKN
338         if( !convMx1_out_ijkn(fp,
339                             inPath,
340                             l_step,
341                             l_dtime,
342                             d_type,
343                             mio,
344                             div,
345                             szS,
346                             //m_stepList[i].dfi,
347                             m_StepRankList[i].dfi,
348                             DFI_Process,
349                             mapHeadX,mapHeadY,mapHeadZ,
350                             min,max
351                             ) ) return false;
352
353       }
354
355       //dfi ごとに minmax を登録
356       for(int ndfi = 0; ndfi<minmaxList.size(); ndfi++) {
357         if( minmaxList[ndfi]->dfi != m_StepRankList[i].dfi ) continue;
358         for(int n=0; n<nsize; n++) {
359           if( minmaxList[ndfi]->Min[j*nsize+n] > min[n] ) minmaxList[ndfi]->Min[j*nsize+n] = min[n];
360           if( minmaxList[ndfi]->Max[j*nsize+n] < max[n] ) minmaxList[ndfi]->Max[j*nsize+n] = max[n];
361         }
362       }
363
364       //データのフッタ書き込み
365       if( !(ConvOut->WriteDataMarker(dummy, fp)) ) {
366         printf("\twrite data error\n");
367         return false;
368       }
369
370       //出力ファイルクローズ
371       ConvOut->OutputFile_Close(fp);
372
373     }
374   }
375
376   //avs のヘッダーファイル出力
377   ConvOut->output_avs(m_myRank, m_in_dfi);
378
379   //出力 dfi ファイル名の取得
380   //vector<std::string> out_dfi_name = m_InputCntl->Get_OutdfiNameList();
381   //vector<std::string> out_proc_name = m_InputCntl->Get_OutprocNameList();
382
383   //出力 dfi ファイルの出力
384   //if( out_dfi_name.size() == 0 || out_proc_name.size() == 0 ) return true;
385   if( !m_param->Get_Outputdfi_on() ) return true;
386
387   //ランク間で通信して MINMAX を求めてランク０に送信
388   for(int i=0; i<minmaxList.size(); i++) {
389    int nComp = minmaxList[i]->dfi->GetNumComponent();
390    const cio_TimeSlice* TSlice = minmaxList[i]->dfi->GetcioTimeSlice();
391    int nStep = TSlice->SliceList.size();
392
393    int n = nComp*nStep;
394    if( nComp > 1 ) n = (nComp+1)*nStep;
395
396     //min の通信
397    double *send1 =  minmaxList[i]->Min;
398    double *recv1 = new double[n];
399    MPI_Reduce(send1, recv1, n, MPI_DOUBLE, MPI_MIN, 0, MPI_COMM_WORLD);
400    minmaxList[i]->Min = recv1;
401
402    //max の通信
403    double *send2 = minmaxList[i]->Max;
404    double *recv2 = new double[n];
405    MPI_Reduce(send2, recv2, n, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
406    minmaxList[i]->Max = recv2;
407
408   }
409
410   if( m_myRank == 0 ) {
411
412     WriteIndexDfiFile(minmaxList);
413
414     for(int i=0; i<m_in_dfi.size(); i++) {
```

```
415       cio_Domain* out_domain = NULL;
416       cio_MPI* out_mpi = NULL;
417       cio_Process* out_process = NULL;
418       const cio_MPI* dfi_mpi = m_in_dfi[i]->GetcioMPI();
419       int numProc = dfi_mpi->NumberOfRank;
420
421       //Proc 情報の生成
422       makeProcInfo(m_in_dfi[i],out_domain,out_mpi,out_process,1);
423
424       //Proc ファイル出力
425       //WriteProcDfiFile(out_proc_name[i],out_domain,out_mpi,out_process);
426       WriteProcDfiFile(m_param->m_dfiList[i].out_proc_name,
427                        out_domain,out_mpi,out_process);
428     }
429   }
430
431   return true;
432
433 }
```

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 1010 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().

```
1012 {
1013
1014   if( !src_old || !src || !outArray ) return false;
1015   //if( !src_old || !src ) return NULL;
1016
1017   //データタイプの取得
1018   //int nComp = src->getNcomp();
1019   CIO::E_CIO_DTYPE dtype = src->getDataType();
1020
1021   //char
1022   if( dtype == CIO::E_CIO_INT8 ) {
1023     cio_TypeArray<char> *O = dynamic_cast<cio_TypeArray<char>*>(outArray);
1024     cio_TypeArray<char> *S = dynamic_cast<cio_TypeArray<char>*>(src);
1025     cio_TypeArray<char> *S_old = dynamic_cast<cio_TypeArray<char>*>(src_old);
1026
1027     //足しこみ領域のゼロクリア
1028     zeroClearArray(O,ivar_out);
1029     //src の足しこみ
1030     setGridData_XY(O,S,     ivar_out,ivar_src);
1031     //src_old の足しこみ
1032     setGridData_XY(O,S_old,ivar_out,ivar_src);
1033     //平均化（8で割る）
1034     VolumeDataDivide8(O,ivar_out);
1035   }
1036   //short
1037   else if( dtype == CIO::E_CIO_INT16 ) {
1038     cio_TypeArray<short> *O = dynamic_cast<cio_TypeArray<short>*>(outArray);
1039     cio_TypeArray<short> *S = dynamic_cast<cio_TypeArray<short>*>(src);
1040     cio_TypeArray<short> *S_old = dynamic_cast<cio_TypeArray<short>*>(src_old);
1041
1042     //足しこみ領域のゼロクリア
1043     zeroClearArray(O,ivar_out);
1044     //src の足しこみ
1045     setGridData_XY(O,S,     ivar_out,ivar_src);
1046     //src_old の足しこみ
1047     setGridData_XY(O,S_old,ivar_out,ivar_src);
1048     //平均化（8で割る）
1049     VolumeDataDivide8(O,ivar_out);
1050   }
1051   //int
```

```
1052   else if( dtype == CIO::E_CIO_INT32 ) {
1053     cio_TypeArray<int> *O = dynamic_cast<cio_TypeArray<int>*>(outArray);
1054     cio_TypeArray<int> *S = dynamic_cast<cio_TypeArray<int>*>(src);
1055     cio_TypeArray<int> *S_old = dynamic_cast<cio_TypeArray<int>*>(src_old);
1056
1057     //足しこみ領域のゼロクリア
1058     zeroClearArray(O,ivar_out);
1059     //src の足しこみ
1060     setGridData_XY(O,S,    ivar_out,ivar_src);
1061     //src_old の足しこみ
1062     setGridData_XY(O,S_old,ivar_out,ivar_src);
1063     //平均化（8で割る）
1064     VolumeDataDivide8(O,ivar_out);
1065   }
1066   //float
1067   else if( dtype == CIO::E_CIO_FLOAT32 ) {
1068     cio_TypeArray<float> *O = dynamic_cast<cio_TypeArray<float>*>(outArray);
1069     cio_TypeArray<float> *S = dynamic_cast<cio_TypeArray<float>*>(src);
1070     cio_TypeArray<float> *S_old = dynamic_cast<cio_TypeArray<float>*>(src_old);
1071
1072     //足しこみ領域のゼロクリア
1073     zeroClearArray(O,ivar_out);
1074     //src の足しこみ
1075     setGridData_XY(O,S,    ivar_out,ivar_src);
1076     //src_old の足しこみ
1077     setGridData_XY(O,S_old,ivar_out,ivar_src);
1078
1079     //平均化（8で割る）
1080     VolumeDataDivide8(O,ivar_out);
1081   }
1082   //double
1083   else if( dtype == CIO::E_CIO_FLOAT64 ) {
1084     cio_TypeArray<double> *O = dynamic_cast<cio_TypeArray<double>*>(outArray);
1085     cio_TypeArray<double> *S = dynamic_cast<cio_TypeArray<double>*>(src);
1086     cio_TypeArray<double> *S_old = dynamic_cast<cio_TypeArray<double>*>(src_old);
1087
1088     //足しこみ領域のゼロクリア
1089     zeroClearArray(O,ivar_out);
1090     //src の足しこみ
1091     setGridData_XY(O,S,    ivar_out,ivar_src);
1092     //src_old の足しこみ
1093     setGridData_XY(O,S_old,ivar_out,ivar_src);
1094     //平均化（8で割る）
1095     VolumeDataDivide8(O,ivar_out);
1096   }
1097
1098   return outArray;
1099
1100 }
```

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 1105 行で定義されています。

参照先 copyArray_nijk_ijk().

参照元 convMx1_out_ijkn().

```
1106 {
1107
1108   //コピー元配列のサイズとデータタイプの取得
1109   const int *sz = src->getArraySizeInt();
1110   CIO::E_CIO_DTYPE d_type = src->getDataType();
1111
1112   cio_Array* outArray = cio_Array::instanceArray
1113                         ( d_type
```

```
1114                        , CIO::E_CIO_IJKN
1115                        , (int *)sz
1116                        , 0
1117                        , 1 );
1118   //unsigned char
1119   if( d_type == CIO::E_CIO_UINT8 ) {
1120     cio_TypeArray<unsigned char> *S = dynamic_cast<cio_TypeArray<unsigned char>*>(src);
1121     cio_TypeArray<unsigned char> *O = dynamic_cast<cio_TypeArray<unsigned char>*>(outArray);
1122     copyArray_nijk_ijk(S,O,ivar);
1123   }
1124   //char
1125   else if( d_type == CIO::E_CIO_INT8 ) {
1126     cio_TypeArray<char> *S = dynamic_cast<cio_TypeArray<char>*>(src);
1127     cio_TypeArray<char> *O = dynamic_cast<cio_TypeArray<char>*>(outArray);
1128     copyArray_nijk_ijk(S,O,ivar);
1129   }
1130   //unsigned short
1131   else if( d_type == CIO::E_CIO_UINT16 ) {
1132     cio_TypeArray<unsigned short> *S = dynamic_cast<cio_TypeArray<unsigned short>*>(src);
1133     cio_TypeArray<unsigned short> *O = dynamic_cast<cio_TypeArray<unsigned short>*>(outArray);
1134     copyArray_nijk_ijk(S,O,ivar);
1135   }
1136   //short
1137   else if( d_type == CIO::E_CIO_INT16 ) {
1138     cio_TypeArray<short> *S = dynamic_cast<cio_TypeArray<short>*>(src);
1139     cio_TypeArray<short> *O = dynamic_cast<cio_TypeArray<short>*>(outArray);
1140     copyArray_nijk_ijk(S,O,ivar);
1141   }
1142   //unsigned int
1143   else if( d_type == CIO::E_CIO_UINT32 ) {
1144     cio_TypeArray<unsigned int> *S = dynamic_cast<cio_TypeArray<unsigned int>*>(src);
1145     cio_TypeArray<unsigned int> *O = dynamic_cast<cio_TypeArray<unsigned int>*>(outArray);
1146     copyArray_nijk_ijk(S,O,ivar);
1147   }
1148   //int
1149   else if( d_type == CIO::E_CIO_INT32 ) {
1150     cio_TypeArray<int> *S = dynamic_cast<cio_TypeArray<int>*>(src);
1151     cio_TypeArray<int> *O = dynamic_cast<cio_TypeArray<int>*>(outArray);
1152     copyArray_nijk_ijk(S,O,ivar);
1153   }
1154   //unsigned long
1155   else if( d_type == CIO::E_CIO_UINT64 ) {
1156     cio_TypeArray<unsigned long long> *S = dynamic_cast<cio_TypeArray<unsigned long long>*>(src);
1157     cio_TypeArray<unsigned long long> *O = dynamic_cast<cio_TypeArray<unsigned long long>*>(outArray);
1158     copyArray_nijk_ijk(S,O,ivar);
1159   }
1160   //long
1161   else if( d_type == CIO::E_CIO_INT64 ) {
1162     cio_TypeArray<long long> *S = dynamic_cast<cio_TypeArray<long long>*>(src);
1163     cio_TypeArray<long long> *O = dynamic_cast<cio_TypeArray<long long>*>(outArray);
1164     copyArray_nijk_ijk(S,O,ivar);
1165   }
1166   //float
1167   else if( d_type == CIO::E_CIO_FLOAT32 ) {
1168     cio_TypeArray<float> *S = dynamic_cast<cio_TypeArray<float>*>(src);
1169     cio_TypeArray<float> *O = dynamic_cast<cio_TypeArray<float>*>(outArray);
1170     copyArray_nijk_ijk(S,O,ivar);
1171   }
1172   //double
1173   else if( d_type == CIO::E_CIO_FLOAT64 ) {
1174     cio_TypeArray<double> *S = dynamic_cast<cio_TypeArray<double>*>(src);
1175     cio_TypeArray<double> *O = dynamic_cast<cio_TypeArray<double>*>(outArray);
1176     copyArray_nijk_ijk(S,O,ivar);
1177   }
1178
1179   return outArray;
1180 }
```

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().

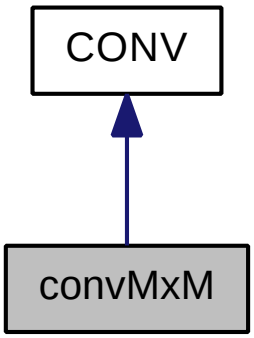
このクラスの説明は次のファイルから生成されました:

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

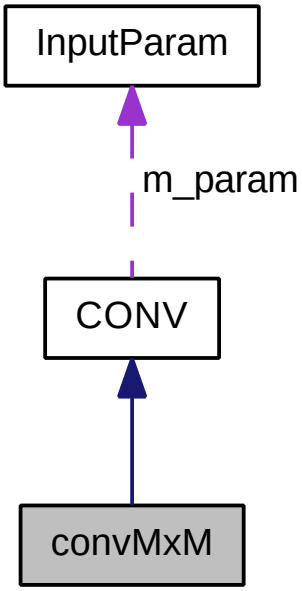
4.3 クラス convMxM

```
#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_StepRankList.clear();
25
26 }
```

4.3.2.2 convMxM::～convMxM ( )

デストラクタ

convMxM.C の 30 行で定義されています。

```
31 {
32
33 }
```

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 37 行で定義されています。

参照先 CONV::dfi_MinMax::dfi, E_CONV_OUTPUT_CAST_UNKNOWN, E_CONV_OUTPUT_RANK, E_CONV_-OUTPUT_STEP, InputParam::Get_MultiFileCasting(), InputParam::Get_Outputdfi_on(), InputParam::m_dfiList, CONV::m_in_dfi, CONV::m_myRank, CONV::m_param, m_StepRankList, CONV::makeProcInfo(), CONV::makeRankList(), CONV::makeStepList(), mxmsolv(), CONV::WriteIndexDfiFile(), と CONV::WriteProcDfiFile().

```
38 {
39
40   if( m_myRank == 0 ) {
41     printf("Convert M x M\n");
42   }
43
44   //並列実行時のファイル割振り方法の取得
45   E_CONV_OUTPUT_MULTI_FILE_CAST outlist = m_param->Get_MultiFileCasting();
46
47   //並列実行時のファイル割振り方法でのリスト生成
48   if( outlist == E_CONV_OUTPUT_STEP || outlist == E_CONV_OUTPUT_CAST_UNKNOWN ) {
49     makeStepList(m_StepRankList);
50   } else if( outlist == E_CONV_OUTPUT_RANK ) {
51     makeRankList(m_StepRankList);
52   }
53
```

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

```
141    MPI_Reduce(send2, recv2, n, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
142    minmaxList[i]->Max = recv2;
143
144   }
145
146   //出力 dfi ファイルの出力
147   if( m_myRank == 0 ) {
148     WriteIndexDfiFile(minmaxList);
149
150     for(int i=0; i<m_in_dfi.size(); i++) {
151       cio_Domain* out_domain = NULL;
152       cio_MPI* out_mpi = NULL;
153       cio_Process* out_process = NULL;
154       const cio_MPI* dfi_mpi = m_in_dfi[i]->GetcioMPI();
155       int numProc = dfi_mpi->NumberOfRank;
156
157       //Proc 情報の生成
158       makeProcInfo(m_in_dfi[i],out_domain,out_mpi,out_process,numProc);
159
160       //Proc ファイル出力
161       //WriteProcDfiFile(out_proc_name[i],out_domain,out_mpi,out_process);
162       WriteProcDfiFile(m_param->m_dfiList[i].out_proc_name,
163                        out_domain,out_mpi,out_process);
164     }
165
166   }
167
168   return true;
169
170 }
```

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 174 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_param.

参照元 exec().

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

```
218     voxel[i]=DFI_Domain->GlobalVoxel[i]/thin_count;
219     if( DFI_Domain->GlobalVoxel[i]%thin_count != 0 ) voxel[i]++;
220   }
221
222   //間引きを考慮したサイズのセット
223   int l_imax_th = DFI_Process->RankList[RankID].VoxelSize[0]/thin_count;
224   int l_jmax_th = DFI_Process->RankList[RankID].VoxelSize[1]/thin_count;
225   int l_kmax_th = DFI_Process->RankList[RankID].VoxelSize[2]/thin_count;
226
227   //間引き後のサイズが 1 つも無い領域のときエラー
228   if( l_imax_th < 1 || l_jmax_th < 1 || l_kmax_th < 1 ) {
229     printf("\toutput domain size error\n");
230     return false;
231   }
232
233   if(DFI_Process->RankList[RankID].VoxelSize[0]%thin_count != 0) l_imax_th++;
234   if(DFI_Process->RankList[RankID].VoxelSize[1]%thin_count != 0) l_jmax_th++;
235   if(DFI_Process->RankList[RankID].VoxelSize[2]%thin_count != 0) l_kmax_th++;
236
237   //間引き後の head インデックスを求める
238   int head[3];
239   head[0] = (DFI_Process->RankList[RankID].HeadIndex[0]-1)/thin_count;
240   head[1] = (DFI_Process->RankList[RankID].HeadIndex[1]-1)/thin_count;
241   head[2] = (DFI_Process->RankList[RankID].HeadIndex[2]-1)/thin_count;
242   if( (DFI_Process->RankList[RankID].HeadIndex[0]-1)%thin_count != 0 ) head[0]++;
243   if( (DFI_Process->RankList[RankID].HeadIndex[1]-1)%thin_count != 0 ) head[1]++;
244   if( (DFI_Process->RankList[RankID].HeadIndex[2]-1)%thin_count != 0 ) head[2]++;
245   //間引き後のオリジンを求める
246   double l_dorg[3];
247   l_dorg[0]= DFI_Domain->GlobalOrigin[0]+head[0]*out_dpit[0];
248   l_dorg[1]= DFI_Domain->GlobalOrigin[1]+head[1]*out_dpit[1];
249   l_dorg[2]= DFI_Domain->GlobalOrigin[2]+head[2]*out_dpit[2];
250
251   //出力タイプのセット
252   CIO::E_CIO_DTYPE d_type;
253   if( m_param->Get_OutputDataType() == CIO::E_CIO_DTYPE_UNKNOWN )
254   {
255     d_type = dfi->GetDataType();
256   } else {
257     d_type = m_param->Get_OutputDataType();
258   }
259
260   //出力バッファのインスタンス
261   int szS[3];
262   szS[0]=l_imax_th;
263   szS[1]=l_jmax_th;
264   szS[2]=l_kmax_th;
265   cio_Array* src = cio_Array::instanceArray
266                  ( d_type
267                  , m_param->Get_OutputArrayShape()
268                  , szS
269                  , outGc
270                  , dfi->GetNumComponent() );
271
272   //読込みファイル名の生成
273   std::string inPath = CIO::cioPath_DirName(dfiname);
274   std::string infile =  CIO::cioPath_ConnectPath(inPath,dfi->Generate_FieldFileName(
275                         RankID,l_step,mio));
276
277   //ファイルの読込み
278   unsigned int avr_step;
279   double l_dtime, avr_time;
280   CIO::E_CIO_ERRORCODE ret;
281   int read_sta[3],read_end[3];
282   for(int i=0; i<3; i++) {
283     read_sta[i]=DFI_Process->RankList[RankID].HeadIndex[i]-outGc;
284     read_end[i]=DFI_Process->RankList[RankID].TailIndex[i]+outGc;
285   }
286
287   cio_Array* buf = dfi->ReadFieldData(infile, l_step, l_dtime,
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
309     for(int n=0; n<dfi->GetNumComponent(); n++) convertXY(buf,src,headS,tailS,n);
310     //delete buf;
311   }
312
313   //出力 DFI のインスタンス
314   int tail[3];
315   head[0]=head[0]+1;
316   head[1]=head[1]+1;
317   head[2]=head[2]+1;
318   tail[0]=head[0]+l_imax_th-1;
319   tail[1]=head[1]+l_jmax_th-1;
320   tail[2]=head[2]+l_kmax_th-1;
321   cio_DFI* out_dfi = cio_DFI::WriteInit(
322                      MPI_COMM_WORLD,
323                      "",
324                      m_param->Get_OutputDir(),
325                      DFI_FInfo->Prefix,
326                      m_param->Get_OutputFormat(),
327                      outGc,
328                      d_type,
329                      m_param->Get_OutputArrayShape(),
330                      DFI_FInfo->Component,
331                      "",
332                      voxel,
333                      out_dpit,
334                      l_dorg,
335                      DFI_Domain->GlobalDivision,
336                      head,
337                      tail,
338                      m_HostName,
339                      CIO::E_CIO_OFF);
340   if( out_dfi == NULL ) {
341     printf("\tFails to instance dfi\n");
342     return false;
343   }
344
345   out_dfi->set_RankID(RankID);
346   out_dfi->SetcioMPI(*DFI_MPI);
347
348   //cio_Process の作成＆更新
349   cio_Process out_Process;
350   cio_Rank out_Rank;
351   for(int i=0; i<DFI_Process->RankList.size(); i++) {
352     out_Rank.RankID = DFI_Process->RankList[i].RankID;
353     out_Rank.HostName = "";
354     for(int j=0; j<3; j++) {
355       out_Rank.HeadIndex[j]=head[j];
356       out_Rank.TailIndex[j]=tail[j];
357       out_Rank.VoxelSize[j]=tail[j]-head[j]+1;
358     }
359     out_Process.RankList.push_back(out_Rank);
360   }
361
362   //out_dfi->SetcioProcess(*DFI_Process);
363   out_dfi->SetcioProcess(out_Process);
364   out_dfi->SetcioTimeSlice(*TSlice);
365
366   //出力
367   out_dfi->set_output_type(m_param->Get_OutputFormatType());
368   CIO::E_CIO_OUTPUT_FNAME output_fname = m_param->Get_OutputFilenameFormat();
369   out_dfi->set_output_fname(output_fname);
370   double tmp_minmax[8];
371   unsigned idummy=0;
372   double ddummy=0.0;
373   ret = out_dfi->WriteData(
374                            (unsigned)l_step,
375                            //0,
376                            outGc,
377                            l_time,
378                            src,
379                            tmp_minmax,
380                            true,
381                            idummy,
382                            ddummy);
383
384   if( ret != CIO::E_CIO_SUCCESS ) return false;
385
386   //minmax を求める
387   if( !DtypeMinMax(src,min,max) ) return false;
388
389   //delete
390   delete out_dfi;
391   delete src;
```

```
392
393   return true;
394 }
```

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().

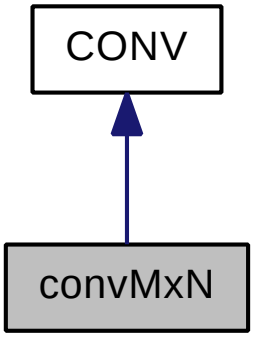
このクラスの説明は次のファイルから生成されました:

- convMxM.h
- convMxM.C

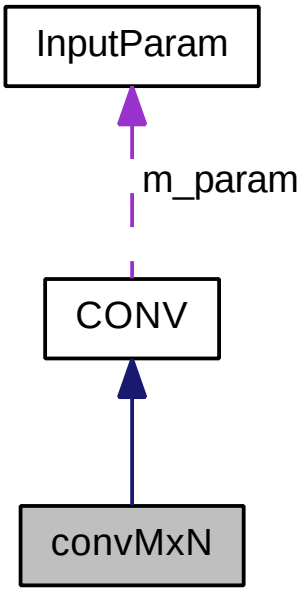
4.4 クラス convMxN

`#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 261 行で定義されています。

参照先 CONV::convertXY(), CONV::DtypeMinMax(), InputParam::Get_CropIndexEnd(), InputParam::Get_CropIndexEnd_on(), InputParam::Get_CropIndexStart(), InputParam::Get_CropIndexStart_on(), InputParam::Get_OutputArrayShape(), InputParam::Get_OutputDataType(), InputParam::Get_Outputdfi_on(), 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_myRank, CONV::m_numProc, m_out_dfi, CONV::m_param, CONV::m_paraMngr, m_Tail, と convOutput::OutputInit().

