Cartesian Input/Output Library

1.3.6

作成： Doxygen 1.8.4

Tue Jan 7 2014 13:40:34

Contents

Chapter 1

ネームスペース索引

1.1 ネームスペース一覧

ネームスペースの一覧です。

Chapter 2

階層索引

2.1 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 3

構成索引

3.1 構成

クラス、構造体、共用体、インタフェースの説明です。

Chapter 4

ファイル索引

4.1 ファイル一覧

これはファイル一覧です。

Chapter 5

ネームスペース

5.1 ネームスペース CIO

列挙型

- enum E_CIO_FORMAT {
  E_CIO_FMT_UNKNOWN = -1, E_CIO_FMT_SPH, E_CIO_FMT_BOV, E_CIO_FMT_AVS,
  E_CIO_FMT_PLOT3D, E_CIO_FMT_VTK }
- enum E_CIO_ONOFF { E_CIO_OFF = 0, E_CIO_ON }
- enum E_CIO_DTYPE {
  E_CIO_DTYPE_UNKNOWN = 0, E_CIO_INT8, E_CIO_INT16, E_CIO_INT32,
  E_CIO_INT64, E_CIO_UINT8, E_CIO_UINT16, E_CIO_UINT32,
  E_CIO_UINT64, E_CIO_FLOAT32, E_CIO_FLOAT64 }
- enum E_CIO_ARRAYSHAPE { E_CIO_ARRAYSHAPE_UNKNOWN =-1, E_CIO_IJKN =0, E_CIO_NIJK }
- enum E_CIO_ENDIANTYPE { E_CIO_ENDIANTYPE_UNKNOWN =-1, E_CIO_LITTLE =0, E_CIO_BIG }
- enum E_CIO_READTYPE {
  E_CIO_SAMEDIV_SAMERES =1, E_CIO_SAMEDIV_REFINEMENT, E_CIO_DIFFDIV_SAMERES,
  E_CIO_DIFFDIV_REFINEMENT,
  E_CIO_READTYPE_UNKNOWN }
- enum E_CIO_OUTPUT_TYPE { E_CIO_OUTPUT_TYPE_DEFAULT =-1, E_CIO_OUTPUT_TYPE_ASCII
  =0, E_CIO_OUTPUT_TYPE_BINARY, E_CIO_OUTPUT_TYPE_FBINARY }
- enum E_CIO_OUTPUT_FNAME { E_CIO_FNAME_DEFAULT =-1, E_CIO_FNAME_STEP_RANK =0,
  E_CIO_FNAME_RANK_STEP }
- enum E_CIO_ERRORCODE {
  E_CIO_SUCCESS = 1, E_CIO_ERROR = -1, E_CIO_ERROR_READ_DFI_GLOBALORIGIN = 1000,
  E_CIO_ERROR_READ_DFI_GLOBALREGION = 1001,
  E_CIO_ERROR_READ_DFI_GLOBALVOXEL = 1002, E_CIO_ERROR_READ_DFI_GLOBALDIVISION =
  1003, E_CIO_ERROR_READ_DFI_DIRECTORYPATH = 1004, E_CIO_ERROR_READ_DFI_TIMESLICEDIRECTORY
  = 1005,
  E_CIO_ERROR_READ_DFI_PREFIX = 1006, E_CIO_ERROR_READ_DFI_FILEFORMAT = 1007,
  E_CIO_ERROR_READ_DFI_GUIDECELL = 1008, E_CIO_ERROR_READ_DFI_DATATYPE = 1009,
  E_CIO_ERROR_READ_DFI_ENDIAN = 1010, E_CIO_ERROR_READ_DFI_ARRAYSHAPE = 1011,
  E_CIO_ERROR_READ_DFI_COMPONENT = 1012, E_CIO_ERROR_READ_DFI_FILEPATH_PROCESS
  = 1013,
  E_CIO_ERROR_READ_DFI_NO_RANK = 1014, E_CIO_ERROR_READ_DFI_ID = 1015, E_CIO_ERROR_READ_DFI_HOST
  = 1016, E_CIO_ERROR_READ_DFI_VOXELSIZE = 1017,
  E_CIO_ERROR_READ_DFI_HEADINDEX = 1018, E_CIO_ERROR_READ_DFI_TAILINDEX = 1019,
  E_CIO_ERROR_READ_DFI_NO_SLICE = 1020, E_CIO_ERROR_READ_DFI_STEP = 1021,
  E_CIO_ERROR_READ_DFI_TIME = 1022, E_CIO_ERROR_READ_DFI_NO_MINMAX = 1023, E_CIO_ERROR_READ_DFI_
  = 1024, E_CIO_ERROR_READ_DFI_MAX = 1025,
  E_CIO_ERROR_READ_INDEXFILE_OPENERROR = 1050, E_CIO_ERROR_TEXTPARSER = 1051,