```
262 {
263
264   if( m_myRank == 0 ) {
265     printf("Convert M x N\n");
266   }
267
268
269   // 出力ファイル形式クラスのインスタンス
270   convOutput *ConvOut = convOutput::OutputInit(m_param->Get_OutputFormat());
271
272   // InputParam のインスタンス
273   if( !ConvOut->importInputParam(m_param) ) {
274     //Exit(0);
275     return false;
276   }
277
278   //出力ファイル名の取得
279   //vector<std::string> out_dfi_name = m_InputCntl->Get_OutdfiNameList();
280   std::string prefix,outfile;
281
282   FILE *fp;
283   int dummy;
284
285   CIO::E_CIO_DTYPE d_type;
286
287   CIO::E_CIO_ERRORCODE ret;
288   double rtime;
289   unsigned idummy;
290   double ddummy;
291   float fminmax[8];
292   double dminmax[8];
293
```

```
294   bool mio;
295   mio = false;
296   if( m_numProc > 1 ) mio=true;
297
298   //間引き数のセット
299   int thin_count = m_param->Get_ThinOut();
300
301   //入力領域指示のセット
302   int IndexStart[3];
303   int IndexEnd[3];
304   for(int i=0; i<3; i++) IndexStart[i]=m_Head[i];
305   for(int i=0; i<3; i++) IndexEnd[i]=m_Tail[i];
306   if( m_param->Get_CropIndexStart_on() ) {
307     const int *cropIndexStart = m_param->Get_CropIndexStart();
308     for(int i=0; i<3; i++) {
309       if( IndexStart[i] < cropIndexStart[i] ) IndexStart[i]=cropIndexStart[i];
310     }
311   }
312   if( m_param->Get_CropIndexEnd_on() ) {
313     const int *cropIndexEnd = m_param->Get_CropIndexEnd();
314     for(int i=0; i<3; i++) {
315       if( IndexEnd[i] > cropIndexEnd[i] ) IndexEnd[i]=cropIndexEnd[i];
316     }
317   }
318
319   //自ノードのボクセルサイズの取得
320   int sz[3];
321   const int* tmp = m_paraMngr->GetLocalVoxelSize();
322   for(int i=0; i<3; i++) sz[i]=tmp[i];
323
324   //自ノードのボクセルサイズを入力指示を考慮して更新
325   for(int i=0; i<3; i++) {
326     if( sz[i] > (IndexEnd[i]-IndexStart[i]+1) ) sz[i]=(IndexEnd[i]-IndexStart[i]+1);
327   }
328
329   //出力 workarea のサイズ
330   int szS[3];
331   const int *cropIndexStart = m_param->Get_CropIndexStart();
332   for(int i=0; i<3; i++) {
333     szS[i]=sz[i]/thin_count;
334     if( szS[i] < 1 ) {
335       printf("\toutput domain size error\n");
336       return false;
337     }
338     if( m_param->Get_CropIndexStart_on() ) {
339       if( IndexStart[i] == cropIndexStart[i] ) {
340         if( sz[i]%thin_count != 0 ) szS[i]++;
341       }
342     } else {
343       if( sz[i]%thin_count != 0 ) szS[i]++;
344     }
345   }
346
347   int head[3],tail[3];
348   for(int i=0; i<3; i++) {
349     head[i]=(m_Head[i]-1)/thin_count;
350     if( (m_Head[i]-1)%thin_count != 0 ) head[i]++;
351     tail[i]=(m_Tail[i]-1)/thin_count;
352   }
353   const double* dtmp;
354   double pit[3],org[3];
355   dtmp = m_paraMngr->GetPitch();
356   for(int i=0; i<3; i++) pit[i]=dtmp[i]*double(thin_count);
357   dtmp = m_paraMngr->GetGlobalOrigin();
358   for(int i=0; i<3; i++) org[i]=dtmp[i]+0.5*pit[i];
359   for(int i=0; i<3; i++) org[i]+=double(head[i])*pit[i];
360
361   const cio_FileInfo* DFI_FInfo = m_in_dfi[0]->GetcioFileInfo();
362
363   //dfi のループ
364   for (int i=0; i<m_in_dfi.size(); i++) {
365
366
367     int nComp = m_in_dfi[i]->GetNumComponent();
368
369     int outGc=0;
370     if( m_param->Get_OutputGuideCell() > 1 ) outGc = m_param->
    Get_OutputGuideCell();
371     if( outGc > 0 ) {
372       const cio_FileInfo* DFI_FInfo = m_in_dfi[i]->GetcioFileInfo();
373       if( outGc > DFI_FInfo->GuideCell ) outGc = DFI_FInfo->GuideCell;
374     }
375
376     if( thin_count > 1 ) outGc=0;
377     if( m_bgrid_interp_flag ) outGc=0;
378
379     //読込みバッファのインスタンス
```

```
380     cio_Array* buf = cio_Array::instanceArray
381     ( m_in_dfi[i]->GetDataType(),
382       m_in_dfi[i]->GetArrayShape(),
383       sz,
384       //0,
385       outGc,
386       //m_in_dfi[i]->GetNumComponent());
387       nComp);
388
389     //出力タイプのセット
390     if( m_param->Get_OutputDataType() == CIO::E_CIO_DTYPE_UNKNOWN )
391     {
392       d_type = m_in_dfi[i]->GetDataType();
393     } else {
394       d_type = m_param->Get_OutputDataType();
395     }
396
397     //出力バッファのインスタンス
398     cio_Array* src = cio_Array::instanceArray
399     ( d_type,
400       //m_in_dfi[i]->GetArrayShape(),
401       m_param->Get_OutputArrayShape(),
402       szS,
403       //0,
404       outGc,
405       //m_in_dfi[i]->GetNumComponent());
406       nComp);
407
408     //DFI_FInfo クラスの取得
409     const cio_FileInfo* DFI_FInfo = m_in_dfi[i]->GetcioFileInfo();
410     prefix=DFI_FInfo->Prefix;
411
412     //TimeSlice クラスの取得
413     const cio_TimeSlice* TSlice = m_in_dfi[i]->GetcioTimeSlice();
414
415
416     //ステップ数のループ
417     for ( int j=0; j<TSlice->SliceList.size(); j++ ) {
418
419       //MxN の読込み
420       ret = m_in_dfi[i]->ReadData(buf,
421                                   (unsigned)TSlice->SliceList[j].step,
422                                   //0,
423                                   outGc,
424                                   m_Gvoxel,
425                                   m_Gdiv,
426                                   m_Head,
427                                   m_Tail,
428                                   rtime,
429                                   true,
430                                   idummy,
431                                   ddummy);
432       if( ret != CIO::E_CIO_SUCCESS ) {
433         printf("ReadData Error\n");
434         return false;
435       }
436
437       //読込みバッファの headIndex のセット
438       int headB[3];
439       for(int k=0; k<3; k++) headB[k]=m_Head[k]-1;
440       buf->setHeadIndex( headB );
441
442       //間引き及び型変換がない場合
443       if( thin_count == 1 && buf->getDataType() == src->getDataType() &&
444           buf->getArrayShape() == src->getArrayShape() ) {
445         src=buf;
446       } else {
447       //間引きまたは型変換がある場合
448         //出力バッファの間引きなしでの HeadIndex,TailIndex
449         int headS[3],tailS[3];
450         for(int k=0; k<3; k++) {
451           headS[k]=m_Head[k]-1;
452           tailS[k]=m_Tail[k]-1;
453         }
454         //出力バッファの HeadIndex セット
455         int headS0[3];
456         if( m_param->Get_CropIndexStart_on() ) {
457           for(int k=0; k<3; k++) {
458             headS0[k]=headS[k]/thin_count;
459           }
460         } else {
461           for(int k=0; k<3; k++) {
462             headS0[k]=headS[k]/thin_count;
463             if( headS[k]%thin_count != 0 ) headS0[k]++;
464           }
465         }
466
```

```
467         src->setHeadIndex( headS0 );
468
469         for(int n=0; n<nComp; n++) convertXY(buf,src,headS,tailS,n);
470       }
471
472       CIO::E_CIO_OUTPUT_FNAME output_fname = m_param->Get_OutputFilenameFormat();
473       m_out_dfi[i]->set_output_fname(output_fname);
474
475       //minmax の初期化
476       int nsize = nComp;
477       if( nComp > 1 ) nsize++;
478       double *min = new double[nsize];
479       double *max = new double[nsize];
480       for(int n=0; n<nsize; n++) {
481         min[n]=DBL_MAX;
482         max[n]=-DBL_MAX;
483       }
484       //minmax を求める
485       if( !DtypeMinMax(src,min,max) ) return false;
486
487       //if( out_dfi_name.size() > 1 ) {
488       if( m_param->Get_Outputdfi_on() ) {
489         //ランク間で通信して MINMAX を求めてランク 0 に送信
490         int nbuff = nsize*1;
491         //min の通信
492         double *send1 = min;
493         double *recv1 = new double[nbuff];
494         MPI_Reduce(send1, recv1, nbuff, MPI_DOUBLE, MPI_MIN, 0, MPI_COMM_WORLD);
495         min = recv1;
496         //max の通信
497         double *send2 = max;
498         double *recv2 = new double[nbuff];
499         MPI_Reduce(send2, recv2, nbuff, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
500         max = recv2;
501       }
502
503       //出力処理
504       double *tmp_minmax = new double[nsize*2];
505       for(int n=0; n<nsize; n++ ) {
506         tmp_minmax[n*2+0] = min[n];
507         tmp_minmax[n*2+1] = max[n];
508       }
509
510       m_out_dfi[i]->SetcioTimeSlice(*TSlice);
511
512       ret = m_out_dfi[i]->WriteData(
513                                     (unsigned)TSlice->SliceList[j].step,
514                                     //0,
515                                     outGc,
516                                     rtime,
517                                     src,
518                                     tmp_minmax,
519                                     true,
520                                     idummy,
521                                     ddummy);
522
523
524     }
525     delete src;
526   }
527
528   return true;
529
530 }
```

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_CropIndexEnd(), InputParam::Get_CropIndexEnd_on(), InputParam::Get_CropIndexStart(), InputParam::Get_CropIndexStart_on(), InputParam::Get_OutputArrayShape(), InputParam::Get_OutputDataType(), InputParam::Get_Outputdfi_on(), InputParam::Get_OutputDir(), InputParam::Get_OutputDivision(), InputParam::Get_OutputFormat(), InputParam::Get_OutputFormatType(), InputParam::Get_OutputGuideCell(), InputParam::Get_ThinOut(), CONV::m_bgrid_interp_flag, InputParam::m_dfiList, m_Gdiv, m_Gvoxel, m_Head, CONV::m_HostName, CONV::m_in_dfi, m_out_dfi, CONV::m_param, 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   //ピッチのセット
42   double dfi_pit[3];
43   for(int i=0; i<3; i++) dfi_pit[i]=DFI_Domain->GlobalRegion[i]/(double)DFI_Domain->GlobalVoxel[i];
44
45   //入力領域指示のセット
46   int IndexStart[3];
47   int IndexEnd[3];
48   for(int i=0; i<3; i++) IndexStart[i]=1;
49   for(int i=0; i<3; i++) IndexEnd[i]=DFI_Domain->GlobalVoxel[i];
50   if( m_param->Get_CropIndexStart_on() ) {
51     const int *cropIndexStart = m_param->Get_CropIndexStart();
52     for(int i=0; i<3; i++) IndexStart[i]=cropIndexStart[i];
53   }
54   if( m_param->Get_CropIndexEnd_on() ) {
55     const int *cropIndexEnd = m_param->Get_CropIndexEnd();
56     for(int i=0; i<3; i++) IndexEnd[i]=cropIndexEnd[i];
57   }
58
59   //全体のサイズをセット
60   int voxel[3];
61   for(int i=0; i<3; i++) voxel[i]=IndexEnd[i]-IndexStart[i]+1;
62
63   //リージョンのセット
64   double region[3];
65   for(int i=0; i<3; i++) region[i]=voxel[i]*dfi_pit[i];
66
67   //出力領域の分割数の取得
68   int* Gdiv = m_param->Get_OutputDivision();
69
70   if( Gdiv[0]>0 && Gdiv[1]>0 && Gdiv[2]>0 ) {
71     //分割数が指示されている場合
72     /*
73     iret = m_paraMngr->VoxelInit(Gdiv, (int *)DFI_Domain->GlobalVoxel,
74                                  (double *)DFI_Domain->GlobalOrigin,
75                                  (double *)DFI_Domain->GlobalRegion, 0, 0);
76     */
77     iret = m_paraMngr->VoxelInit(Gdiv, voxel,
78                                    (double *)DFI_Domain->GlobalOrigin,
79                                    region, 0, 0);
80     if( iret != 0 ) {
81        printf("\tVoxelInit Error cpm_ErrorCode : %d\n",iret);
82        Exit(0);
83     }
84   } else {
85     //分割数が指示されていない場合
86     /*
87     iret =  m_paraMngr->VoxelInit((int *)DFI_Domain->GlobalVoxel,
88                                   (double *)DFI_Domain->GlobalOrigin,
89                                   (double *)DFI_Domain->GlobalRegion, 0, 0);
90     */
91     iret =  m_paraMngr->VoxelInit(voxel,
92                                    (double *)DFI_Domain->GlobalOrigin,
93                                    region, 0, 0);
94     if( iret != 0 ) {
95        printf("\tVoxelInit Error cpm_ErrorCode : %d\n",iret);
96        Exit(0);
97     }
98   }
99
100
101   const int* tmp;
102   /*
103   tmp = m_paraMngr->GetGlobalVoxelSize();
104   for(int i=0; i<3; i++) m_Gvoxel[i]=tmp[i];
105   */
106   for(int i=0; i<3; i++) m_Gvoxel[i]=DFI_Domain->GlobalVoxel[i];
107
108   tmp = m_paraMngr->GetVoxelHeadIndex();
109   for(int i=0; i<3; i++) m_Head[i]=tmp[i]+1;
110   tmp = m_paraMngr->GetVoxelTailIndex();
111   for(int i=0; i<3; i++) m_Tail[i]=tmp[i]+1;
112   tmp = m_paraMngr->GetDivNum();
113   for(int i=0; i<3; i++) m_Gdiv[i]=tmp[i];
114
115   //入力指示を考慮したヘッド、テイルに更新
116
117   tmp = m_paraMngr->GetLocalVoxelSize();
118
119   for(int i=0; i<3; i++) {
120     //if( m_Head[i] < IndexStart[i] ) m_Head[i]=IndexStart[i];
121     if( m_Head[i] < IndexStart[i] ) {
122       m_Head[i]=m_Head[i]+IndexStart[i]-1;
```

```
123       m_Tail[i]=m_Head[i]+tmp[i]-1;
124     }
125     if( m_Tail[i] > IndexEnd[i]   ) m_Tail[i]=IndexEnd[i];
126   }
127
128   //間引き数のセット
129   int thin_count = m_param->Get_ThinOut();
130
131   //間引きを考慮した全体サイズのセット
132   int voxel_thin[3];
133   for(int i=0; i<3; i++) {
134     voxel_thin[i]=voxel[i]/thin_count;
135     if( voxel[i]%thin_count != 0 ) voxel_thin[i]++;
136   }
137
138
139   //間引きを考慮したヘッド、テイルインデックスの作成
140   int head[3],tail[3];
141   for(int i=0; i<3; i++) {
142     head[i]=(m_Head[i]-1)/thin_count;
143     if( (m_Head[i]-1)%thin_count != 0 ) head[i]++;
144     tail[i]=(m_Tail[i]-1)/thin_count;
145   }
146
147   const double* dtmp;
148   double pit[3],org[3];
149   //dtmp = m_paraMngr->GetPitch();
150   //for(int i=0; i<3; i++) pit[i]=dtmp[i];
151   //for(int i=0; i<3; i++) pit[i]=dtmp[i]*double(thin_count);
152   pit[0]=region[0]/voxel_thin[0];
153   pit[1]=region[1]/voxel_thin[1];
154   pit[2]=region[2]/voxel_thin[2];
155
156   dtmp = m_paraMngr->GetGlobalOrigin();
157   for(int i=0; i<3; i++) org[i]=dtmp[i];
158
159   const int *tmp_head = m_paraMngr->GetVoxelHeadIndex();
160   const int *tmp_tail = m_paraMngr->GetVoxelTailIndex();
161   for(int i=0; i<3; i++) {
162     head[i]=tmp_head[i]/thin_count;
163     if( tmp_head[i]%thin_count != 0 ) head[i]++;
164     tail[i]=tmp_tail[i]/thin_count;
165   }
166
167   for(int i=0; i<3; i++) org[i]+=double(head[i])*pit[i];
168
169   for(int i=0; i<3; i++) {
170     head[i]=head[i]+1;
171     tail[i]=tail[i]+1;
172   }
173
174   //出力 DFI の初期化
175   for(int i=0; i<m_in_dfi.size(); i++) {
176     const cio_FileInfo* DFI_FInfo = m_in_dfi[i]->GetcioFileInfo();
177
178     std::string outdfifname="";
179     std::string outprocfname="";
180     if( m_param->Get_Outputdfi_on() ) {
181        outdfifname =m_param->m_dfiList[i].out_dfi_name;
182        outprocfname=m_param->m_dfiList[i].out_proc_name;
183     }
184
185     //出力タイプのセット
186     CIO::E_CIO_DTYPE d_type;
187     if( m_param->Get_OutputDataType() == CIO::E_CIO_DTYPE_UNKNOWN )
188     {
189       d_type = m_in_dfi[i]->GetDataType();
190     } else {
191       d_type = m_param->Get_OutputDataType();
192     }
193
194     //出力ガイドセルの設定
195     int outGc=0;
196     if( m_param->Get_OutputGuideCell() > 1 ) outGc = m_param->
    Get_OutputGuideCell();
197     if( outGc > 1 ) {
198       const cio_FileInfo* DFI_FInfo = m_in_dfi[i]->GetcioFileInfo();
199       if( outGc > DFI_FInfo->GuideCell ) outGc=DFI_FInfo->GuideCell;
200     }
201     //間引きありのとき、出力ガイドセルを 0 に設定
202     if( thin_count > 1 ) outGc=0;
203     //格子点出力のとき、出力ガイドセルを 0 に設定
204     if( m_bgrid_interp_flag ) outGc=0;
205
206     cio_DFI* dfi=cio_DFI::WriteInit(MPI_COMM_WORLD,
207                             outdfifname,
208                             m_param->Get_OutputDir(),
```

```
209                           DFI_FInfo->Prefix,
210                           m_param->Get_OutputFormat(),
211                           //0,
212                           outGc,
213                           d_type,
214                           m_param->Get_OutputArrayShape(),
215                           DFI_FInfo->Component,
216                           outprocfname,
217                           voxel_thin,
218                           pit,
219                           org,
220                           m_Gdiv,
221                           head,
222                           tail,
223                           m_HostName,
224                           CIO::E_CIO_OFF);
225     if( dfi == NULL ) {
226       printf("\tFails to instance dfi\n");
227       Exit(0);
228     }
229
230     //Proc ファイル出力
231     if( m_param->Get_Outputdfi_on() ) dfi->WriteProcDfiFile(MPI_COMM_WORLD,false);
232
233     //出力形式 (ascii,binary,Fbinary) のセット
234     dfi->set_output_type(m_param->Get_OutputFormatType());
235
236     //Unit のセット
237     std::string unit;
238     double ref;
239     double diff;
240     bool bdiff;
241     m_in_dfi[i]->GetUnit("Length",unit,ref,diff,bdiff);
242     dfi->AddUnit("Length",unit,ref,diff,bdiff);
243     m_in_dfi[i]->GetUnit("Velocity",unit,ref,diff,bdiff);
244     dfi->AddUnit("Velocity",unit,ref,diff,bdiff);
245     m_in_dfi[i]->GetUnit("Pressure",unit,ref,diff,bdiff);
246     dfi->AddUnit("Pressure",unit,ref,diff,bdiff);
247
248     //成分名の取り出しとセット
249     for(int n=0; n<DFI_FInfo->Component; n++) {
250        std::string variable = m_in_dfi[i]->getComponentVariable(n);
251        if( variable != "" ) dfi->setComponentVariable(n,variable);
252     }
253
254     m_out_dfi.push_back(dfi);
255   }
256
257 }
```

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().

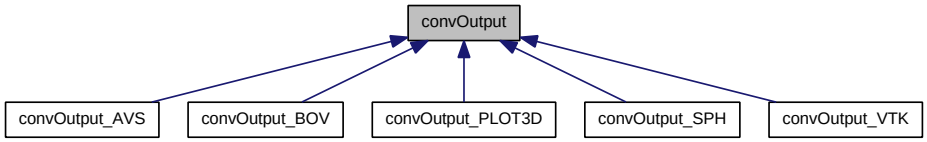
このクラスの説明は次のファイルから生成されました:

- convMxN.h
- convMxN.C

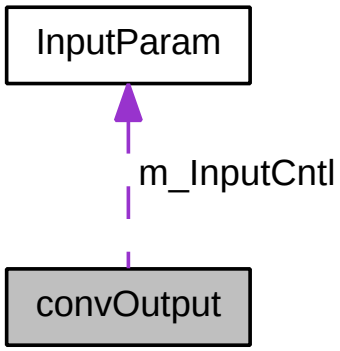
4.5 クラス convOutput

`#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 (std::string prefix, int step, int myRank, int dType, int guide, double org[3], double pit[3], int sz[3])

grid **出力** *(plot3d* **用***)*

- virtual bool WriteHeaderRecord (int step, int dim, CIO::E_CIO_DTYPE d_type, int imax, int jmax, int kmax, double time, double ∗org, double ∗pit, std::string prefix, FILE ∗fp)

  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)

  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 CIO::E_CIO_FORMAT 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*) [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

convOutput_AVSで再定義されています。

convOutput.h の 164 行で定義されています。

参照元 convMx1::exec().

```
166                                     {};
```

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 CIO::E_CIO_FORMAT *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 155 行で定義されています。

参照元 convMx1::exec().

```
155 { 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   //if( src->writeAscii(fp) != dLen ) Exit(0);
72   return true;
73 }
```

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

grid 出力 (plot3d 用)

引数

in	*prefix*	ファイル接頭文字
in	*step*	step 番号
in	*myRank*	ランク番号
in	*dType*	dfi のデータタイプ
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 100 行で定義されています。

参照元 convMx1::exec().

```
107                             {};
```

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

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

convOutput.h の 124 行で定義されています。

参照元 convMx1::exec().

```
135   { 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_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_BOV::WriteHeaderRecord(), convOutput_VTK::WriteHeaderRecord(), convOutput_PLOT3D::WriteNgrid(), と convOutput_PLOT3D::WriteXYZData().

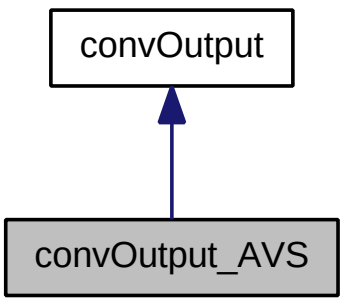
このクラスの説明は次のファイルから生成されました:

- convOutput.h
- convOutput.C

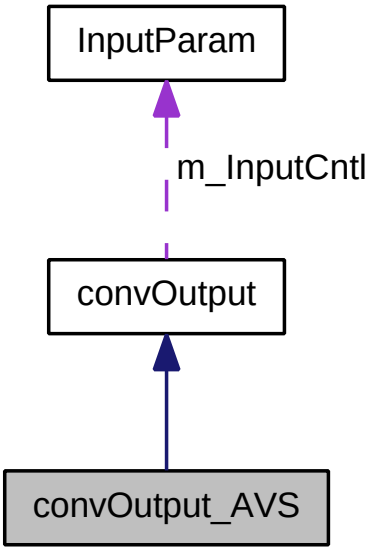
4.6 クラス convOutput_AVS

`#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)

  avs **ファイルのヘッダー処理**

Protected メソッド

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

  avs **ファイルのヘッダー処理（***Mx1)*
- 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*) [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

convOutputを再定義しています。

convOutput_AVS.C の 112 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().

```
114 {
115
116   if( myRank != 0 ) return; //myRank==0 のときのみヘッダーレコードを出力
117
118   //間引き数のセット
119   int thin_count = m_InputCntl->Get_ThinOut();
120
121   int ndim,nspace;
122
123   int dims[3];
124   double min_ext[3],max_ext[3];
125
126   for(int i=0; i<in_dfi.size(); i++) {
127
128     //cio_Domain クラスポインタの取得
129     const cio_Domain* DFI_Domain  = in_dfi[i]->GetcioDomain();
130
131     //間引きを考慮しての計算空間サイズをセット
132     dims[0]=DFI_Domain->GlobalVoxel[0]/thin_count;
133     dims[1]=DFI_Domain->GlobalVoxel[1]/thin_count;
134     dims[2]=DFI_Domain->GlobalVoxel[2]/thin_count;
135     if(DFI_Domain->GlobalVoxel[0]%thin_count != 0 ) dims[0]++;
136     if(DFI_Domain->GlobalVoxel[1]%thin_count != 0 ) dims[1]++;
137     if(DFI_Domain->GlobalVoxel[2]%thin_count != 0 ) dims[2]++;
138
139     if( i==0 ) {
140       //座標値の最小値、最大値をセット
141       for(int j=0; j<3; j++) {
142         double pit=(DFI_Domain->GlobalRegion[j])/(double)(DFI_Domain->GlobalVoxel[j])*thin_count;
143         //min_ext[j]=DFI_Domain->GlobalOrigin[j]+0.5*(pit*(double)thin_count);
```

```
144         //
    max_ext[j]=(DFI_Domain->GlobalOrigin[j]+DFI_Domain->GlobalRegion[j])-0.5*(pit*(double)thin_count);
145         min_ext[j]=DFI_Domain->GlobalOrigin[j];
146         max_ext[j]=DFI_Domain->GlobalOrigin[j]+(pit*(double)dims[j]);
147       }
148
149       //coord データファイル出力
150       output_avs_coord(myRank, false, min_ext, max_ext);
151     }
152
153     //avs のヘッダーファイル出力
154     ndim=3;
155     nspace=3;
156     output_avs_header(in_dfi[i], myRank, false, ndim, nspace, dims);
157
158   }
159
160   return;
161 }
```

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 278 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(), output_avs_MxM(), と output_avs_MxN().

```
282 {
283
284   FILE* fp;
285   std::string cod_fname;
286
287   //座標値データファイルオープン
288   CIO::E_CIO_OUTPUT_FNAME fnameformat = m_InputCntl->Get_OutputFilenameFormat();
289   cod_fname = m_InputCntl->Get_OutputDir() +"/"+
290               cio_DFI::Generate_FileName("cord",
291                                          RankID,
292                                          -1,
293                                          "cod",
294                                          fnameformat,
295                                          mio,
296                                          CIO::E_CIO_OFF);
297
298   if( (fp = fopen(cod_fname.c_str(),"w")) == NULL ) {
299     printf("\tCan't open file.(%s)\n",cod_fname.c_str());
300     Exit(0);
301   }
302
303   //座標値データ（min,max）の出力
304   fprintf(fp,"#### X #####\n");
305   fprintf(fp,"%.6f\n",min_ext[0]);
306   fprintf(fp,"%.6f\n",max_ext[0]);
307   fprintf(fp,"#### Y #####\n");
308   fprintf(fp,"%.6f\n",min_ext[1]);
309   fprintf(fp,"%.6f\n",max_ext[1]);
310   fprintf(fp,"#### Z #####\n");
311   fprintf(fp,"%.6f\n",min_ext[2]);
312   fprintf(fp,"%.6f\n",max_ext[2]);
313
314   //座標値データファイルクローズ
315   fclose(fp);
316
317 }
```

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 320 行で定義されています。

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

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

```
326 {
327
328   FILE* fp;
329   std::string dType;
330   std::string fld_fname, out_fname;
331   std::string cod_fname;
332
333   //cio_FileInfo クラスポインタの取得
334   const cio_FileInfo* DFI_FInfo = dfi->GetcioFileInfo();
335   //cio_TimeSlice クラスポインタの取得
336   const cio_TimeSlice* TSlice   = dfi->GetcioTimeSlice();
337
338   //データタイプのセット
339   int out_dtype = m_InputCntl->Get_OutputDataType();
340   if( out_dtype == CIO::E_CIO_DTYPE_UNKNOWN ) out_dtype = dfi->GetDataType();
341   if(      out_dtype == CIO::E_CIO_INT8    ) {
342     dType="byte";
343   } else if( out_dtype == CIO::E_CIO_INT16   ) {
344     dType="short";
345   } else if( out_dtype == CIO::E_CIO_INT32   ) {
346     dType="integer";
347   } else if( out_dtype == CIO::E_CIO_FLOAT32 ) {
348     dType="float";
349   } else if( out_dtype == CIO::E_CIO_FLOAT64 ) {
350     dType="double";
351   } else {
352     dType = m_InputCntl->Get_OutputDataType_string();
353     printf("\tillergal data type.(%s)\n",dType.c_str());
354     Exit(0);
355   }
356
357   //出力ヘッダーファイルオープン
358   CIO::E_CIO_OUTPUT_FNAME fnameformat = m_InputCntl->Get_OutputFilenameFormat();
359   fld_fname = m_InputCntl->Get_OutputDir() +"/"+
360               cio_DFI::Generate_FileName(DFI_FInfo->Prefix,
361                                          RankID,
362                                          -1,
363                                          "fld",
364                                          fnameformat,
365                                          mio,
366                                          CIO::E_CIO_OFF);
367
368   if( (fp = fopen(fld_fname.c_str(),"w")) == NULL ) {
369     printf("\tCan't open file.(%s)\n",fld_fname.c_str());
370     Exit(0);
371   }
372
373   //先頭レコードの出力
374   fprintf(fp,"# AVS field file\n");
375
376   //計算空間の次元数を出力
377   fprintf(fp,"ndim=%d\n",ndim);
378
379   //計算空間サイズを出力
380   fprintf(fp,"dim1=%d\n",dims[0]+1);
381   fprintf(fp,"dim2=%d\n",dims[1]+1);
382   fprintf(fp,"dim3=%d\n",dims[2]+1);
383
384   //物理空間の次元数を出力
385   fprintf(fp,"nspace=%d\n",nspace);
386
387   //成分数の出力
388   fprintf(fp,"veclen=%d\n",DFI_FInfo->Component);
389
390   //データのタイプ出力
391   fprintf(fp,"data=%s\n",dType.c_str());
```

```
392
393   //座標定義情報の出力
394   fprintf(fp,"field=uniform\n");
395
396   //label の出力
397   for(int j=0; j<DFI_FInfo->Component; j++) {
398     std::string label=dfi->getComponentVariable(j);
399     if( label == "" ) continue;
400     fprintf(fp,"label=%s\n",label.c_str());
401   }
402
403   //step 毎の出力
404   if( TSlice->SliceList.size()>1 ) {
405     fprintf(fp,"nstep=%d\n",TSlice->SliceList.size());
406   }
407   for(int j=0; j<TSlice->SliceList.size(); j++ ) {
408     fprintf(fp,"time value=%.6f\n",TSlice->SliceList[j].time);
409     for(int n=1; n<=DFI_FInfo->Component; n++) {
410       int skip;
411       /*
412       if( dType == "float" ) {
413         skip=96+(n-1)*4;
414       } else {
415         skip=140+(n-1)*8;
416       }
417       */
418       skip=0;
419       out_fname = cio_DFI::Generate_FileName(DFI_FInfo->Prefix,
420                                              RankID,
421                                              TSlice->SliceList[j].step,
422                                              "sph",
423                                              fnameformat,
424                                              mio,
425                                              CIO::E_CIO_OFF);
426
427       fprintf(fp,"variable %d file=%s filetype=binary skip=%d stride=%d\n",
428               n,out_fname.c_str(),skip,DFI_FInfo->Component);
429     }
430     //coord data
431     cod_fname = cio_DFI::Generate_FileName("cord",
432                                            RankID,
433                                            -1,
434                                            "cod",
435                                            fnameformat,
436                                            mio,
437                                            CIO::E_CIO_OFF);
438
439     fprintf(fp,"coord 1 file=%s filetype=ascii skip=1\n",cod_fname.c_str());
440     fprintf(fp,"coord 2 file=%s filetype=ascii skip=4\n",cod_fname.c_str());
441     fprintf(fp,"coord 3 file=%s filetype=ascii skip=7\n",cod_fname.c_str());
442     fprintf(fp,"EOT\n");
443   }
444
445
446   //出力ヘッダーファイルクローズ
447   fclose(fp);
448
449 }
```

4.6.3.4 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（Mx1)

引数

in	*myRank*	rankID
in	*in_dfi*	dfi のポインター avs ファイルのヘッダー処理（MxM)
in	*myRank*	rankID
in	*in_dfi*	dfi のポインター

convOutput_AVS.C の 164 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().

```
166 {
167   if( myRank != 0 ) return; //myRank==0 のときのみヘッダーレコードを出力
168
169   //間引き数のセット
170   int thin_count = m_InputCntl->Get_ThinOut();
```

```
171
172   int ndim,nspace;
173
174   int dims[3];
175   double min_ext[3],max_ext[3];
176
177   double pit[3];
178
179   for(int i=0; i<in_dfi.size(); i++) {
180     //cio_Domain クラスポインタの取得
181     const cio_Domain* DFI_Domain  = in_dfi[i]->GetcioDomain();
182     //ピッチのセット
183     pit[0]=(DFI_Domain->GlobalRegion[0]/(double)DFI_Domain->GlobalVoxel[0])*(double)thin_count;
184     pit[1]=(DFI_Domain->GlobalRegion[1]/(double)DFI_Domain->GlobalVoxel[1])*(double)thin_count;
185     pit[2]=(DFI_Domain->GlobalRegion[2]/(double)DFI_Domain->GlobalVoxel[2])*(double)thin_count;
186
187     //cio_Process クラスポインタの取得
188     const cio_Process* DFI_Process = in_dfi[i]->GetcioProcess();
189
190     for(int j=0; j<DFI_Process->RankList.size(); j++) {
191       //間引きを考慮しての計算空間サイズをセット
192       dims[0]=DFI_Process->RankList[j].VoxelSize[0]/thin_count;
193       dims[1]=DFI_Process->RankList[j].VoxelSize[1]/thin_count;
194       dims[2]=DFI_Process->RankList[j].VoxelSize[2]/thin_count;
195       if(DFI_Process->RankList[j].VoxelSize[0]%thin_count != 0) dims[0]++;
196       if(DFI_Process->RankList[j].VoxelSize[1]%thin_count != 0) dims[1]++;
197       if(DFI_Process->RankList[j].VoxelSize[2]%thin_count != 0) dims[2]++;
198
199       if( i==0 ) {
200         //座標値の最小値、最大値をセット
201         for(int k=0; k<3; k++) {
202           int head = (DFI_Process->RankList[j].HeadIndex[k]-1)/thin_count;
203           if( (DFI_Process->RankList[j].HeadIndex[k]-1)%thin_count != 0 ) head++;
204           min_ext[k]=DFI_Domain->GlobalOrigin[k]+0.5*pit[k]+double(head)*pit[k];
205           max_ext[k]=min_ext[k]+(double(dims[k]-1))*pit[k];
206         }
207
208         //coord データファイル出力
209         output_avs_coord(j, true, min_ext, max_ext);
210       }
211
212       //avs のヘッダーファイル出力
213       ndim=3;
214       nspace=3;
215       output_avs_header(in_dfi[i], j, true, ndim, nspace, dims);
216
217     }
218   }
219
220   return;
221
222 }
```

4.6.3.5 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 226 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().

```
230 {
231   int ndim,nspace;
232   int dims[3];
233   double min_ext[3],max_ext[3];
234
235   //間引き数のセット
236   int thin_count = m_InputCntl->Get_ThinOut();
237
238   //自ノードでのボクセルサイズ取得
```

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

4.6.3.6 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   CIO::E_CIO_OUTPUT_FNAME fnameformat = m_InputCntl->Get_OutputFilenameFormat();
48   outfile = m_InputCntl->Get_OutputDir()+"/"+
49             cio_DFI::Generate_FileName(prefix,
50                                        id,
51                                        step,
52                                        "dat",
53                                        fnameformat,
54                                        mio,
55                                        CIO::E_CIO_OFF);
56
57   //ファイルオープン
58   if( (fp = fopen(outfile.c_str(), "wb")) == NULL ) {
59     printf("\tCan't open file.(%s)\n",outfile.c_str());
60     Exit(0);
61   }
62
63   return fp;
```

```
64 }
```

4.6.3.7 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 68 行で定義されています。

参照先 Exit.

```
69 {
70
71   const int* sz = src->getArraySizeInt();
72
73   cio_Array *out = cio_Array::instanceArray
74                    (src->getDataType(),
75                     src->getArrayShape(),
76                     (int *)sz,
77                     0,
78                     src->getNcomp());
79   int ret = src->copyArray(out);
80   if( out->writeBinary(fp) != dLen ) {
81     delete out;
82     Exit(0);
83   }
84
85   delete out;
86   return true;
87 }
```

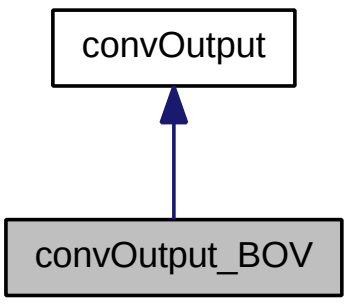
このクラスの説明は次のファイルから生成されました:

- convOutput_AVS.h
- convOutput_AVS.C

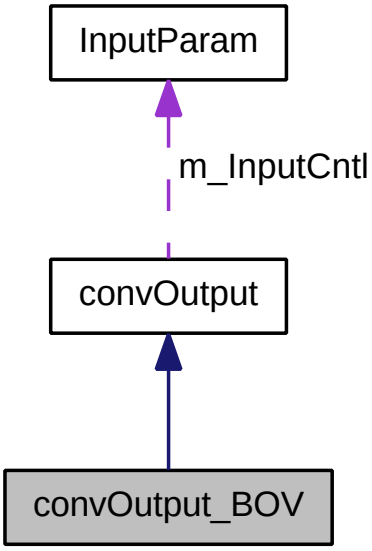
4.7 クラス convOutput_BOV

`#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
- bool WriteHeaderRecord (int step, int dim, CIO::E_CIO_DTYPE d_type, int imax, int jmax, int kmax, double time, double *org, double *pit, const std::string prefix, FILE *fp)
  bov **ファイルの** *header* **の書き込み**

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

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

bov 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を再定義しています。

convOutput_BOV.C の 70 行で定義されています。

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

```
81 {
82
83   FILE* fp;
84
85   //HeadFile 名の生成とオープン
86   std::string head_file;
87   CIO::E_CIO_OUTPUT_FNAME fnameformat = m_InputCntl->Get_OutputFilenameFormat();
88   head_file = m_InputCntl->Get_OutputDir()+"/"+
89             cio_DFI::Generate_FileName(prefix,
90                                        0,
91                                        step,
92                                        "bov",
93                                        fnameformat,
94                                        false,
95                                        CIO::E_CIO_OFF);
96
97   if( (fp = fopen(head_file.c_str(), "wb")) == NULL ) {
98     printf("\tCan't open file.(%s)\n",head_file.c_str());
99     Exit(0);
100   }
101
102   //データファイル名の生成
103   std::string data_file;
104   data_file = cio_DFI::Generate_FileName(prefix,
105                                          0,
106                                          step,
107                                          "dat",
108                                          fnameformat,
109                                          false,
110                                          CIO::E_CIO_OFF);
111
112   fprintf( fp, "Time: %e\n",time);
113   fprintf( fp, "DATA_FILE: %s\n",data_file.c_str());
114   fprintf( fp, "DATA_SIZE: %d %d %d\n",imax,jmax,kmax);
115   std::string dataType;
116   if      ( d_type == CIO::E_CIO_INT8    ) dataType = D_CIO_BYTE;
117   else if( d_type == CIO::E_CIO_UINT8   ) dataType = D_CIO_UINT8;
118   else if( d_type == CIO::E_CIO_INT16   ) dataType = D_CIO_INT16;
119   else if( d_type == CIO::E_CIO_UINT16  ) dataType = D_CIO_UINT16;
120   else if( d_type == CIO::E_CIO_INT32   ) dataType = D_CIO_INT;
121   else if( d_type == CIO::E_CIO_UINT32  ) dataType = D_CIO_UINT32;
122   else if( d_type == CIO::E_CIO_INT64   ) dataType = D_CIO_INT64;
123   else if( d_type == CIO::E_CIO_UINT64  ) dataType = D_CIO_UINT64;
124   else if( d_type == CIO::E_CIO_FLOAT32 ) dataType = D_CIO_FLOAT;
125   else if( d_type == CIO::E_CIO_FLOAT64 ) dataType = D_CIO_DOUBLE;
126   fprintf( fp, "DATA_FORMAT: %s\n",dataType.c_str());
127   fprintf( fp, "DATA_COMPONENTS: %d\n",dim);
128   fprintf( fp, "VARIABLE: %s\n",prefix.c_str());
129
130   std::string endian;
131   int idumy = 1;
132   char* cdumy = (char*)(&idumy);
133   if      ( cdumy[0] == 0x01 ) endian = "LITTLE";
134   else if( cdumy[0] == 0x00 ) endian = "BIG";
135   fprintf( fp, "DATA_ENDIAN: %s\n",endian.c_str());
136   fprintf( fp, "CENTERING: zonal\n");
137   //fprintf( fp, "BRICK_ORIGIN: %e %e %e\n",org[0]-0.5*pit[0],org[1]-0.5*pit[1],org[2]-0.5*pit[2]);
138   fprintf( fp, "BRICK_ORIGIN: %e %e %e\n",org[0],org[1],org[2] );
139   fprintf( fp, "BRICK_SIZE: %e %e %e\n",pit[0]*imax,pit[1]*jmax,pit[2]*kmax);
140
141   std::string arrayShape;
142   if( m_InputCntl->Get_OutputArrayShape() == CIO::E_CIO_IJKN ) arrayShape="IJKN";
143   else arrayShape="NIJK";
144   fprintf( fp, "#CIO_ARRAY_SHAPE: %s\n",arrayShape.c_str());
145
146   fclose(fp);
147   return true;
148 }
```