E_CIO_ERROR_READ_FILEINFO = 1052, E_CIO_ERROR_READ_FILEPATH = 1053,
E_CIO_ERROR_READ_UNIT = 1054, E_CIO_ERROR_READ_TIMESLICE = 1055, E_CIO_ERROR_READ_PROCFILE_OPE
= 1056, E_CIO_ERROR_READ_DOMAIN = 1057,
E_CIO_ERROR_READ_MPI = 1058, E_CIO_ERROR_READ_PROCESS = 1059, E_CIO_ERROR_READ_FIELDDATA_FILE
= 1900, E_CIO_ERROR_READ_SPH_FILE = 2000,
E_CIO_ERROR_READ_SPH_REC1 = 2001, E_CIO_ERROR_READ_SPH_REC2 = 2002, E_CIO_ERROR_READ_SPH_REC
= 2003, E_CIO_ERROR_READ_SPH_REC4 = 2004,
E_CIO_ERROR_READ_SPH_REC5 = 2005, E_CIO_ERROR_READ_SPH_REC6 = 2006, E_CIO_ERROR_READ_SPH_REC
= 2007, E_CIO_ERROR_UNMATCH_VOXELSIZE = 2050,
E_CIO_ERROR_NOMATCH_ENDIAN = 2051, E_CIO_ERROR_READ_BOV_FILE = 2100, E_CIO_ERROR_READ_FIELD_H
= 2102, E_CIO_ERROR_READ_FIELD_DATA_RECORD = 2103,
E_CIO_ERROR_READ_FIELD_AVERAGED_RECORD = 2104, E_CIO_ERROR_MISMATCH_NP_SUBDOMAIN
= 3003, E_CIO_ERROR_INVALID_DIVNUM = 3011, E_CIO_ERROR_OPEN_SBDM = 3012,
E_CIO_ERROR_READ_SBDM_HEADER = 3013, E_CIO_ERROR_READ_SBDM_FORMAT = 3014,
E_CIO_ERROR_READ_SBDM_DIV = 3015, E_CIO_ERROR_READ_SBDM_CONTENTS = 3016,
E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO = 3017, E_CIO_ERROR_MAKEDIRECTORY = 3100,
E_CIO_ERROR_OPEN_FIELDDATA = 3101, E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD =
3102,
E_CIO_ERROR_WRITE_FIELD_DATA_RECORD = 3103, E_CIO_ERROR_WRITE_FIELD_AVERAGED_RECORD
= 3104, E_CIO_ERROR_WRITE_SPH_REC1 = 3201, E_CIO_ERROR_WRITE_SPH_REC2 = 3202,
E_CIO_ERROR_WRITE_SPH_REC3 = 3203, E_CIO_ERROR_WRITE_SPH_REC4 = 3204, E_CIO_ERROR_WRITE_SPH_R
= 3205, E_CIO_ERROR_WRITE_SPH_REC6 = 3206,
E_CIO_ERROR_WRITE_SPH_REC7 = 3207, E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY = 3500,
E_CIO_ERROR_WRITE_PROCFILE_OPENERROR = 3501, E_CIO_ERROR_WRITE_DOMAIN = 3502,
E_CIO_ERROR_WRITE_MPI = 3503, E_CIO_ERROR_WRITE_PROCESS = 3504, E_CIO_ERROR_WRITE_RANKID
= 3505, E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY = 3510,
E_CIO_ERROR_WRITE_PREFIX_EMPTY = 3511, E_CIO_ERROR_WRITE_INDEXFILE_OPENERROR =
3512, E_CIO_ERROR_WRITE_FILEINFO = 3513, E_CIO_ERROR_WRITE_UNIT = 3514,
E_CIO_ERROR_WRITE_TIMESLICE = 3515, E_CIO_ERROR_WRITE_FILEPATH = 3516, E_CIO_WARN_GETUNIT
= 4000 }

関数

- char cioPath_getDelimChar ()
- std::string cioPath_getDelimString ()
- bool cioPath_hasDrive (const std::string &path)
- std::string vfvPath_emitDrive (std::string &path)
- bool cioPath_isAbsolute (const std::string &path)
- std::string cioPath_DirName (const std::string &path, const char dc=cioPath_getDelimChar())
- std::string cioPath_FileName (const std::string &path, const std::string &addext=std::string(""), const char dc=cioPath_getDelimChar())
- std::string cioPath_ConnectPath (std::string dirName, std::string fname)
- std::string ExtractPathWithoutExt (const std::string &fn)