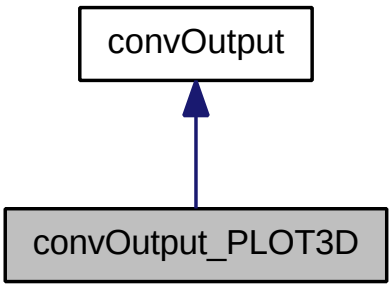
このクラスの説明は次のファイルから生成されました:

- convOutput_BOV.h
- convOutput_BOV.C

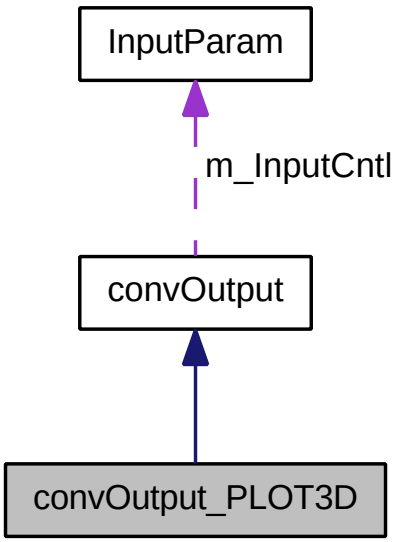
4.8 クラス convOutput_PLOT3D

`#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 (std::string prefix, int step, int myRank, int dType, 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 (std::string prefix, 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, CIO::E_CIO_DTYPE 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**
- 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 (std::string prefix, 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**

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 126 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.

```
131 {
132   FILE* fp;
133
134   //ファイル名の生成
135   std::string outfile;
136   CIO::E_CIO_OUTPUT_FNAME fnameformat = m_InputCntl->Get_OutputFilenameFormat();
137   outfile = m_InputCntl->Get_OutputDir() +"/"+
138             cio_DFI::Generate_FileName(prefix,
139                                        id,
140                                        step,
141                                        "func",
142                                        fnameformat,
143                                        mio,
144                                        CIO::E_CIO_OFF);
145
146   //出力ファイルオープン
147   // ascii
148   if( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
149     if( (fp = fopen(outfile.c_str(), "wa")) == NULL ) {
150       printf("\tCan't open file.(%s)\n",outfile.c_str());
151       Exit(0);
152     }
153   } else {
154   //binary
155     if( (fp = fopen(outfile.c_str(), "wb")) == NULL ) {
156       printf("\tCan't open file.(%s)\n",outfile.c_str());
157       Exit(0);
158     }
159   }
160
161   return fp;
162
163 }
```

4.8.3.2 template<class T1 , class T2 > CONV_INLINE void convOutput_PLOT3D::OutputPlot3D_xyz (std::string *prefix,* 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	*prefix*	ファイル接頭文字
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 43 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().

```
53 {
54   //value
55   int ngrid=1;
56   int ix = size[0];
57   int jx = size[1];
58   int kx = size[2];
59   int gd = guide;
60   int gc_out = 0;//plot3d は常にガイドセルを出力しない
61
62   //間引き数の取得
63   int thin_count = m_InputCntl->Get_ThinOut();
64
65   int *iblank=NULL;//dummy
66   int id,jd,kd;//出力サイズ
67   id=size[0]+1;//+2*gc_out
68   jd=size[1]+1;//+2*gc_out
69   kd=size[2]+1;//+2*gc_out
70
71   //間引きのための処理
72   int irest=(id-1)%thin_count;
73   int jrest=(jd-1)%thin_count;
74   int krest=(kd-1)%thin_count;
75   id=(id-1)/thin_count;
76   jd=(jd-1)/thin_count;
77   kd=(kd-1)/thin_count;
78   id=id+1;
79   jd=jd+1;
80   kd=kd+1;
81   if(irest!=0) id=id+1;
82   if(jrest!=0) jd=jd+1;
83   if(krest!=0) kd=kd+1;
84
85   // 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
86   T2 l_org[3], l_pit[3];
87   for (int i=0; i<3; i++)
88   {
89     l_org[i] = (T2)origin[i] + (T2)pitch[i]*(T2)gd;
90     l_pit[i] = (T2)pitch[i];
91   }
92
93   // 出力ファイル名
94   std::string tmp;
95   //std::string t_prefix=prefix+"_Grid";
96   int fnameformat = m_InputCntl->Get_OutputFilenameFormat();
97   tmp = m_InputCntl->Get_OutputDir() +"/"+
98         cio_DFI::Generate_FileName(prefix,
99                                    rank,
100                                    //step,
101                                    -1,
102                                    "xyz",
103                                    (CIO::E_CIO_OUTPUT_FNAME)fnameformat,
104                                    false,
105                                    CIO::E_CIO_OFF);
106
107    //open file
108    FILE*fp;
109    if( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
110      if( (fp = fopen(tmp.c_str(), "wa")) == NULL ) {
111        printf("\tCan't open file.(%s)\n",tmp.c_str());
112        Exit(0);
113      }
```

```
114   } else {
115     if( (fp = fopen(tmp.c_str(), "wb")) == NULL ) {
116       printf("\tCan't open file.(%s)\n",tmp.c_str());
117       Exit(0);
118     }
119   }
120
121   //write block data
122   WriteNgrid(fp,ngrid);//if multi grid
123   WriteBlockData(fp,id,jd,kd);
124
125   for(int k=0;k<kd;k++){
126   for(int j=0;j<jd;j++){
127   for(int i=0;i<id;i++){
128     size_t ip = _F_IDX_S3D(i+1, j+1, k+1, id, jd, kd, 0);
129     x[ip]=l_org[0]+(T2)thin_count*l_pit[0]*(T2)i;//-pitch[0]*(float)gc_out;
130     y[ip]=l_org[1]+(T2)thin_count*l_pit[1]*(T2)j;//-pitch[1]*(float)gc_out;
131     z[ip]=l_org[2]+(T2)thin_count*l_pit[2]*(T2)k;//-pitch[2]*(float)gc_out;
132   }}}
133
134   //x direction modify
135   if(irest!=0 && (id-2)>=0 ){
136     for(int k=0;k<kd;k++){
137     for(int j=0;j<jd;j++){
138       size_t ip = _F_IDX_S3D(id, j+1, k+1, id, jd, kd, 0);
139       x[ip]=l_org[0]+(T2)thin_count*l_pit[0]*(T2)(id-2)+(T2)irest*l_pit[0];//-pitch[0]*(float)gc_out;
140     }}
141   }
142
143   //y direction modify
144   if(jrest!=0 && (jd-2)>=0 ){
145     for(int k=0;k<kd;k++){
146     for(int i=0;i<id;i++){
147       size_t ip = _F_IDX_S3D(i+1, jd, k+1, id, jd, kd, 0);
148       y[ip]=l_org[1]+(T2)thin_count*l_pit[1]*(T2)(jd-2)+(T2)jrest*l_pit[1];//-pitch[1]*(float)gc_out;
149     }}
150   }
151
152   //z direction modify
153   if(krest!=0 && (kd-2)>=0 ){
154     for(int j=0;j<jd;j++){
155     for(int i=0;i<id;i++){
156       size_t ip = _F_IDX_S3D(i+1, j+1, kd, id, jd, kd, 0);
157       z[ip]=l_org[2]+(T2)thin_count*l_pit[2]*(T2)(kd-2)+(T2)krest*l_pit[2];//-pitch[2]*(float)gc_out;
158     }}
159   }
160
161   //z direction modify
162   if(krest!=0){
163     for(int k=kd-1;k<kd;k++){
164     for(int j=0;j<jd;j++){
165     for(int i=0;i<id;i++){
166       size_t ip = _F_IDX_S3D(i+1, j+1, k+1, id, jd, kd, 0);
167       z[ip]=l_org[2]+(T2)krest*l_pit[2]*(T2)k;//-pitch[2]*(float)gc_out;
168     }}}
169   }
170
171   //write
172   if(!WriteXYZData(fp, id, jd, kd, ngrid, x, y, z)) printf("\terror WriteXYZData\n");
173
174   //close file
175   fclose(fp);
176
177 }
```

4.8.3.3 template<class T1 , class T2 > void convOutput_PLOT3D::OutputPlot3D_xyz (std::string *prefix*, 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	*prefix*	ファイル接頭文字
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 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 97 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().

```
98 {
99
100   switch (m_InputCntl->Get_OutputFormatType()) {
101     case CIO::E_CIO_OUTPUT_TYPE_FBINARY:
102       unsigned int dmy;
103       dmy = sizeof(int)*3;
104       WriteDataMarker(dmy,fp,true);
105       fwrite(&id, sizeof(int), 1, fp);
106       fwrite(&jd, sizeof(int), 1, fp);
107       fwrite(&kd, sizeof(int), 1, fp);
108       WriteDataMarker(dmy,fp,true);
109       break;
110     case CIO::E_CIO_OUTPUT_TYPE_ASCII:
111       fprintf(fp,"%5d%5d%5d\n",id,jd,kd);
112       break;
113     case CIO::E_CIO_OUTPUT_TYPE_BINARY:
114       fwrite(&id, sizeof(int), 1, fp);
115       fwrite(&jd, sizeof(int), 1, fp);
116       fwrite(&kd, sizeof(int), 1, fp);
117       break;
118    default:
119       break;
120    }
121
122 }
```

4.8.3.5 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 293 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().

```
294 {
295   if( !out ) return true;
296   if( m_InputCntl->Get_OutputFormatType() != CIO::E_CIO_OUTPUT_TYPE_FBINARY ) return true;
297   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
298   return true;
299 }
```

4.8.3.6 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 200 行で定義されています。

参照先 WriteFuncData().

```
203 {
204
205   const int* sz = src->getArraySizeInt();
206
207   cio_Array *out = cio_Array::instanceArray
208                    (src->getDataType(),
209                     src->getArrayShape(),
210                     (int *)sz,
211                     0,
212                     src->getNcomp());
213
214   int ret = src->copyArray(out);
215
216   WriteFuncData(fp, out);
217   delete out;
218
219   return true;
220
221 }
```

4.8.3.7 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 225 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().

```
226 {
227
228   switch (m_InputCntl->Get_OutputFormatType()) {
229     case CIO::E_CIO_OUTPUT_TYPE_FBINARY:
230       unsigned int dmy;
231       dmy = sizeof(int)*4;
232       WriteDataMarker(dmy,fp,true);
233       fwrite(&id, sizeof(int), 1, fp);
234       fwrite(&jd, sizeof(int), 1, fp);
235       fwrite(&kd, sizeof(int), 1, fp);
236       fwrite(&nvar, sizeof(int), 1, fp);
237       WriteDataMarker(dmy,fp,true);
238       break;
239     case CIO::E_CIO_OUTPUT_TYPE_ASCII:
240       fprintf(fp,"%5d%5d%5d%5d\n",id,jd,kd,nvar);
241       break;
242     case CIO::E_CIO_OUTPUT_TYPE_BINARY:
243       fwrite(&id, sizeof(int), 1, fp);
244       fwrite(&jd, sizeof(int), 1, fp);
245       fwrite(&kd, sizeof(int), 1, fp);
246       fwrite(&nvar, sizeof(int), 1, fp);
247       break;
248     default:
249       break;
250   }
251
252 }
```

4.8.3.8 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 255 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().

```
256 {
257
258   const int* sz = p3src->getArraySizeInt();
259   size_t dLen = (size_t)sz[0]*(size_t)sz[1]*(size_t)sz[2]*p3src->getNcomp();
260
261   switch (m_InputCntl->Get_OutputFormatType()) {
262     case CIO::E_CIO_OUTPUT_TYPE_FBINARY:
263       unsigned int dmy;
264       if( p3src->getDataType() == CIO::E_CIO_FLOAT32 ) {
265         dmy = sizeof(float)*dLen;
266       } else {
267         dmy = sizeof(double)*dLen;
268       }
269       WriteDataMarker(dmy,fp,true);
270       p3src->writeBinary(fp);
271       WriteDataMarker(dmy,fp,true);
272       break;
273     case CIO::E_CIO_OUTPUT_TYPE_ASCII:
274       if( p3src->getDataType() == CIO::E_CIO_FLOAT32) {
275         float *data = (float*)p3src->getData();
276         for(int i=0; i<dLen; i++) fprintf(fp,"%15.6E\n",data[i]);
277       } else if( p3src->getDataType() == CIO::E_CIO_FLOAT64) {
278         double *data = (double*)p3src->getData();
279         for(int i=0; i<dLen; i++) fprintf(fp,"%15.6E\n",data[i]);
280       }
281       break;
282     case CIO::E_CIO_OUTPUT_TYPE_BINARY:
```

```
283       p3src->writeBinary(fp);
284       break;
285     default:
286       break;
287   }
288
289 }
```

4.8.3.9 void convOutput_PLOT3D::WriteGridData (std::string *prefix*, int *step*, int *myRank*, int *dType*, int *guide*, double *org[3]*, double *pit[3]*, int *sz[3]*) [virtual]

GRID ファイル出力

引数

in	*prefix*	ファイル接頭文字
in	*step*	step 番号
in	*myRank*	ランク番号
in	*dType*	dfi のデータタイプ
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().

```
46 {
47
48   //step 0 以外は出力しない
49   if( step != 0 ) return;
50
51   int id,jd,kd;
52   id=sz[0]+1;
53   jd=sz[1]+1;
54   kd=sz[2]+1;
55   size_t maxsize = (size_t)id*(size_t)jd*(size_t)kd*3;
56   size_t outsize = (size_t)id*(size_t)jd*(size_t)kd;
57
58   int outDtype = m_InputCntl->Get_OutputDataType();
59   if( outDtype == CIO::E_CIO_DTYPE_UNKNOWN ) outDtype = dType;
60
61   if( outDtype == CIO::E_CIO_FLOAT64 ) {
62     double* x = new double[maxsize];
63     OutputPlot3D_xyz(prefix, step, myRank, guide, org, pit, sz, &x[0],
64                      &x[outsize] , &x[outsize*2] );
65   }else if( outDtype == CIO::E_CIO_FLOAT32 ) {
66     float* x = new float[maxsize];
67     OutputPlot3D_xyz(prefix, step, myRank, guide, org, pit, sz, &x[0],
68                      &x[outsize] , &x[outsize*2] );
69   }
70 }
```

4.8.3.10 bool convOutput_PLOT3D::WriteHeaderRecord (int *step*, int *dim*, CIO::E_CIO_DTYPE *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 167 行で定義されています。

参照先 WriteFuncBlockData(), と WriteNgrid().

```
179 {
180   if( !fp ) return false;
181
182   //ngird の初期化
183   int ngrid=1;
184
185   //ngrid の出力
186   WriteNgrid(fp,ngrid);
187
188   //block data の出力
189
190   WriteFuncBlockData(fp,imax+1,jmax+1,kmax+1,dim);
191
192   return true;
193
194 }
```

4.8.3.11 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 74 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().

```
75 {
76   switch (m_InputCntl->Get_OutputFormatType()) {
77     case CIO::E_CIO_OUTPUT_TYPE_FBINARY:
78       unsigned int dmy;
79       dmy = sizeof(int);
80       WriteDataMarker(dmy,fp,true);
81       fwrite(&ngrid, sizeof(int), 1, fp);
82       WriteDataMarker(dmy,fp,true);
83       break;
84     case CIO::E_CIO_OUTPUT_TYPE_ASCII:
85       fprintf(fp,"%5d\n",ngrid);
86       break;
87     case CIO::E_CIO_OUTPUT_TYPE_BINARY:
88       fwrite(&ngrid, sizeof(int), 1, fp);
89       break;
90     default:
91       break;
92     }
93 }
```

4.8.3.12 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().

4.8.3.13 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 252 行で定義されています。

```
257 {
258
259   int s12 =(size_t)id*(size_t)jd;
260   int ns12=s12/10;
261
262 /*
263   for(int k=0; k<kd; k++) {
264     //x-y 面の出力
265     for(int i=0; i<ns12; i++) {
266       for(int ii=0; ii<10; ii++) {
267         fprintf(fp,"%15.6E",x[(k*id*jd)+(i*10)+ii]);
268       }
269       fprintf(fp,"\n");
270     }
271     //余りの出力
272     if( s12%10 > 0 ) {
273       for(int i=0; i<s12%10; i++) fprintf(fp,"%15.6E",x[k*id*jd+(ns12*10)+i]);
274     }
275     fprintf(fp,"\n");
276   }
277 */
278   for(int i=0; i<id*jd*kd; i++) {
279     fprintf(fp,"%15.6E\n",x[i]);
280   }
281
282 }
```

4.8.3.14 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().

4.8.3.15 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 193 行で定義されています。

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

```
201 {
202
203   size_t sz = (size_t)id*(size_t)jd*(size_t)kd;
204   unsigned int dmy;
205
206   int s12 =(size_t)id*(size_t)jd;
207   int ns12=s12/10;
208
209   switch (m_InputCntl->Get_OutputFormatType()) {
210
211     //Fortran Binary 出力
212     case CIO::E_CIO_OUTPUT_TYPE_FBINARY:
213       dmy = sizeof(T)*sz*3;
214       WriteDataMarker(dmy,fp,true);
215       fwrite(x, sizeof(T), sz, fp);
216       fwrite(y, sizeof(T), sz, fp);
217       fwrite(z, sizeof(T), sz, fp);
218       WriteDataMarker(dmy,fp,true);
219       break;
220
221     //ascii 出力
222     case CIO::E_CIO_OUTPUT_TYPE_ASCII:
223       WriteXYZ_FORMATTED(fp, id, jd, kd, x);
224       WriteXYZ_FORMATTED(fp, id, jd, kd, y);
225       WriteXYZ_FORMATTED(fp, id, jd, kd, z);
226       break;
227
228     //C Binary 出力
229     case CIO::E_CIO_OUTPUT_TYPE_BINARY:
230       fwrite(x, sizeof(T), sz, fp);
231       fwrite(y, sizeof(T), sz, fp);
232       fwrite(z, sizeof(T), sz, fp);
233       break;
234     default:
235       return false;
236       break;
237   }
238   return true;
239 }
```

このクラスの説明は次のファイルから生成されました:

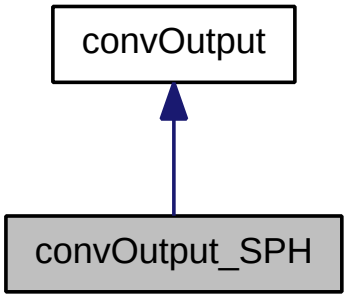
- convOutput_PLOT3D.h

- convOutput_PLOT3D.C
- conv_plot3d_inline.h

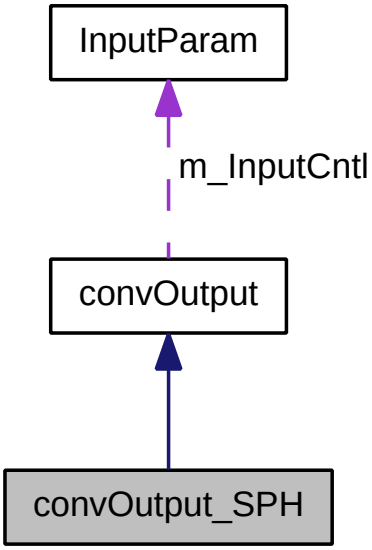
4.9 クラス convOutput_SPH

`#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, CIO::E_CIO_DTYPE 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   CIO::E_CIO_OUTPUT_FNAME fnameformat = m_InputCntl->Get_OutputFilenameFormat();
49   outfile = m_InputCntl->Get_OutputDir()+"/"+
50             cio_DFI::Generate_FileName(prefix,
51                                        id,
52                                        step,
53                                        "sph",
54                                        fnameformat,
55                                        mio,
```

```
56                                          CIO::E_CIO_OFF);
57
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
67 }
```

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 175 行で定義されています。

```
176 {
177   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
178   return true;
179 }
```

4.9.3.3 bool convOutput_SPH::WriteHeaderRecord (int *step*, int *dim*, CIO::E_CIO_DTYPE *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 71 行で定義されています。

参照先 SPH_DOUBLE, SPH_FLOAT, SPH_SCALAR, と SPH_VECTOR.

```
83 {
84   if( !fp ) return false;
85
86   unsigned int dmy;
87
88   //出力データ種別フラグの設定
89   int sv_type;
90   if( dim == 1 ) sv_type = SPH_SCALAR;
91   else if( dim == 3 ) sv_type = SPH_VECTOR;
```

```
92
93   //出力データ型フラグの設定
94   int d_type;
95   if( out_type == CIO::E_CIO_FLOAT32 ) d_type = SPH_FLOAT;
96   else if( out_type == CIO::E_CIO_FLOAT64 ) d_type = SPH_DOUBLE;
97
98   dmy = 2 * sizeof(int);
99   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
100   if( fwrite(&sv_type, sizeof(int), 1, fp) != 1 ) return false;
101   if( fwrite(&d_type, sizeof(int), 1, fp) != 1 ) return false;
102   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
103
104   if( d_type == SPH_FLOAT ) {
105     dmy = 3 * sizeof(int);
106   } else if( d_type == SPH_DOUBLE ) {
107     dmy = 3 * sizeof(long long);
108   }
109
110   //dmy = 3 * sizeof(long long);
111   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
112   if( d_type == SPH_FLOAT ) {
113     if( fwrite(&imax, sizeof(int), 1, fp) != 1 ) return false;
114     if( fwrite(&jmax, sizeof(int), 1, fp) != 1 ) return false;
115     if( fwrite(&kmax, sizeof(int), 1, fp) != 1 ) return false;
116   } else {
117     if( fwrite(&imax, sizeof(long long), 1, fp) != 1 ) return false;
118     if( fwrite(&jmax, sizeof(long long), 1, fp) != 1 ) return false;
119     if( fwrite(&kmax, sizeof(long long), 1, fp) != 1 ) return false;
120   }
121   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
122
123   if( d_type == SPH_FLOAT ) {
124     dmy = 3 * sizeof(float);
125   } else {
126     dmy = 3 * sizeof(double);
127   }
128   //dmy = 3 * sizeof(double);
129   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
130   if( d_type == SPH_FLOAT ) {
131     float tmp[3];
132     tmp[0] = (float)org[0];
133     tmp[1] = (float)org[1];
134     tmp[2] = (float)org[2];
135     if( fwrite(tmp, sizeof(float), 3, fp) != 3 ) return false;
136   } else {
137     if( fwrite(org, sizeof(double), 3, fp) != 3 ) return false;
138   }
139   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
140
141   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
142   if( d_type == SPH_FLOAT ) {
143     float tmp[3];
144     tmp[0] = (float)pit[0];
145     tmp[1] = (float)pit[1];
146     tmp[2] = (float)pit[2];
147     if( fwrite(tmp, sizeof(float), 3, fp) != 3 ) return false;
148   } else {
149     if( fwrite(pit, sizeof(double), 3, fp) != 3 ) return false;
150   }
151   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
152
153   if( d_type == SPH_FLOAT ) {
154     dmy = sizeof(int) + sizeof(float);
155   } else {
156     dmy = sizeof(long long) + sizeof(double);
157   }
158   //dmy = sizeof(long long) + sizeof(double);
159   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
160   if( d_type == SPH_FLOAT ) {
161     float ftmp = (float)time;
162     if( fwrite(&step, sizeof(int), 1, fp) != 1 ) return false;
163     if( fwrite(&ftmp, sizeof(float), 1, fp) != 1 ) return false;
164   } else {
165     if( fwrite(&step, sizeof(long long), 1, fp) != 1 ) return false;
166     if( fwrite(&time, sizeof(double), 1, fp) != 1 ) return false;
167   }
168   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
169
170   return true;
171 }
```

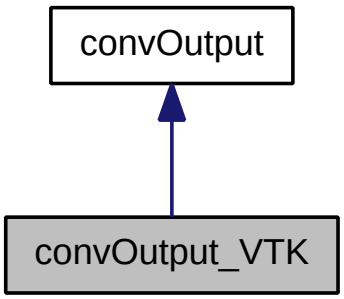
このクラスの説明は次のファイルから生成されました:

- convOutput_SPH.h
- convOutput_SPH.C

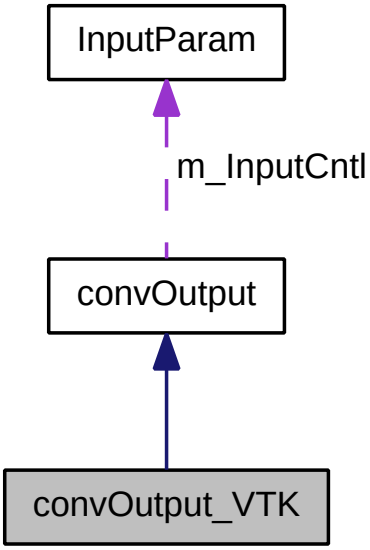
4.10 クラス convOutput_VTK

`#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, CIO::E_CIO_DTYPE 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 259 行で定義されています。

```
260 {
261   fprintf( fp, "\n" );
262   fclose(fp);
263 }
```

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   CIO::E_CIO_OUTPUT_FNAME fnameformat = m_InputCntl->Get_OutputFilenameFormat();
49   outfile = m_InputCntl->Get_OutputDir()+ "/"+
50             cio_DFI::Generate_FileName(prefix,
51                                        id,
52                                        step,
53                                        "vtk",
54                                        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 250 行で定義されています。

```
251 {
252   if( !out ) return true;
253   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return false;
254   return true;
255 }
```

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     //出力タイプごとにポインターを取得して出力
167     if( out->getDataType() == CIO::E_CIO_UINT8 ) {
168       //UINT8
169       unsigned char *data = (unsigned char*)out->getData();
170       BSWAPVEC(data,dLen);
171       fwrite( data, sizeof(unsigned char), dLen, fp);
172
173     } else if( out->getDataType() == CIO::E_CIO_INT8 ) {
174       //INT8
175       char *data = (char*)out->getData();
176       BSWAPVEC(data,dLen);
177       fwrite( data, sizeof(char), dLen, fp);
178
179     } else if( out->getDataType() == CIO::E_CIO_UINT16 ) {
180       //UINT16
181       unsigned short *data = (unsigned short*)out->getData();
182       BSWAPVEC(data,dLen);
183       fwrite( data, sizeof(unsigned short), dLen, fp);
184
185     } else if( out->getDataType() == CIO::E_CIO_INT16 ) {
186       //INT16
187       short *data = (short*)out->getData();
188       BSWAPVEC(data,dLen);
189       fwrite( data, sizeof(short), dLen, fp);
190
191     } else if( out->getDataType() == CIO::E_CIO_UINT32 ) {
192       //UINT32
193       unsigned int *data = (unsigned int*)out->getData();
194       BSWAPVEC(data,dLen);
195       fwrite( data, sizeof(unsigned int), dLen, fp);
196
197     } else if( out->getDataType() == CIO::E_CIO_INT32 ) {
198       //INT32
199       int *data = (int*)out->getData();
200       BSWAPVEC(data,dLen);
201       fwrite( data, sizeof(int), dLen, fp);
202
203     } else if( out->getDataType() == CIO::E_CIO_UINT64 ) {
204       //UINT64
205       unsigned long long *data = (unsigned long long*)out->getData();
206       BSWAPVEC(data,dLen);
207       fwrite( data, sizeof(unsigned long long), dLen, fp);
208
209     } else if( out->getDataType() == CIO::E_CIO_INT64 ) {
210       //INT64
211       long long *data = (long long*)out->getData();
212       BSWAPVEC(data,dLen);
213       fwrite( data, sizeof(long long), dLen, fp);
214
215     } else if( out->getDataType() == CIO::E_CIO_FLOAT32 ) {
216       //FLOAT32
217       float *data = (float*)out->getData();
218       BSWAPVEC(data,dLen);
219       fwrite( data, sizeof(float), dLen, fp );
220
221     } else if( out->getDataType() == CIO::E_CIO_FLOAT64 ) {
222       //FLOAT 64
```

```
223       double *data = (double*)out->getData();
224       DBSWAPVEC(data,dLen);
225       fwrite( data, sizeof(double), dLen, fp );
226
227     } else {
228       printf("\tIllegal datatype\n");
229       delete out;
230       Exit(0);
231     }
232
233   //アスキー出力のとき
234   } else if ( m_InputCntl->Get_OutputFormatType() == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
235
236     if( out->writeAscii(fp) != dLen ) {
237       delete out;
238       Exit(0);
239     }
240
241   }
242
243   delete out;
244   return true;
245 }
```

4.10.3.5 bool convOutput_VTK::WriteHeaderRecord (int *step*, int *dim*, CIO::E_CIO_DTYPE *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   std::string out_d_type;
119   if(      d_type == CIO::E_CIO_UINT8  ) out_d_type="unsigned_char";
120   else if( d_type == CIO::E_CIO_INT8   ) out_d_type="char";
121   else if( d_type == CIO::E_CIO_UINT16 ) out_d_type="unsigned_short";
122   else if( d_type == CIO::E_CIO_INT16  ) out_d_type="short";
123   else if( d_type == CIO::E_CIO_UINT32 ) out_d_type="unsigned_int";
124   else if( d_type == CIO::E_CIO_INT32  ) out_d_type="int";
125   else if( d_type == CIO::E_CIO_UINT64 ) out_d_type="unsigned_long";
126   else if( d_type == CIO::E_CIO_INT64  ) out_d_type="long";
127   else if( d_type == CIO::E_CIO_FLOAT32) out_d_type="float";
128   else if( d_type == CIO::E_CIO_FLOAT64) out_d_type="double";
129
130   if( dim == 1 )
131   {
132     fprintf( fp, "SCALARS %s %s\n", prefix.c_str(), out_d_type.c_str() );
133     fprintf( fp, "LOOKUP_TABLE default\n" );
134   }
135   else if( dim == 3 )
136   {
137     fprintf( fp, "VECTORS %s %s\n", prefix.c_str(), out_d_type.c_str() );
138   }
139   else
140   {
141     fprintf( fp, "FIELD %s 1\n", prefix.c_str() );
142     fprintf( fp, "%s %d %d %s\n", prefix.c_str(), dim, nw, out_d_type.c_str() );
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 InputParam::dfi_info

`#include <InputParam.h>`

Public 変数

- std::string in_dfi_name
  読込み *dfi* ファイル名
- cio_DFI * in_dfi
  読込み *dfi* ポインタ
- std::string out_dfi_name
  出力 *dfi* ファイル名
- std::string out_proc_name
  出力 *proc* ファイル名

4.11.1 説明

InputParam.h の 49 行で定義されています。

4.11.2 変数

4.11.2.1 cio_DFI* InputParam::dfi_info::in_dfi

読込み dfi ポインタ

InputParam.h の 51 行で定義されています。
参照元 InputParam::Read().

4.11.2.2 std::string InputParam::dfi_info::in_dfi_name

読込み dfi ファイル名
InputParam.h の 50 行で定義されています。
参照元 InputParam::Read().

4.11.2.3 std::string InputParam::dfi_info::out_dfi_name

出力 dfi ファイル名
InputParam.h の 52 行で定義されています。
参照元 InputParam::Read().

4.11.2.4 std::string InputParam::dfi_info::out_proc_name

出力 proc ファイル名
InputParam.h の 53 行で定義されています。
参照元 InputParam::Read().
この構造体の説明は次のファイルから生成されました:

- InputParam.h

4.12 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

4.12.1 説明

conv.h の 61 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 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     };
```

4.12.2.2 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     };
```

4.12.3 変数

4.12.3.1 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().

4.12.3.2 double* CONV::dfi_MinMax::Max

conv.h の 64 行で定義されています。

4.12.3.3 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

4.13 クラス InputParam

`#include <InputParam.h>`

構成

- struct dfi_info

Public メソッド

- InputParam (cpm_ParaManager ∗paraMngr)
- ∼InputParam ()
- bool Read (std::string input_file_name)
  CPM のポインタをコピー
- bool InputParamCheck ()
  入力パラメータのエラーチェック
- void PrintParam (FILE ∗fp)
  入力パラメータのログ出力
- vector< dfi_info > Get_dfiList ()
  dfiList の取り出し
- bool Get_Outputdfi_on ()
  dfi 出力指示フラグの取り出し
- E_CONV_OUTPUT_CONV_TYPE Get_ConvType ()
  input dfi ファイル名リストの取り出し
- int ∗ Get_OutputDivision ()
  出力分割数の取り出し
- CIO::E_CIO_FORMAT Get_OutputFormat ()
  出力ファイルフォーマットの取り出し
- std::string Get_OutputFormat_string ()
- CIO::E_CIO_DTYPE Get_OutputDataType ()
  出力タイプの取り出し
- std::string Get_OutputDataType_string ()
  出力タイプの取り出し *(*文字列*)*
- CIO::E_CIO_OUTPUT_TYPE 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
- CIO::E_CIO_ARRAYSHAPE Get_OutputArrayShape ()
  出力配列形状の取り出し
- void Set_OutputArrayShape (CIO::E_CIO_ARRAYSHAPE outputArrayShape)
  出力配列形状のセット
- CIO::E_CIO_OUTPUT_FNAME 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
- void Set_OutputGuideCell (int outgc)
  出力ガイドセル数の更新
- E_CONV_OUTPUT_MULTI_FILE_CAST Get_MultiFileCasting ()
  並列処理時のファイル割振り方法の取り出し
- bool Get_CropIndexStart_on ()
  入力領域のスタート指示フラグの取り出し
- bool Get_CropIndexEnd_on ()
  入力領域のエンド指示フラグの取り出し
- int ∗ Get_CropIndexStart ()
  入力領域のスタート位置取り出し
- int ∗ Get_CropIndexEnd ()
  入力領域のエンド位置取り出し

- void Set_CropIndexStart (int sta[3])
  入力領域のスタート位置の更新
- void Set_CropIndexEnd (int end[3])
  入力領域のエンド位置の更新

Public 変数

- vector< dfi_info > m_dfiList
  入出力する *DFI* **名等が格納**
- cpm_ParaManager * m_paraMngr

Protected 変数

- bool m_output_dfi_on
  dfi **出力指示フラグ**
- bool m_cropIndexStart_on
  入力領域のスタート指示フラグ
- bool m_cropIndexEnd_on
  入力領域のエンド指示フラグ
- 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)**
- int m_outputDiv [3]
  出力分割数 *MxN* **で有効**
- int m_thin_count
  間引き数
- int m_outputGuideCell
  出力するガイドセル数
- std::string m_outdir_name
  出力先ディレクトリー名
- CIO::E_CIO_DTYPE m_output_data_type
  出力実数タイプ *byte,short,int,float,double*
- CIO::E_CIO_FORMAT m_out_format
  出力ファイルフォーマット *sph,bov,avs,plot3d,vtk*
- CIO::E_CIO_OUTPUT_TYPE m_out_format_type
  出力形式 *ascii,binary,FortranBinary*
- E_CONV_OUTPUT_CONV_TYPE m_conv_type
  convert **タイプ** *Mx1 MxN MxM*
- CIO::E_CIO_ARRAYSHAPE m_outputArrayShape
  出力配列形状
- CIO::E_CIO_OUTPUT_FNAME m_outputFilenameFormat
  出力ファイル名命名順
- E_CONV_OUTPUT_MULTI_FILE_CAST m_multiFileCasting
  並列処理時のファイル割振り方法

4.13.1 説明

InputParam.h の 44 行で定義されています。

4.13.2 コンストラクタとデストラクタ

4.13.2.1 InputParam::InputParam (cpm_ParaManager * *paraMngr*)

コンストラクタ

InputParam.C の 21 行で定義されています。

参照先 E_CONV_OUTPUT_UNKNOWN, m_conv_type, m_cropIndexEnd_on, m_cropIndexStart_on, m_dfiList, m_out_format_type, m_output_data_type, m_output_dfi_on, m_outputArrayShape, m_outputDiv, m_outputFilenameFormat, m_outputGuideCell, m_paraMngr, と m_thin_count.

```
22 {
23
24   //CPM のポインターをセット
25   m_paraMngr=paraMngr;
26
27   m_thin_count=1;
28   m_out_format_type=CIO::E_CIO_OUTPUT_TYPE_BINARY;
29   m_outputDiv[0]=-1;m_outputDiv[1]=-1;m_outputDiv[2]=-1;
30   m_outputArrayShape=CIO::E_CIO_ARRAYSHAPE_UNKNOWN;
31   m_outputFilenameFormat=CIO::E_CIO_FNAME_STEP_RANK;
32   m_conv_type=E_CONV_OUTPUT_UNKNOWN;
33   m_outputGuideCell=0;
34   m_output_data_type = CIO::E_CIO_DTYPE_UNKNOWN;
35   //m_in_dfi_name.clear();
36   //m_out_dfi_name.clear();
37   //m_out_proc_name.clear();
38   m_dfiList.clear();
39   m_output_dfi_on = false;
40
41   m_cropIndexStart_on=false;
42   m_cropIndexEnd_on  =false;
43
44 }
```

4.13.2.2 InputParam::~InputParam ( )

デストラクタ

InputParam.C の 48 行で定義されています。

```
49 {
50
51 }
```

4.13.3 関数

4.13.3.1 E_CONV_OUTPUT_CONV_TYPE InputParam::Get_ConvType ( ) [inline]

input dfi ファイル名リストの取り出し

output dfi ファイル名リストの取り出し output proc ファイル名リストの取り出し 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 コンバートタイプの取り出し

InputParam.h の 155 行で定義されています。

参照元 CONV::CheckDFIdata(), と CONV::ConvInit().

```
155 { return m_conv_type; };
```

4.13.3.2 int* InputParam::Get_CropIndexEnd ( ) [inline]

入力領域のエンド位置取り出し

InputParam.h の 274 行で定義されています。

参照元 CONV::CheckDFIdata(), convMx1::convMx1_out_ijkn(), convMx1::convMx1_out_nijk(), convMx1::exec(), convMxN::exec(), CONV::makeProcInfo(), と convMxN::VoxelInit().

```
274 { return m_cropIndexEnd; };
```

4.13.3.3 bool InputParam::Get_CropIndexEnd_on ( ) [inline]

入力領域のエンド指示フラグの取り出し

InputParam.h の 264 行で定義されています。

参照元 CONV::CheckDFIdata(), convMx1::convMx1_out_ijkn(), convMx1::convMx1_out_nijk(), convMx1::exec(), convMxN::exec(), CONV::makeProcInfo(), と convMxN::VoxelInit().

```
264 { return m_cropIndexEnd_on; }
```

4.13.3.4 int* InputParam::Get_CropIndexStart ( ) [inline]

入力領域のスタート位置取り出し

InputParam.h の 269 行で定義されています。

参照元 CONV::CheckDFIdata(), convMx1::convMx1_out_ijkn(), convMx1::convMx1_out_nijk(), CONV::copyArray(), convMx1::exec(), convMxN::exec(), CONV::makeProcInfo(), と convMxN::VoxelInit().

```
269 { return m_cropIndexStart; };
```

4.13.3.5 bool InputParam::Get_CropIndexStart_on ( ) [inline]

入力領域のスタート指示フラグの取り出し

InputParam.h の 259 行で定義されています。

参照元 CONV::CheckDFIdata(), convMx1::convMx1_out_ijkn(), convMx1::convMx1_out_nijk(), CONV::copyArray(), convMx1::exec(), convMxN::exec(), CONV::makeProcInfo(), と convMxN::VoxelInit().

```
259 { return m_cropIndexStart_on; }
```

4.13.3.6 vector<dfi_info> InputParam::Get_dfiList ( ) [inline]

dfiList の取り出し

InputParam.h の 117 行で定義されています。

```
117 { return m_dfiList; };
```

4.13.3.7 E_CONV_OUTPUT_MULTI_FILE_CAST 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 254 行で定義されています。

参照元 convMxM::exec().

```
254 { return m_multiFileCasting; };
```

4.13.3.8 CIO::E_CIO_ARRAYSHAPE InputParam::Get_OutputArrayShape ( ) [inline]

出力配列形状の取り出し

InputParam.h の 225 行で定義されています。

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

```
225 { return m_outputArrayShape; };
```

4.13.3.9 CIO::E_CIO_DTYPE InputParam::Get_OutputDataType ( ) [inline]

出力タイプの取り出し

InputParam.h の 185 行で定義されています。

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

```
185 { return m_output_data_type; };
```

4.13.3.10 std::string InputParam::Get_OutputDataType_string ( ) [inline]

出力タイプの取り出し (文字列)

InputParam.h の 190 行で定義されています。

参照元 InputParamCheck(), と convOutput_AVS::output_avs_header().

```
191   {
192     if( Get_OutputDataType() == CIO::E_CIO_INT8    ) return "Int8";
193     if( Get_OutputDataType() == CIO::E_CIO_UINT8   ) return "UInt8";
194     if( Get_OutputDataType() == CIO::E_CIO_INT16   ) return "Int16";
195     if( Get_OutputDataType() == CIO::E_CIO_UINT16  ) return "UInt16";
196     if( Get_OutputDataType() == CIO::E_CIO_INT32   ) return "Int32";
197     if( Get_OutputDataType() == CIO::E_CIO_UINT32  ) return "UInt32";
198     if( Get_OutputDataType() == CIO::E_CIO_INT64   ) return "Int64";
199     if( Get_OutputDataType() == CIO::E_CIO_UINT64  ) return "UInt64";
200     if( Get_OutputDataType() == CIO::E_CIO_FLOAT32 ) return "Float32";
201     if( Get_OutputDataType() == CIO::E_CIO_FLOAT64 ) return "Float64";
202     return "";
203   };
```

4.13.3.11 bool InputParam::Get_Outputdfi_on ( ) [inline]

dfi 出力指示フラグの取り出し

InputParam.h の 122 行で定義されています。

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

```
122 { return m_output_dfi_on; };
```

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

出力先ディレクトリ名の取り出し

InputParam.h の 214 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(), convOutput_BOV::WriteHeaderRecord(), と CONV::WriteIndexDfiFile().

```
214 { return m_outdir_name; };
```

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

出力分割数の取り出し

InputParam.h の 160 行で定義されています。

参照元 convMxN::VoxelInit().

```
160 { return m_outputDiv; };
```

4.13.3.14 CIO::E_CIO_OUTPUT_FNAME InputParam::Get_OutputFilenameFormat ( ) [inline]

出力ファイル名命名順の取り出し

InputParam.h の 237 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(), convOutput_PLOT3D::OutputPlot3D_xyz(), convOutput_BOV::WriteHeaderRecord(), と CONV::WriteIndexDfiFile().

```
237 { return m_outputFilenameFormat; };
```

4.13.3.15 CIO::E_CIO_FORMAT InputParam::Get_OutputFormat ( ) [inline]

出力ファイルフォーマットの取り出し

InputParam.h の 165 行で定義されています。

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

```
165 { return m_out_format; };
```

4.13.3.16 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 170 行で定義されています。

参照元 InputParamCheck().

```
171   {
172     if( Get_OutputFormat() == CIO::E_CIO_FMT_SPH ) return "sph";
173     if( Get_OutputFormat() == CIO::E_CIO_FMT_BOV ) return "bov";
174     if( Get_OutputFormat() == CIO::E_CIO_FMT_AVS ) return "avs";
175     if( Get_OutputFormat() == CIO::E_CIO_FMT_VTK ) return "vtk";
176     if( Get_OutputFormat() == CIO::E_CIO_FMT_PLOT3D ) return "plot3d";
177     return "";
178   };
```

4.13.3.17 CIO::E_CIO_OUTPUT_TYPE InputParam::Get_OutputFormatType ( ) [inline]

出力形式の取り出し（ascii,binary,FortranBinary)

InputParam.h の 209 行で定義されています。

参照元 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().

```
209 { return m_out_format_type; };
```

4.13.3.18 int InputParam::Get_OutputGuideCell ( ) [inline]

出力ガイドセル数の取り出し

InputParam.h の 242 行で定義されています。

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

```
242 { return m_outputGuideCell; };
```

4.13.3.19 int InputParam::Get_ThinOut ( ) [inline]

間引き数の取り出し

InputParam.h の 219 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(), convOutput_AVS::output_avs_MxM(), convOutput_AVS::output_avs_MxN(), convOutput_PLOT3D::OutputPlot3D_xyz(), と convMxN::VoxelInit().

```
219 { return m_thin_count; };
```

4.13.3.20 bool InputParam::InputParamCheck ( )

入力パラメータのエラーチェック

戻り値

error code

InputParam.C の 429 行で定義されています。

参照先 E_CONV_OUTPUT_Mx1, E_CONV_OUTPUT_MxM, E_CONV_OUTPUT_RANK, E_CONV_OUTPUT_-UNKNOWN, Get_OutputDataType_string(), Get_OutputFormat_string(), m_conv_type, m_cropIndexEnd_on, m_-cropIndexStart_on, m_dfiList, m_multiFileCasting, m_out_format, m_out_format_type, m_outdir_name, m_output-_data_type, m_output_dfi_on, m_outputArrayShape, m_outputGuideCell, と m_thin_count.