5.1.1 説明

namespace の設定

5.1.2 列挙型

5.1.2.1 enum CIO::E_CIO_ARRAYSHAPE

配列形式

列挙型の値

E_CIO_ARRAYSHAPE_UNKNOWN 未定

E_CIO_IJKN ijkn

E_CIO_NIJK nijk

cio_Define.h の 96 行で定義されています。

```
97   {
98     E_CIO_ARRAYSHAPE_UNKNOWN=-1,
99     E_CIO_IJKN=0,
100     E_CIO_NIJK
101   };
```

5.1.2.2 enum CIO::E_CIO_DTYPE

データ形式

列挙型の値

E_CIO_DTYPE_UNKNOWN 未定

E_CIO_INT8 char

E_CIO_INT16 short

E_CIO_INT32 int

E_CIO_INT64 long long

E_CIO_UINT8 unsigned char

E_CIO_UINT16 unsigned short

E_CIO_UINT32 unsigned int

E_CIO_UINT64 unsigned long long

E_CIO_FLOAT32 float

E_CIO_FLOAT64 double

cio_Define.h の 80 行で定義されています。

```
81   {
82     E_CIO_DTYPE_UNKNOWN = 0,
83     E_CIO_INT8,
84     E_CIO_INT16,
85     E_CIO_INT32,
86     E_CIO_INT64,
87     E_CIO_UINT8,
88     E_CIO_UINT16,
89     E_CIO_UINT32,
90     E_CIO_UINT64,
91     E_CIO_FLOAT32,
92     E_CIO_FLOAT64
93   };
```

5.1.2.3 enum CIO::E_CIO_ENDIANTYPE

Endian 形式

列挙型の値

E_CIO_ENDIANTYPE_UNKNOWN

E_CIO_LITTLE

E_CIO_BIG

cio_Define.h の 104 行で定義されています。

```
105   {
106     E_CIO_ENDIANTYPE_UNKNOWN=-1,
107     E_CIO_LITTLE=0,
108     E_CIO_BIG
109   };
```