参照元 Read().

```
430 {
431
432   bool ierr=true;
433
434   //読込む DFI ファイル名指示のチェック
435   if( m_dfiList.size() < 1 ) {
436     printf("\tundefined input dfi file name\n");
437     ierr=false;
438   }
439
440   //コンバートタイプのチェック
441   if( m_conv_type == E_CONV_OUTPUT_UNKNOWN ) {
442     printf("\tundefine Converter Type\n");
443     ierr=false;
444   }
445
446   //出力先ディレクトリのチェック
447   if( m_outdir_name.empty() ) {
448     printf("\tundefine OutputDir\n");
449     ierr=false;
450   }
451
452   //出力形式のチェック
453   if( m_out_format_type == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
454     if( m_out_format != CIO::E_CIO_FMT_PLOT3D &&
455         m_out_format != CIO::E_CIO_FMT_VTK ) {
456       printf("\tCan't Converter OutputFormatType ascii.\n");
457       ierr=false;
458     }
459   }
460
461   //出力配列形状のチェック
462   //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
463   //なので、指定があった場合はメッセージを出力する
464   if( m_outputArrayShape != CIO::E_CIO_ARRAYSHAPE_UNKNOWN ) {
465     if( m_out_format != CIO::E_CIO_FMT_BOV ) printf("\tCan't OutputArrayShape.\n");
466   }
467
468   //出力配列形状のセット
469   if      ( m_out_format == CIO::E_CIO_FMT_SPH ) m_outputArrayShape = CIO::E_CIO_NIJK;
470   else if( m_out_format == CIO::E_CIO_FMT_AVS ) m_outputArrayShape = CIO::E_CIO_NIJK;
471   else if( m_out_format == CIO::E_CIO_FMT_VTK ) m_outputArrayShape = CIO::E_CIO_NIJK;
472   else if( m_out_format == CIO::E_CIO_FMT_PLOT3D ) m_outputArrayShape = CIO::E_CIO_IJKN;
473   else if( m_out_format == CIO::E_CIO_FMT_BOV &&
474            m_outputArrayShape == CIO::E_CIO_ARRAYSHAPE_UNKNOWN )
475        m_outputArrayShape = CIO::E_CIO_NIJK;
476
477   //未対応のデータ型への変換チェック
478   if( m_output_data_type != CIO::E_CIO_DTYPE_UNKNOWN ) {
479     //(sph,plot3d)
480     if( m_out_format == CIO::E_CIO_FMT_SPH || m_out_format == CIO::E_CIO_FMT_PLOT3D ) {
481       if( m_output_data_type != CIO::E_CIO_FLOAT32 &&
482           m_output_data_type != CIO::E_CIO_FLOAT64 ) {
483         printf("\tCan't Converter OutputDataType %s.\n",
484                Get_OutputDataType_string().c_str());
485         ierr=false;
486       }
487     }
488
489     //(avs)
490     if( m_out_format == CIO::E_CIO_FMT_AVS ) {
491       if( m_output_data_type != CIO::E_CIO_INT8    &&
492           m_output_data_type != CIO::E_CIO_INT16   &&
493           m_output_data_type != CIO::E_CIO_INT32   &&
494           m_output_data_type != CIO::E_CIO_FLOAT32 &&
495           m_output_data_type != CIO::E_CIO_FLOAT64 ) {
496         printf("\tCan't Converter OutputDataType %s.\n",
497                Get_OutputDataType_string().c_str());
498         ierr=false;
499       }
```

```
500     }
501   }
502
503   //DFI 出力のチェック、出力する DFI ファイル名のチェック
504   if( m_output_dfi_on ) {
505     //DFI 出力が SPH, BOV 以外で指定された場合はエラー
506     if( m_out_format != CIO::E_CIO_FMT_SPH && m_out_format != CIO::E_CIO_FMT_BOV ) {
507       printf("\tCan't output dfi OutputFormat. %s\n",Get_OutputFormat_string().c_str());
508       ierr=false;
509     }
510     //出力 DFI 名のチェック、パスが指定されている場合はエラー
511     for(int i=0; i<m_dfiList.size(); i++) {
512       std::string inPath = CIO::cioPath_DirName(m_dfiList[i].out_dfi_name);
513       if( inPath != "" && inPath !="./" ) {
514         printf("\tIllegal OutputDFI : %s\n",m_dfiList[i].out_dfi_name.c_str());
515         ierr=false;
516       }
517       if( m_dfiList[i].out_proc_name != "" ) {
518         inPath = CIO::cioPath_DirName(m_dfiList[i].out_proc_name);
519         if( inPath != "" && inPath !="./" ) {
520           printf("\tIllegal OutputProc : %s\n",m_dfiList[i].out_proc_name.c_str());
521           ierr=false;
522         }
523       } else {
524       //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
525         std::string proc = CIO::ExtractPathWithoutExt(m_dfiList[i].out_dfi_name);
526         std::string fname = proc+"_proc.dfi";
527         m_dfiList[i].out_proc_name = fname;
528       }
529     }
530   }
531
532   //出力ガイドセル数のチェック
533   if( m_outputGuideCell > 0 ) {
534     //sph,bov 以外は出力指定不可
535     if( m_out_format != CIO::E_CIO_FMT_SPH && m_out_format != CIO::E_CIO_FMT_BOV ) {
536       printf("\tCan't output guide cell : %s\n",Get_OutputFormat_string().c_str());
537       ierr=false;
538     }
539     //間引きありとガイドセル出力を両方指定は不可
540     if( m_thin_count > 1 ) {
541       printf("\tCan't output guide cell and thinning out\n");
542       ierr=false;
543     }
544   }
545
546   //ファイル割振り方法のチェック
547   if( m_multiFileCasting == E_CONV_OUTPUT_RANK && m_conv_type ==
      E_CONV_OUTPUT_Mx1 ) {
548     printf("\tCan't multi file casting type rank\n");
549   }
550
551   //入力領域指示のチェック
552   if( m_conv_type == E_CONV_OUTPUT_MxM ) {
553     if( m_cropIndexStart_on || m_cropIndexEnd_on ) {
554       printf("\tCan't define CropIndex option exec type MxM\n");
555       ierr=false;
556     }
557   }
558
559   return ierr;
560
561 }
```

4.13.3.21 void InputParam::PrintParam (FILE * *fp*)

入力パラメータのログ出力

引数

in	*fp*	出力ファイルポインタ

InputParam.C の 565 行で定義されています。

参照先 E_CONV_OUTPUT_Mx1, E_CONV_OUTPUT_MxM, E_CONV_OUTPUT_MxN, E_CONV_OUTPUT_RANK, E_CONV_OUTPUT_STEP, Exit, m_conv_type, m_cropIndexEnd, m_cropIndexEnd_on, m_cropIndexStart, m_cropIndexStart_on, m_dfiList, m_multiFileCasting, m_out_format, m_out_format_type, m_outdir_name, m_output_data_type, m_output_dfi_on, m_outputArrayShape, m_outputDiv, m_outputFilenameFormat, m_outputGuideCell, と m_thin_count.

参照元 CONV::ReadDfiFiles().

```
566 {
567   fprintf(fp,"\n");
568   fprintf(fp,"*** fconv file info ***\n");
569   fprintf(fp,"\n");
570 
571   for(int i=0; i<m_dfiList.size(); i++) {
572     fprintf(fp,"\tInputDFI[@]          : \"%s\"\n",m_dfiList[i].in_dfi_name.c_str());
573   }
574 
575   if( m_output_dfi_on ) {
576     for(int i=0; i<m_dfiList.size(); i++) {
577       fprintf(fp,"\tOutputDFI[@]         : \"%s\"\n",m_dfiList[i].out_dfi_name.c_str());
578     }
579     for(int i=0; i<m_dfiList.size(); i++) {
580       fprintf(fp,"\tOutputProcDFI[@]     : \"%s\"\n",m_dfiList[i].out_proc_name.c_str());
581     }
582   }
583 
584   if( m_conv_type == E_CONV_OUTPUT_Mx1 ) {
585     fprintf(fp,"\tConvType             : \"Mx1\"\n");
586   } else if( m_conv_type == E_CONV_OUTPUT_MxM ) {
587     fprintf(fp,"\tConvType             : \"MxM\"\n");
588   } else if( m_conv_type == E_CONV_OUTPUT_MxN ) {
589     fprintf(fp,"\tConvType             : \"MxN\"\n");
590   } else {
591     Exit(0);
592   }
593 
594   fprintf(fp,"\tOutputDivision       : (%d,%d,%d)\n",m_outputDiv[0],m_outputDiv[1],
    m_outputDiv[2]);
595 
596   if( m_out_format == CIO::E_CIO_FMT_SPH ) {
597     fprintf(fp,"\tOutputFormat         : \"sph\"\n");
598   }else if( m_out_format == CIO::E_CIO_FMT_BOV ) {
599     fprintf(fp,"\tOutputFormat         : \"bov\"\n");
600   }else if( m_out_format == CIO::E_CIO_FMT_AVS ) {
601     fprintf(fp,"\tOutputFormat         : \"avs\"\n");
602   }else if( m_out_format == CIO::E_CIO_FMT_PLOT3D ) {
603     fprintf(fp,"\tOutputFormat         : \"plot3d\"\n");
604   }else if( m_out_format == CIO::E_CIO_FMT_VTK ) {
605     fprintf(fp,"\tOutputFormat         : \"vtk\"\n");
606   }else {
607     Exit(0);
608   }
609 
610   if( m_output_data_type == CIO::E_CIO_INT8 ) {
611     fprintf(fp,"\tOutputDataType       : \"Int8\"\n");
612   }else if( m_output_data_type == CIO::E_CIO_INT16 ) {
613     fprintf(fp,"\tOutputDataType       : \"Int16\"\n");
614   }else if( m_output_data_type == CIO::E_CIO_INT32 ) {
615     fprintf(fp,"\tOutputDataType       : \"Int32\"\n");
616   }else if( m_output_data_type == CIO::E_CIO_INT64 ) {
617     fprintf(fp,"\tOutputDataType       : \"Int64\"\n");
618   }else if( m_output_data_type == CIO::E_CIO_UINT8 ) {
619     fprintf(fp,"\tOutputDataType       : \"UInt8\"\n");
620   }else if( m_output_data_type == CIO::E_CIO_UINT16 ) {
621     fprintf(fp,"\tOutputDataType       : \"UInt16\"\n");
622   }else if( m_output_data_type == CIO::E_CIO_UINT32 ) {
623     fprintf(fp,"\tOutputDataType       : \"UInt32\"\n");
624   }else if( m_output_data_type == CIO::E_CIO_UINT64 ) {
625     fprintf(fp,"\tOutputDataType       : \"UInt64\"\n");
626   }else if( m_output_data_type == CIO::E_CIO_FLOAT32 ) {
627     fprintf(fp,"\tOutputDataType       : \"Float32\"\n");
628   }else if( m_output_data_type == CIO::E_CIO_FLOAT64 ) {
629     fprintf(fp,"\tOutputDataType       : \"Float64\"\n");
630   } else {
631     fprintf(fp,"\tOutputDataType       : \"undefine\"\n");
632   }
633 
634   if( m_out_format_type == CIO::E_CIO_OUTPUT_TYPE_DEFAULT ) {
635     fprintf(fp,"\tOutputForamtType     : \"undefine\"\n");
636   } else if( m_out_format_type == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
637     fprintf(fp,"\tOutputForamtType     : \"ascii\"\n");
638   } else if( m_out_format_type == CIO::E_CIO_OUTPUT_TYPE_BINARY ) {
639     fprintf(fp,"\tOutputForamtType     : \"binary\"\n");
640   } else if( m_out_format_type == CIO::E_CIO_OUTPUT_TYPE_FBINARY ) {
641     fprintf(fp,"\tOutputForamtType     : \"Fortran Binary\"\n");
642   }
643 
644   fprintf(fp,"\tOutputdir            : \"%s\"\n",m_outdir_name.c_str());
645   fprintf(fp,"\tThinning             : %d\n",m_thin_count);
646 
647   if( m_outputArrayShape == CIO::E_CIO_IJKN ) {
648     fprintf(fp,"\tOutputArrayShape     : \"ijkn\"\n");
649   }else if( m_outputArrayShape == CIO::E_CIO_NIJK ) {
```

```
650       fprintf(fp,"\tOutputArrayShape     : \"nijk\"\n");
651     }else {
652       fprintf(fp,"\tOutputArrayShape     : \"undefine\"\n");
653     }
654
655     if( m_outputFilenameFormat == CIO::E_CIO_FNAME_STEP_RANK ) {
656       fprintf(fp,"\tOutputFilenameFormat : \"step_rank\"\n");
657     }else if( m_outputFilenameFormat == CIO::E_CIO_FNAME_RANK_STEP ) {
658       fprintf(fp,"\tOutputFilenameFormat : \"rank_step\"\n");
659     }else {
660       fprintf(fp,"\tOutputFilenameFormat : \"undefine\"\n");
661     }
662
663     fprintf(fp,"\tOutputGuideCell      : %d\n",m_outputGuideCell);
664
665     if( m_multiFileCasting == E_CONV_OUTPUT_STEP ) {
666       fprintf(fp,"\tMultiFileCasting     : \"step\"\n");
667     }else if( m_multiFileCasting == E_CONV_OUTPUT_RANK ) {
668       fprintf(fp,"\tMultiFileCasting     : \"rank\"\n");
669     }else {
670       fprintf(fp,"\tMultiFileCasting     : \"undefine\"\n");
671     }
672
673     if( m_cropIndexStart_on ) {
674       fprintf(fp,"\tCropIndexStart       : (%d,%d,%d)\n",m_cropIndexStart[0],
675                                                          m_cropIndexStart[1],
676                                                          m_cropIndexStart[2]);
677     }
678     if( m_cropIndexEnd_on ) {
679       fprintf(fp,"\tCropIndexEnd         : (%d,%d,%d)\n",m_cropIndexEnd[0],
680                                                          m_cropIndexEnd[1],
681                                                          m_cropIndexEnd[2]);
682     }
683
684     fprintf(fp,"\n");
685 }
```

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

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 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 66 行で定義されています。

参照先 E_CONV_OUTPUT_Mx1, E_CONV_OUTPUT_MxM, E_CONV_OUTPUT_MxN, E_CONV_OUTPUT_RANK, E_CONV_OUTPUT_STEP, Exit, Hostonly_, InputParam::dfi_info::in_dfi, InputParam::dfi_info::in_dfi_name, InputParamCheck(), m_conv_type, m_cropIndexEnd, m_cropIndexEnd_on, m_cropIndexStart, m_cropIndexStart_on, m_dfiList, m_multiFileCasting, m_out_format, m_out_format_type, m_outdir_name, m_output_data_type, m_output_dfi_on, m_outputArrayShape, m_outputDiv, m_outputFilenameFormat, m_outputGuideCell, m_thin_count, InputParam::dfi_info::out_dfi_name, InputParam::dfi_info::out_proc_name, と stamped_printf.

参照元 main().

```
67 {
68   FILE* fp=NULL;
69   string str;
70   string label,label_base,label_leaf;
71
72   // TextParser のインスタンス
73   TextParser tpCntl;
74
75   // 入力ファイルをセット
76   int err = tpCntl.read(input_file_name);
```

```
77
78   // node 数の取得
79   int nnode=0;
80   label_base = "/ConvData";
81   if( tpCntl.chkNode(label_base) ) {
82     nnode = tpCntl.countLabels(label_base);
83   } else Exit(0);
84
85   vector<std::string> in_dfi_name;
86   vector<std::string> out_dfi_name;
87   vector<std::string> out_proc_name;
88   in_dfi_name.clear();
89   out_dfi_name.clear();
90   out_proc_name.clear();
91
92   int ncnt=0;
93   label_base = "/ConvData";
94   ncnt++;
95   for(int i=0; i<nnode; i++) {
96     if( !tpCntl.getNodeStr(label_base, ncnt, str) ) {
97       printf("\tParsing error : No Elem name\n");
98       Exit(0);
99     }
100    // 読込み dfi ファイル名の読込み
101    if( !strcasecmp(str.substr(0,8).c_str(), "InputDFI") ) {
102      label = label_base+"/"+str;
103      if ( !(tpCntl.getInspectedValue(label, str )) ) {
104        printf("\tParsing error : fail to get '%s'\n", label.c_str());
105        Exit(0);
106      }
107      //dfi ファイル名を格納
108      in_dfi_name.push_back(str);
109      ncnt++;
110      continue;
111    } else
112
113    //出力 dfi ファイル名の読込み
114    if( !strcasecmp(str.substr(0,9).c_str(), "OutputDFI") ) {
115      label = label_base+"/"+str;
116      if ( !(tpCntl.getInspectedValue(label, str )) ) {
117        printf("\tParsing error : fail to get '%s'\n", label.c_str());
118        Exit(0);
119      }
120      //dfi ファイル名を格納
121      out_dfi_name.push_back(str);
122      ncnt++;
123      continue;
124    } else
125
126    //出力 proc ファイル名の読込み
127    if( !strcasecmp(str.substr(0,13).c_str(), "OutputProcDFI") ) {
128      label = label_base+"/"+str;
129      if ( !(tpCntl.getInspectedValue(label, str )) ) {
130        printf("\tParsing error : fail to get '%s'\n", label.c_str());
131        Exit(0);
132      }
133      //dfi ファイル名を格納
134      out_proc_name.push_back(str);
135      ncnt++;
136      continue;
137    } else
138
139    // コンバートタイプの読込み
140    if( !strcasecmp(str.c_str(),"ConvType") ) {
141      label = "/ConvData/ConvType";
142      if( (tpCntl.getInspectedValue(label,str)) ) {
143        if     ( !strcasecmp(str.c_str(), "Mx1") )  m_conv_type =
     E_CONV_OUTPUT_Mx1;
144        else if( !strcasecmp(str.c_str(), "MxN") )  m_conv_type =
     E_CONV_OUTPUT_MxN;
145        else if( !strcasecmp(str.c_str(), "MxM") )  m_conv_type =
     E_CONV_OUTPUT_MxM;
146        else
147        {
148          Hostonly_ stamped_printf("\tInvalid keyword is described for '%s'\n", label.c_str());
149          Exit(0);
150        }
151      }
152      ncnt++;
153      continue;
154    } else
155
156    // 出力分割数の読込み
157    if( !strcasecmp(str.c_str(),"OutputDivision") ) {
158      int vec[3];
159      label = "/ConvData/OutputDivision";
160      if( (tpCntl.getInspectedVector(label, vec, 3)) ) {
```

```
161         if( vec[0]<1 || vec[1]<1 || vec[2]<1 ) {
162            Hostonly_ stamped_printf("\tInvalid keyword is described for '%s'\n", label.c_str());
163            Exit(0);
164         }
165         m_outputDiv[0]=vec[0];
166         m_outputDiv[1]=vec[1];
167         m_outputDiv[2]=vec[2];
168       }
169       ncnt++;
170       continue;
171     } else
172
173     // 出力ファイルフォーマットの読込み
174     if( !strcasecmp(str.c_str(),"OutputFormat") ) {
175       label = "/ConvData/OutputFormat";
176       if ( !(tpCntl.getInspectedValue(label, str )) )
177       {
178          Hostonly_ stamped_printf("\tParsing error : fail to get '%s'\n", label.c_str());
179          Exit(0);
180       }
181       if     ( !strcasecmp(str.c_str(), "sph" ) )    m_out_format = CIO::E_CIO_FMT_SPH;
182       else if( !strcasecmp(str.c_str(), "bov" ) )    m_out_format = CIO::E_CIO_FMT_BOV;
183       else if( !strcasecmp(str.c_str(), "avs" ) )    m_out_format = CIO::E_CIO_FMT_AVS;
184       else if( !strcasecmp(str.c_str(), "plot3d" ) ) m_out_format = CIO::E_CIO_FMT_PLOT3D;
185       else if( !strcasecmp(str.c_str(), "vtk" ) )    m_out_format = CIO::E_CIO_FMT_VTK;
186       else
187       {
188         Hostonly_ stamped_printf("\tInvalid keyword is described for  '%s'\n", label.c_str());
189         Exit(0);
190       }
191       ncnt++;
192       continue;
193     } else
194
195      // 出力データタイプの読込み
196     if( !strcasecmp(str.c_str(),"OutputDataType") ) {
197       label = "/ConvData/OutputDataType";
198       if( !(tpCntl.getInspectedValue(label, str )) ) {
199         m_output_data_type = CIO::E_CIO_DTYPE_UNKNOWN;
200       } else {
201         if     ( !strcasecmp(str.c_str(), "UInt8"  ) ) m_output_data_type = CIO::E_CIO_UINT8;
202         else if( !strcasecmp(str.c_str(),  "Int8"  ) ) m_output_data_type = CIO::E_CIO_INT8;
203         else if( !strcasecmp(str.c_str(), "UInt16" ) ) m_output_data_type = CIO::E_CIO_UINT16;
204         else if( !strcasecmp(str.c_str(),  "Int16" ) ) m_output_data_type = CIO::E_CIO_INT16;
205         else if( !strcasecmp(str.c_str(), "UInt32" ) ) m_output_data_type = CIO::E_CIO_UINT32;
206         else if( !strcasecmp(str.c_str(),  "Int32" ) ) m_output_data_type = CIO::E_CIO_INT32;
207         else if( !strcasecmp(str.c_str(), "UInt64" ) ) m_output_data_type = CIO::E_CIO_UINT64;
208         else if( !strcasecmp(str.c_str(),  "Int64" ) ) m_output_data_type = CIO::E_CIO_INT64;
209         else if( !strcasecmp(str.c_str(), "Float32") ) m_output_data_type = CIO::E_CIO_FLOAT32;
210         else if( !strcasecmp(str.c_str(), "Float64") ) m_output_data_type = CIO::E_CIO_FLOAT64;
211         else
212         {
213           printf("\tInvalid keyword is described for '%s'\n", label.c_str());
214           Exit(0);
215         }
216       }
217       ncnt++;
218       continue;
219     }
220
221     // 出力形式の読込み
222     if( !strcasecmp(str.c_str(),"OutputFormatType") ) {
223       label = "/ConvData/OutputFormatType";
224       if( !(tpCntl.getInspectedValue(label, str )) ) {
225         m_out_format_type = CIO::E_CIO_OUTPUT_TYPE_BINARY;
226       } else {
227         if     ( !strcasecmp(str.c_str(), "binary") )        m_out_format_type=
    CIO::E_CIO_OUTPUT_TYPE_BINARY;
228         else if( !strcasecmp(str.c_str(), "ascii") )         m_out_format_type=CIO::E_CIO_OUTPUT_TYPE_ASCII
    ;
229         else if( !strcasecmp(str.c_str(), "FortranBinary") ) m_out_format_type=
    CIO::E_CIO_OUTPUT_TYPE_FBINARY;
230         else
231         {
232            printf("\tInvalid keyword is described for '%s'\n", label.c_str());
233            Exit(0);
234         }
235       }
236       ncnt++;
237       continue;
238     } else
239
240     // 出力先ディレクトリの読込み
241     if( !strcasecmp(str.c_str(),"OutputDir") ) {
242       label = "/ConvData/OutputDir";
243       if( !(tpCntl.getInspectedValue(label, str )) ) {
244         Hostonly_ stamped_printf("\tParsing error : fail to get '%s'\n", label.c_str());
```

```
245         Exit(0);
246       } else {
247         m_outdir_name = str;
248       }
249       ncnt++;
250       continue;
251     } else
252
253     //間引き数の読込み
254     if( !strcasecmp(str.c_str(),"ThinningOut") ) {
255       int ict;
256       label = "/ConvData/ThinningOut";
257       if( !(tpCntl.getInspectedValue(label, ict )) ) {
258          m_thin_count=1;
259       } else {
260         if( ict < 0 ) {
261            printf("\tInvalid keyword is described for '%s'\n", label.c_str());
262            Exit(0);
263         }
264         m_thin_count = ict;
265       }
266       ncnt++;
267       continue;
268     } else
269
270     //出力配列形状の読込み
271     if( !strcasecmp(str.c_str(),"OutputArrayShape") ) {
272       label = "/ConvData/OutputArrayShape";
273       if( !(tpCntl.getInspectedValue(label, str )) ) {
274         m_outputArrayShape = CIO::E_CIO_ARRAYSHAPE_UNKNOWN;
275       } else {
276         if      ( !strcasecmp(str.c_str(), "ijkn") ) m_outputArrayShape = CIO::E_CIO_IJKN;
277         else if( !strcasecmp(str.c_str(), "nijk") ) m_outputArrayShape = CIO::E_CIO_NIJK;
278         else {
279           printf("\tInvalid keyword is described for '%s'\n", label.c_str());
280           Exit(0);
281         }
282       }
283       ncnt++;
284       continue;
285     } else
286
287     //出力ファイル名命名順の読込み
288     if( !strcasecmp(str.c_str(),"OutputFilenameFormat") ) {
289       label = "/ConvData/OutputFilenameFormat";
290       if( !(tpCntl.getInspectedValue(label, str )) ) {
291         m_outputFilenameFormat = CIO::E_CIO_FNAME_STEP_RANK;
292       } else {
293         if      ( !strcasecmp(str.c_str(), "step_rank") ) m_outputFilenameFormat =
    CIO::E_CIO_FNAME_STEP_RANK;
294         else if( !strcasecmp(str.c_str(), "rank_step") ) m_outputFilenameFormat =
    CIO::E_CIO_FNAME_RANK_STEP;
295         else {
296           printf("\tInvalid keyword is described for '%s'\n", label.c_str());
297           Exit(0);
298         }
299       }
300       ncnt++;
301       continue;
302     } else
303
304     //出力ガイドセル数
305     if( !strcasecmp(str.c_str(),"OutputGuideCell") ) {
306       int ict;
307       label = "/ConvData/OutputGuideCell";
308       if( !(tpCntl.getInspectedValue(label, ict )) ) {
309         m_outputGuideCell=0;
310       } else {
311         if( ict < 0 ) {
312           printf("\tInvalid keyword is described for '%s'\n", label.c_str());
313           Exit(0);
314         }
315         m_outputGuideCell=ict;
316       }
317       ncnt++;
318       continue;
319     } else
320
321     //並列処理時のファイル割振り方法
322     if( !strcasecmp(str.c_str(),"MultiFileCasting") ) {
323       label = "/ConvData/MultiFileCasting";
324       if( !(tpCntl.getInspectedValue(label, str )) ) {
325         m_multiFileCasting = E_CONV_OUTPUT_STEP;
326       } else {
327         if      ( !strcasecmp(str.c_str(), "step") ) m_multiFileCasting =
    E_CONV_OUTPUT_STEP;
328         else if( !strcasecmp(str.c_str(), "rank") ) m_multiFileCasting =
```

```
    E_CONV_OUTPUT_RANK;
329         else {
330           printf("\tInvalid keyword is described for '%s'\n", label.c_str());
331           Exit(0);
332         }
333       }
334       ncnt++;
335       continue;
336     } else
337
338     //入力領域のスタート位置)
339     if( !strcasecmp(str.c_str(),"CropIndexStart") ) {
340       int vec[3];
341       label = "/ConvData/CropIndexStart";
342       if( (tpCntl.getInspectedVector(label, vec, 3)) ) {
343         m_cropIndexStart[0]=vec[0];
344         m_cropIndexStart[1]=vec[1];
345         m_cropIndexStart[2]=vec[2];
346         m_cropIndexStart_on=true;
347       } else {
348         printf("\tInvalid keyword is described for '%s'\n", label.c_str());
349         Exit(0);
350       }
351       ncnt++;
352       continue;
353     } else
354
355     //入力領域のエンド位置
356     if( !strcasecmp(str.c_str(),"CropIndexEnd") ) {
357       int vec[3];
358       label = "/ConvData/CropIndexEnd";
359       if( (tpCntl.getInspectedVector(label, vec, 3)) ) {
360         m_cropIndexEnd[0]=vec[0];
361         m_cropIndexEnd[1]=vec[1];
362         m_cropIndexEnd[2]=vec[2];
363         m_cropIndexEnd_on=true;
364       } else {
365         printf("\tInvalid keyword is described for '%s'\n", label.c_str());
366         Exit(0);
367       }
368       ncnt++;
369       continue;
370     } else {
371       printf("\tParsing error : No Elem name : %s\n",str.c_str());
372       Exit(0);
373     }
374   }
375
376   // TextParser の破棄
377   tpCntl.remove();
378
379   //入力 dfi ファイル名の登録
380   if( in_dfi_name.size() > 0 ) {
381     dfi_info dInfo;
382     for(int i=0; i<in_dfi_name.size(); i++){
383       dInfo.in_dfi_name  = in_dfi_name[i];
384       dInfo.in_dfi       = NULL;
385       dInfo.out_dfi_name = "";
386       dInfo.out_proc_name= "";
387       m_dfiList.push_back(dInfo);
388     }
389     in_dfi_name.clear();
390   }
391
392   //出力 dfi ファイル名の登録
393   if( out_dfi_name.size() > 0 ) {
394     if( out_dfi_name.size() != m_dfiList.size() ) {
395       printf("\tmismatch outout dfi file names\n");
396       out_dfi_name.clear();
397       return false;
398     }
399     for(int i=0; i<m_dfiList.size(); i++) {
400        m_dfiList[i].out_dfi_name = out_dfi_name[i];
401     }
402     out_dfi_name.clear();
403     m_output_dfi_on = true;
404   } else m_output_dfi_on = false;
405
406
407   //出力 proc ファイル名の登録
408   if( out_proc_name.size() > 0 ) {
409     if( out_proc_name.size() != m_dfiList.size() ) {
410       printf("\tmismatch outout proc file names\n");
411       out_proc_name.clear();
412       return false;
413     }
414     for(int i=0; i<m_dfiList.size(); i++) {
```

```
415         m_dfiList[i].out_proc_name = out_proc_name[i];
416       }
417       out_proc_name.clear();
418     }
419
420     //入力パラメータのチェック
421     if( !InputParamCheck() ) return false;
422
423     return true;
424
425 }
```

4.13.3.23 void InputParam::Set_CropIndexEnd (int *end[3]*) [inline]

入力領域のエンド位置の更新

引数

in	*end*	

InputParam.h の 289 行で定義されています。

参照元 CONV::CheckDFIdata().

```
290     {
291       for(int i=0; i<3; i++) m_cropIndexEnd[i]=end[i];
292     };
```

4.13.3.24 void InputParam::Set_CropIndexStart (int *sta[3]*) [inline]

入力領域のスタート位置の更新

引数

in	*sta*	

InputParam.h の 280 行で定義されています。

参照元 CONV::CheckDFIdata().

```
281     {
282       for(int i=0; i<3; i++) m_cropIndexStart[i]=sta[i];
283     };
```

4.13.3.25 void InputParam::Set_OutputArrayShape (CIO::E_CIO_ARRAYSHAPE *outputArrayShape*) [inline]

出力配列形状のセット

InputParam.h の 230 行で定義されています。

```
231         { m_outputArrayShape = outputArrayShape; };
```

4.13.3.26 void InputParam::Set_OutputGuideCell (int *outgc*) [inline]

出力ガイドセル数の更新

引数

in	*outgc*	出力ガイドセル数

InputParam.h の 248 行で定義されています。

参照元 CONV::CheckDFIdata().

```
248 { m_outputGuideCell = outgc; };
```

4.13.4 変数

4.13.4.1 E_CONV_OUTPUT_CONV_TYPE InputParam::m_conv_type [protected]

convert タイプ Mx1 MxN MxM

InputParam.h の 75 行で定義されています。

参照元 InputParam(), InputParamCheck(), PrintParam(), と Read().

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

入力領域のエンド位置（2014 対応予定)

InputParam.h の 66 行で定義されています。

参照元 PrintParam(), と Read().

4.13.4.3 bool InputParam::m_cropIndexEnd_on [protected]

入力領域のエンド指示フラグ

InputParam.h の 64 行で定義されています。

参照元 InputParam(), InputParamCheck(), PrintParam(), と Read().

4.13.4.4 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 65 行で定義されています。

参照元 PrintParam(), と Read().

4.13.4.5 bool InputParam::m_cropIndexStart_on [protected]

入力領域のスタート指示フラグ

InputParam.h の 63 行で定義されています。

参照元 InputParam(), InputParamCheck(), PrintParam(), と Read().

4.13.4.6 vector<dfi_info> InputParam::m_dfiList

入出力するDFI 名等が格納

InputParam.h の 56 行で定義されています。

参照元 convMxM::exec(), convMx1::exec(), InputParam(), InputParamCheck(), CONV::PrintDFI(), PrintParam(), Read(), CONV::ReadDfiFiles(), convMxN::VoxelInit(), と CONV::WriteIndexDfiFile().

4.13.4.7 E_CONV_OUTPUT_MULTI_FILE_CAST InputParam::m_multiFileCasting [protected]

並列処理時のファイル割振り方法

InputParam.h の 78 行で定義されています。

参照元 InputParamCheck(), PrintParam(), と Read().

4.13.4.8 CIO::E_CIO_FORMAT 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 73 行で定義されています。

参照元 InputParamCheck(), PrintParam(), と Read().

4.13.4.9 CIO::E_CIO_OUTPUT_TYPE InputParam::m_out_format_type [protected]

出力形式 ascii,binary,FortranBinary

InputParam.h の 74 行で定義されています。

参照元 InputParam(), InputParamCheck(), PrintParam(), と Read().

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

出力先ディレクトリー名

InputParam.h の 70 行で定義されています。

参照元 InputParamCheck(), PrintParam(), と Read().

4.13.4.11 CIO::E_CIO_DTYPE InputParam::m_output_data_type [protected]

出力実数タイプ byte,short,int,float,double

InputParam.h の 72 行で定義されています。

参照元 InputParam(), InputParamCheck(), PrintParam(), と Read().

4.13.4.12 bool InputParam::m_output_dfi_on [protected]

dfi 出力指示フラグ

InputParam.h の 62 行で定義されています。

参照元 InputParam(), InputParamCheck(), PrintParam(), と Read().

4.13.4.13 CIO::E_CIO_ARRAYSHAPE InputParam::m_outputArrayShape [protected]

出力配列形状

InputParam.h の 76 行で定義されています。

参照元 InputParam(), InputParamCheck(), PrintParam(), と Read().

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

出力分割数 MxN で有効

InputParam.h の 67 行で定義されています。

参照元 InputParam(), PrintParam(), と Read().

4.13.4.15 CIO::E_CIO_OUTPUT_FNAME InputParam::m_outputFilenameFormat [protected]

出力ファイル名命名順

InputParam.h の 77 行で定義されています。

参照元 InputParam(), PrintParam(), と Read().

4.13.4.16 int InputParam::m_outputGuideCell [protected]

出力するガイドセル数

InputParam.h の 69 行で定義されています。

参照元 InputParam(), InputParamCheck(), PrintParam(), と Read().

4.13.4.17 cpm_ParaManager* InputParam::m_paraMngr

InputParam.h の 58 行で定義されています。

参照元 InputParam().

4.13.4.18 int InputParam::m_thin_count [protected]

間引き数

InputParam.h の 68 行で定義されています。

参照元 InputParam(), InputParamCheck(), PrintParam(), 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

4.14 構造体 CONV::step_rank_info

`#include <conv.h>`

Public 変数

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

4.14.1 説明

conv.h の 52 行で定義されています。

4.14.2 変数

4.14.2.1 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().

4.14.2.2 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().

4.14.2.3 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().

4.14.2.4 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().

4.14.2.5 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

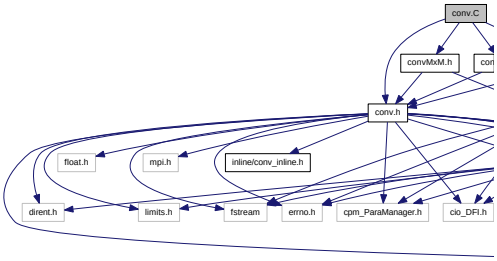
ファイル

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。

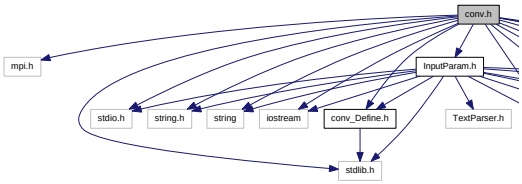
5.2 conv.h

CONV Class Header.

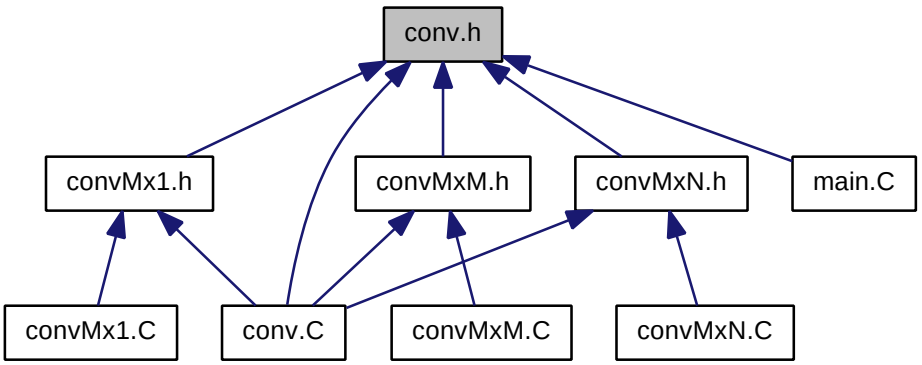
```
#include "mpi.h"
```

```
#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

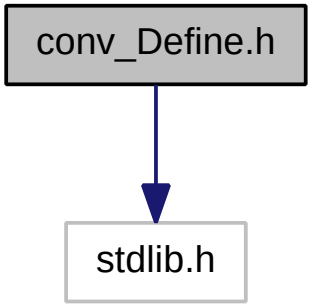
conv.h で定義されています。

5.3 conv_Define.h

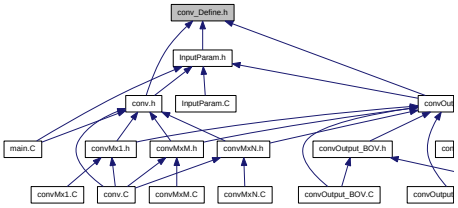
CONV Definition Header.

```
#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_CONV_OUTPUT_CONV_TYPE { E_CONV_OUTPUT_UNKNOWN = -1, E_CONV_OUTPUT_Mx1 = 0, E_CONV_OUTPUT_MxN, E_CONV_OUTPUT_MxM }
- enum E_CONV_OUTPUT_MULTI_FILE_CAST { E_CONV_OUTPUT_CAST_UNKNOWN = -1, E_CONV_OUTPUT_STEP = 0, E_CONV_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 82 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_BOV::OutputFile_Open(), convOutput_SPH::OutputFile_Open(), convOutput_VTK::OutputFile_Open(), convOutput_PLOT3D::OutputFile_Open(), convOutput_PLOT3D::OutputPlot3D_xyz(), InputParam::PrintParam(), InputParam::Read(), convMxN::VoxelInit(), convOutput_AVS::WriteFieldData(), convOutput_VTK::WriteFieldData(), convOutput::WriteFieldData(), と convOutput_BOV::WriteHeaderRecord().

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(), main(), と CONV::∼CONV().

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 61 行で定義されています。

5.3.2.9 #define ON 1

conv_Define.h の 60 行で定義されています。

5.3.2.10 #define REAL_UNKNOWN 0

conv_Define.h の 63 行で定義されています。

5.3.2.11 #define SPH_DATA_UNKNOWN 0

conv_Define.h の 67 行で定義されています。

5.3.2.12 #define SPH_DOUBLE 2

conv_Define.h の 65 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 64 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 68 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 69 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_CONV_OUTPUT_CONV_TYPE

コンバート形式

列挙型の値

E_CONV_OUTPUT_UNKNOWN
E_CONV_OUTPUT_Mx1 未定義

E_CONV_OUTPUT_MxN M 対 1.

E_CONV_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_CONV_OUTPUT_UNKNOWN = -1,
46   E_CONV_OUTPUT_Mx1 = 0,
47   E_CONV_OUTPUT_MxN,
48   E_CONV_OUTPUT_MxM
49 };
```

5.3.3.2 enum E_CONV_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_CONV_OUTPUT_CAST_UNKNOWN

E_CONV_OUTPUT_STEP

E_CONV_OUTPUT_RANK

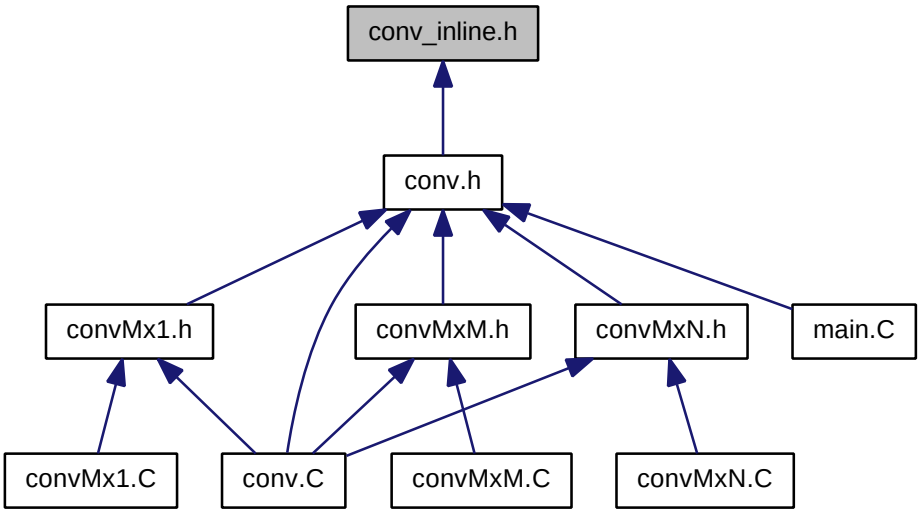
conv_Define.h の 52 行で定義されています。

```
53 {
54   E_CONV_OUTPUT_CAST_UNKNOWN = -1, //未定義
55   E_CONV_OUTPUT_STEP         = 0,  //step 基準
56   E_CONV_OUTPUT_RANK               //rank 基準
57 };
```

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。

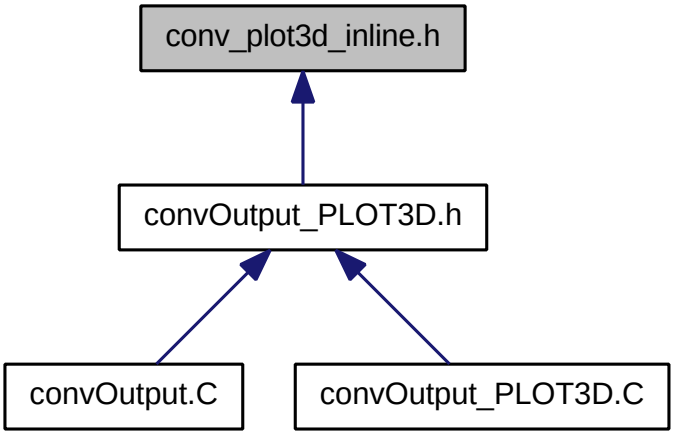
5.4.2 マクロ定義

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

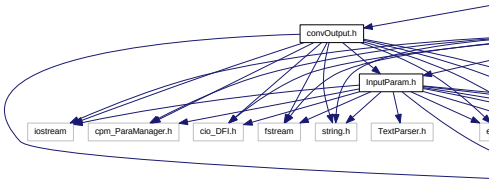
conv_plot3d_inline.h の 22 行で定義されています。

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

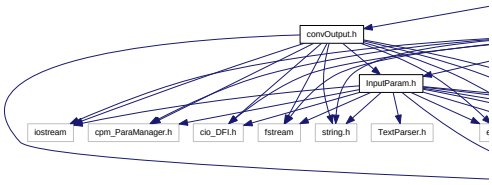
convMx1.C で定義されています。

5.7 convMx1.h

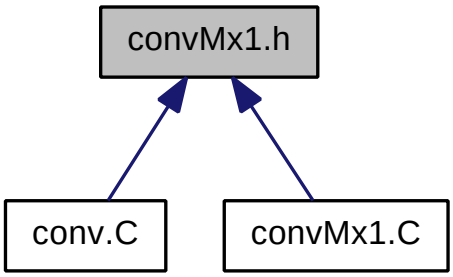
convMx1 Class Header

```
#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

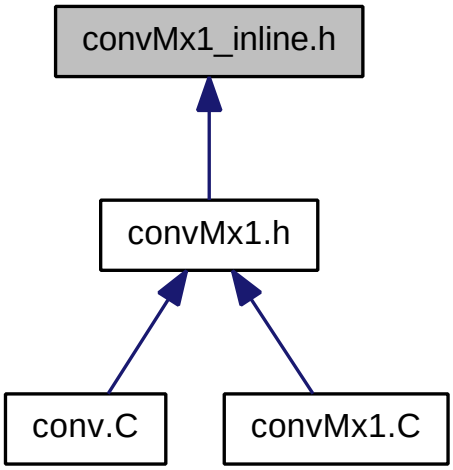
2013/11/14

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

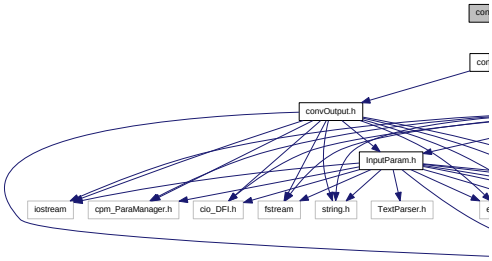
5.8.2.1 #define CONV_INLINE inline

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

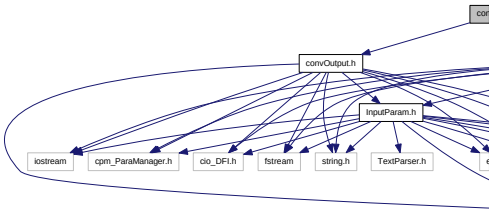
kero

convMxM.C で定義されています。

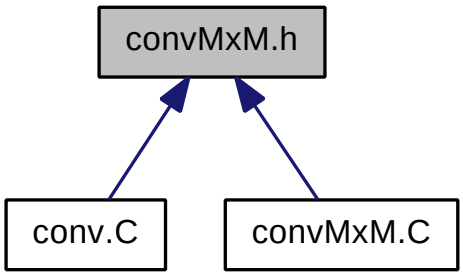
5.10 convMxM.h

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

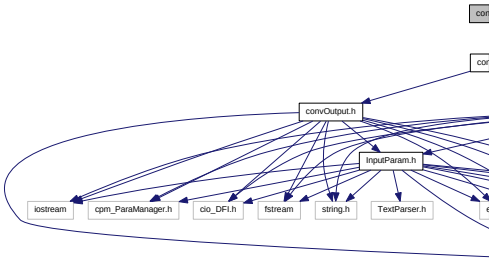
convMxM.h で定義されています。

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

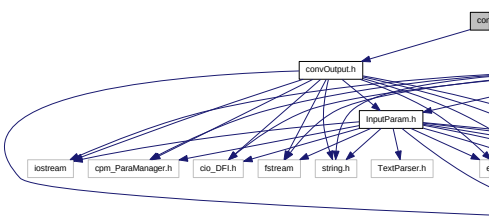
convMxN.C で定義されています。

5.12 convMxN.h

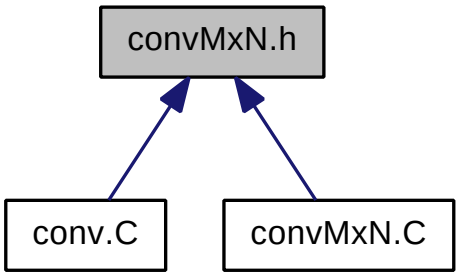
convMxN Class Header

```
#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

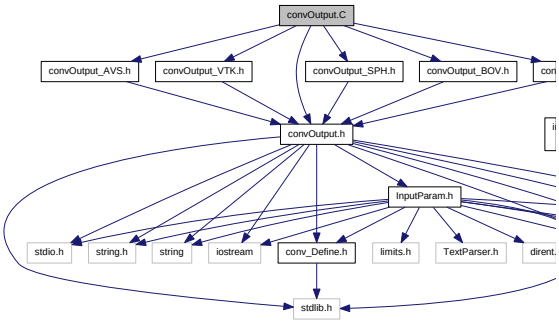
convMxN.h で定義されています。

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

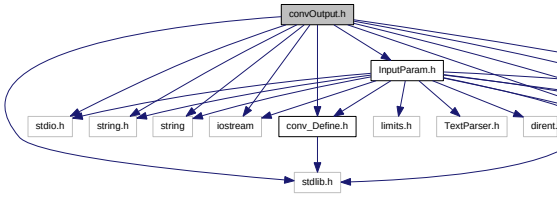
convOutput.C で定義されています。

5.14 convOutput.h

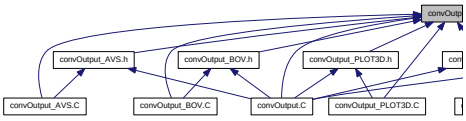
convOutput Class Header

```
#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

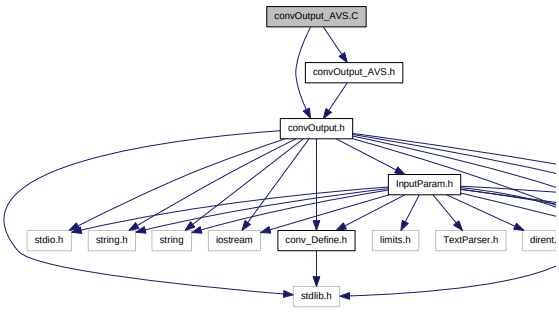
convOutput.h で定義されています。

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

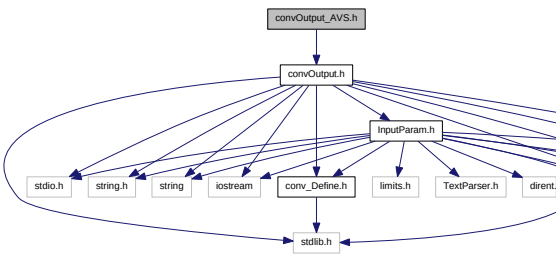
convOutput_AVS.C で定義されています。

5.16 convOutput_AVS.h

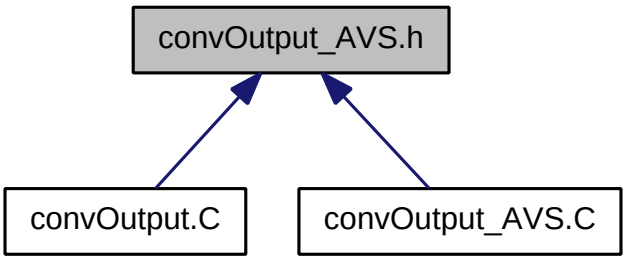
convOutput_AVS Class Header

```
#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

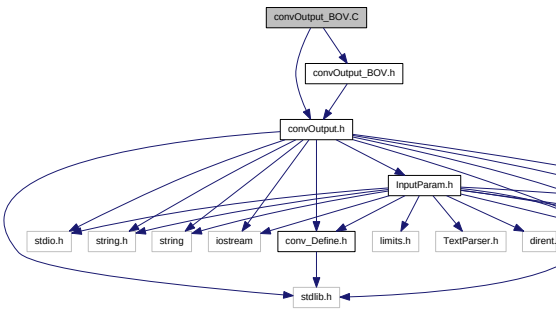
convOutput_AVS.h で定義されています。

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

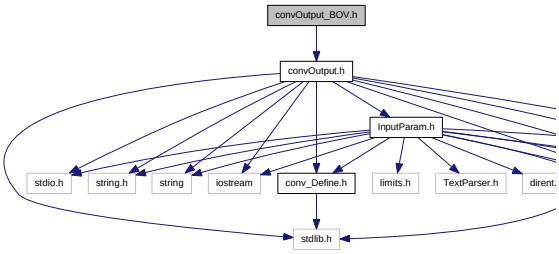
convOutput_BOV.C で定義されています。

5.18 convOutput_BOV.h

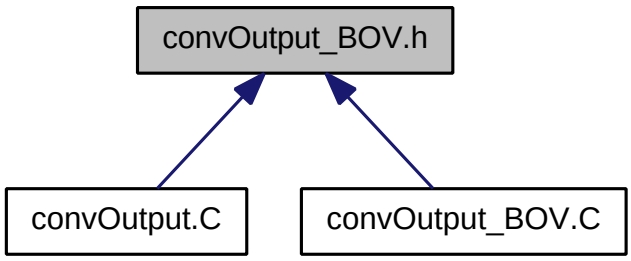
convOutput_BOV Class Header

```
#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

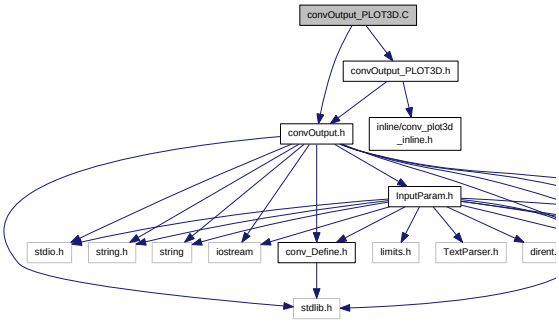
convOutput_BOV.h で定義されています。

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

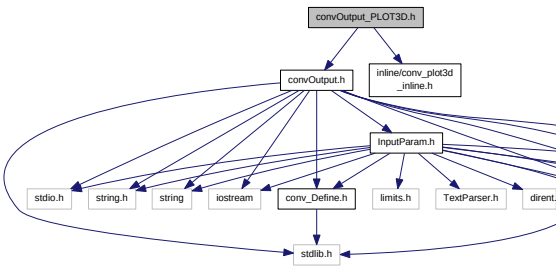
convOutput_PLOT3D.C で定義されています。

5.20 convOutput_PLOT3D.h

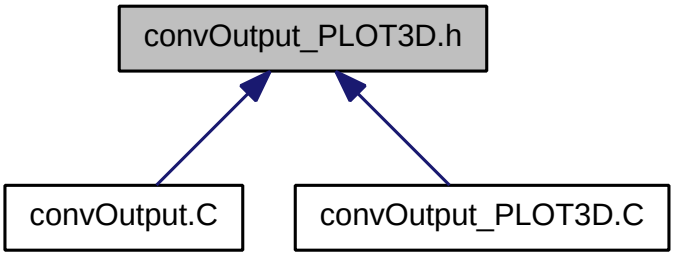
convOutput_PLOT3D Class Header

```
#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

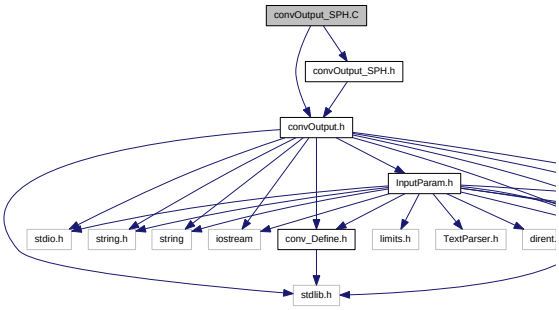
convOutput_PLOT3D.h で定義されています。

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

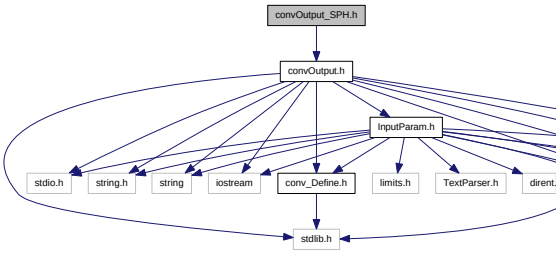
convOutput_SPH.C で定義されています。

5.22 convOutput_SPH.h

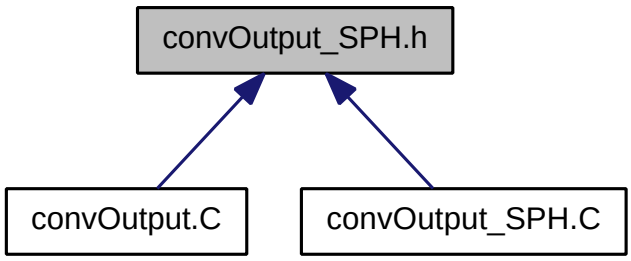
convOutput_SPH Class Header

```
#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

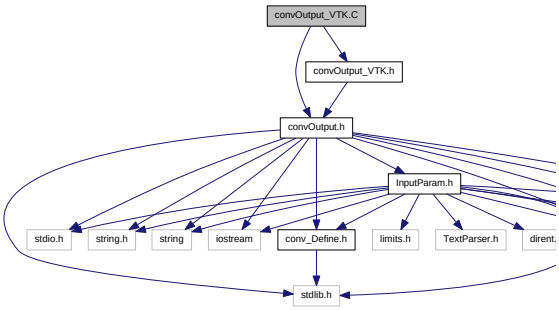
convOutput_SPH.h で定義されています。

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

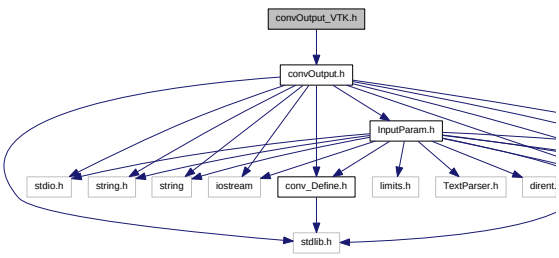
convOutput_VTK.C で定義されています。

5.24 convOutput_VTK.h

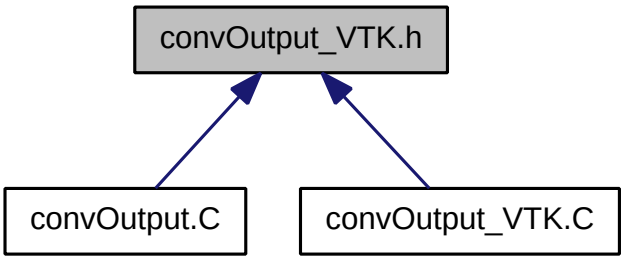
convOutput_VTK Class Header

```
#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

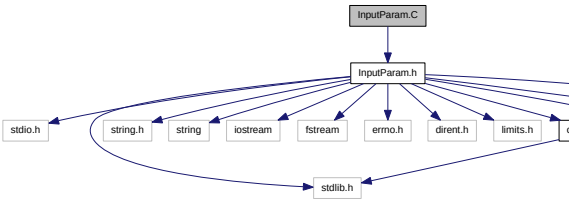
convOutput_VTK.h で定義されています。

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

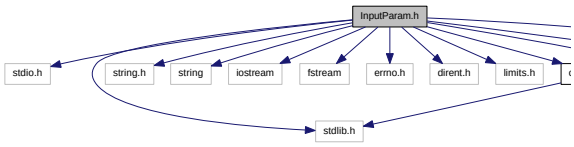
InputParam.C で定義されています。

5.26 InputParam.h

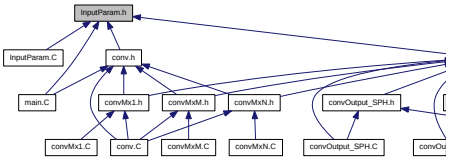
InputParam Class Header.

```
#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
- struct InputParam::dfi_info

5.26.1 説明

InputParam Class Header.

作者

kero

日付

2013/11/13

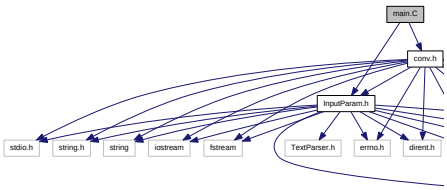
InputParam.h で定義されています。

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 35 行で定義されています。

参照先 CONV::CheckDir(), CONV::ConvInit(), CONV::exec(), InputParam::Get_OutputDir(), CONV::importCPM(), 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().

```
36 {
37   char *progname = argv[0];
38   bool out_comb = false;
39   bool out_log  = false;
40   bool thin_out = false;
41   int thin_count=1;
42   string fname;
43   string dname="";
44   //int pflag=0;//出力しない
45   int pflag=1;//出力する
46   int pflagv=0;
47   int lflag;
48   int lflagv;
49
50   // タイミング用変数
51   double t0, t1, t2, t3, t4, t5;
52
53   // ################################################################
54   // 初期処理
55
56   // 並列管理クラスのインスタンスと初期化
57   // ここで MPI_Init も行う
```

```
58   cpm_ParaManager* paraMngr = cpm_ParaManager::get_instance(argc, argv);
59   if( !paraMngr ) {
60     return CPM_ERROR_PM_INSTANCE;
61   }
62
63   // ####################################################################
64   // 入力オプション処理
65
66   int opt;
67   while ((opt = getopt(argc, argv, "avlf:d:hs:")) != -1) {
68     switch (opt) {
69     case 'f':
70       fname = optarg;
71       out_comb = true;
72       break;
73 /*
74     case 'd':
75       dname = optarg;
76       break;
77 */
78     case 'v':
79       pflagv = 1;
80       break;
81     case 'l':
82       out_log = true;
83       break;
84 /*
85     case 's':
86       thin_out = true;
87       thin_count = atoi(optarg);
88       break;
89 */
90     case 'h': // Show usage and exit
91       usage(progname);
92       return 0;
93       break;
94     case ':': // not find argument
95       usage(progname);
96       return 0;
97       break;
98     case '?': // unknown option
99       usage(progname);
100       return 0;
101       break;
102     }
103   }
104
105   // 入力ファイルが存在するかどうか
106   if( !(out_comb) ){
107     usage(progname);
108     return 0;
109   }
110
111   // 画面出力、ログ出力の整理
112   if(pflagv==1) pflag =1;
113   if(pflag ==0) pflagv=0;
114   lflag=0;
115   lflagv=0;
116   if(out_log){
117     lflag=pflag;
118     lflagv=pflagv;
119   }
120
121   // ####################################################################
122   // InputParam インスタンス
123   InputParam param(paraMngr);
124
125   // ####################################################################
126   // 入力ファイルの読込みとチェック
127   cout << endl;
128   cout << "Input Parameter File Read" << endl;
129   t0 = cpm_Base::GetWTime();
130   if( !param.Read(fname) ) {
131     cout << "Input Parameter File Read Error" << endl;
132     return 0;
133   };
134
135   // ####################################################################
136   // conv のインスタンス
137   CONV* conv = CONV::ConvInit(&param);
138   if( conv == NULL ) return 0;
139   if ( !conv->importCPM(paraMngr) )
140   {
141     return CPM_ERROR_PM_INSTANCE;
142   }
143
144   // ####################################################################
```

```
145   // ログファイルのオープン
146   int m_lflag=lflag;
147   LOG_OUT_ conv->OpenLogFile();
148
149   // ####################################################################
150   // 出力指定ディレクトリのチェック
151   if( dname.size() > 0 ) conv->CheckDir(dname);
152   conv->CheckDir(param.Get_OutputDir());
153
154   // ####################################################################
155   // 引数のセット
156   conv->m_pflag=pflag;
157   conv->m_pflagv=pflagv;
158   conv->m_lflag=lflag;
159   conv->m_lflagv=lflagv;
160
161   // ####################################################################
162   // dfi ファイルの読み込み
163   cout << endl;
164   cout << "ReadDfiFiles" << endl;
165   t1 = cpm_Base::GetWTime();
166   if( conv->ReadDfiFiles() != CIO::E_CIO_SUCCESS ) return 0;
167
168   t2 = cpm_Base::GetWTime();
169   // ####################################################################
170   // VoxelInit
171   conv->VoxelInit();
172
173   // ####################################################################
174   // 実行
175   if( !conv->exec() ) return 0;
176
177   // ####################################################################
178   // 終了処理
179   cout << endl;
180   cout << "converter finish" << endl;
181   t3 = cpm_Base::GetWTime();
182
183   double tt[4];
184   tt[0]=t1-t0;
185   tt[1]=t2-t1;
186   tt[2]=t3-t2;
187   tt[3]=t3-t0;
188
189   printf("\n\n");
190   printf("TIME : ReadInit      %10.3f sec.\n", tt[0]);
191   printf("TIME : ReadDfiFiles  %10.3f sec.\n", tt[1]);
192   printf("TIME : ConvertFiles  %10.3f sec.\n", tt[2]);
193   printf("TIME : Total Time    %10.3f sec.\n", tt[3]);
194   LOG_OUT_ conv->WriteTime(tt);
195
196   // ####################################################################
197   // ログファイルのクローズ（デストラクタへ移動）
198   //LOG_OUT_ conv->CloseLogFile();
199
200   // ####################################################################
201   // 並列環境の終了
202
203   return 0;
204 }
```

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     << "  -h           : Show usage and exit\n"
32     << std::endl;
33 }
```

Index