5.1.2.4 enum CIO::E_CIO_ERRORCODE

CIO のエラーコード

列挙型の値

E_CIO_SUCCESS 正常終了

E_CIO_ERROR エラー終了

E_CIO_ERROR_READ_DFI_GLOBALORIGIN DFI GlobalOrigin 読込みエラー

E_CIO_ERROR_READ_DFI_GLOBALREGION DFI GlobalRegion 読込みエラー

E_CIO_ERROR_READ_DFI_GLOBALVOXEL DFI GlobalVoxel 読込みエラー

E_CIO_ERROR_READ_DFI_GLOBALDIVISION DFI GlobalDivision 読込みエラー

E_CIO_ERROR_READ_DFI_DIRECTORYPATH DFI DirectoryPath 読込みエラー

E_CIO_ERROR_READ_DFI_TIMESLICEDIRECTORY DFI TimeSliceDirectoryPath 読込みエラー

E_CIO_ERROR_READ_DFI_PREFIX DFI Prefix 読込みエラー

E_CIO_ERROR_READ_DFI_FILEFORMAT DFI FileFormat 読込みエラー

E_CIO_ERROR_READ_DFI_GUIDECELL DFI GuideCell 読込みエラー

E_CIO_ERROR_READ_DFI_DATATYPE DFI DataType 読込みエラー

E_CIO_ERROR_READ_DFI_ENDIAN DFI Endian 読込みエラー

E_CIO_ERROR_READ_DFI_ARRAYSHAPE DFI ArrayShape 読込みエラー

E_CIO_ERROR_READ_DFI_COMPONENT DFI Component 読込みエラー

E_CIO_ERROR_READ_DFI_FILEPATH_PROCESS DFI FilePath/Process 読込みエラー

E_CIO_ERROR_READ_DFI_NO_RANK DFI Rank 要素なし

E_CIO_ERROR_READ_DFI_ID DFI ID 読込みエラー

E_CIO_ERROR_READ_DFI_HOSTNAME DFI HoatName 読込みエラー

E_CIO_ERROR_READ_DFI_VOXELSIZE DFI VoxelSize 読込みエラー

E_CIO_ERROR_READ_DFI_HEADINDEX DFI HeadIndex 読込みエラー

E_CIO_ERROR_READ_DFI_TAILINDEX DFI TailIndex 読込みエラー

E_CIO_ERROR_READ_DFI_NO_SLICE DFI TimeSlice 要素なし

E_CIO_ERROR_READ_DFI_STEP DFI Step 読込みエラー

E_CIO_ERROR_READ_DFI_TIME DFI Time 読込みエラー

E_CIO_ERROR_READ_DFI_NO_MINMAX DFI MinMax 要素なし

E_CIO_ERROR_READ_DFI_MIN DFI Min 読込みエラー

E_CIO_ERROR_READ_DFI_MAX DFI Max 読込みエラー

E_CIO_ERROR_READ_INDEXFILE_OPENERROR Index ファイルオープンエラー

E_CIO_ERROR_TEXTPARSER TextParser エラー

E_CIO_ERROR_READ_FILEINFO FileInfo 読込みエラー

E_CIO_ERROR_READ_FILEPATH FilePath 読込みエラー

E_CIO_ERROR_READ_UNIT UNIT 読込みエラー

E_CIO_ERROR_READ_TIMESLICE TimeSlice 読込みエラー

E_CIO_ERROR_READ_PROCFILE_OPENERROR Proc ファイルオープンエラー
E_CIO_ERROR_READ_DOMAIN Domain 読込みエラー
E_CIO_ERROR_READ_MPI MPI 読込みエラー
E_CIO_ERROR_READ_PROCESS Process 読込みエラー
E_CIO_ERROR_READ_FIELDDATA_FILE フィールドデータファイル読込みエラー
E_CIO_ERROR_READ_SPH_FILE SPH ファイル読込みエラー
E_CIO_ERROR_READ_SPH_REC1 SPH ファイルレコード 1 読込みエラー
E_CIO_ERROR_READ_SPH_REC2 SPH ファイルレコード 2 読込みエラー
E_CIO_ERROR_READ_SPH_REC3 SPH ファイルレコード 3 読込みエラー
E_CIO_ERROR_READ_SPH_REC4 SPH ファイルレコード 4 読込みエラー
E_CIO_ERROR_READ_SPH_REC5 SPH ファイルレコード 5 読込みエラー
E_CIO_ERROR_READ_SPH_REC6 SPH ファイルレコード 6 読込みエラー
E_CIO_ERROR_READ_SPH_REC7 SPH ファイルレコード 7 読込みエラー
E_CIO_ERROR_UNMATCH_VOXELSIZE SPH のボクセルサイズとDFI のボクセルサイズが合致しない
E_CIO_ERROR_NOMATCH_ENDIAN 出力Fornat が合致しない（Endian 形式がBig,Little 以外）
E_CIO_ERROR_READ_BOV_FILE BOV ファイル読込みエラー
E_CIO_ERROR_READ_FIELD_HEADER_RECORD フィールドヘッダーレコード読込み失敗
E_CIO_ERROR_READ_FIELD_DATA_RECORD フィールドデータレコード読込み失敗
E_CIO_ERROR_READ_FIELD_AVERAGED_RECORD フィールドAverage 読込み失敗
E_CIO_ERROR_MISMATCH_NP_SUBDOMAIN 並列数とサブドメイン数が一致していない
E_CIO_ERROR_INVALID_DIVNUM 領域分割数が不正
E_CIO_ERROR_OPEN_SBDM ActiveSubdomain ファイルのオープンに失敗
E_CIO_ERROR_READ_SBDM_HEADER ActiveSubdomain ファイルのヘッダー読み込みに失敗
E_CIO_ERROR_READ_SBDM_FORMAT ActiveSubdomain ファイルのフォーマットエラー
E_CIO_ERROR_READ_SBDM_DIV ActiveSubdomain ファイルの領域分割数読み込みに失敗
E_CIO_ERROR_READ_SBDM_CONTENTS ActiveSubdomain ファイルのContents 読み込みに失敗
E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO ActiveSubdomain ファイルの活性ドメイン数が 0.
E_CIO_ERROR_MAKEDIRECTORY Directory 生成で失敗
E_CIO_ERROR_OPEN_FIELDDATA フィールドデータのオープンに失敗
E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD フィールドヘッダーレコード出力失敗
E_CIO_ERROR_WRITE_FIELD_DATA_RECORD フィールドデータレコード出力失敗
E_CIO_ERROR_WRITE_FIELD_AVERAGED_RECORD フィールドAverage 出力失敗
E_CIO_ERROR_WRITE_SPH_REC1 SPH ファイルレコード 1 出力エラー
E_CIO_ERROR_WRITE_SPH_REC2 SPH ファイルレコード 2 出力エラー
E_CIO_ERROR_WRITE_SPH_REC3 SPH ファイルレコード 3 出力エラー
E_CIO_ERROR_WRITE_SPH_REC4 SPH ファイルレコード 4 出力エラー
E_CIO_ERROR_WRITE_SPH_REC5 SPH ファイルレコード 5 出力エラー
E_CIO_ERROR_WRITE_SPH_REC6 SPH ファイルレコード 6 出力エラー
E_CIO_ERROR_WRITE_SPH_REC7 SPH ファイルレコード 7 出力エラー
E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY proc dfi ファイル名が未定義
E_CIO_ERROR_WRITE_PROCFILE_OPENERROR proc dfi ファイルオープン失敗
E_CIO_ERROR_WRITE_DOMAIN Domain 出力失敗
E_CIO_ERROR_WRITE_MPI MPI 出力失敗
E_CIO_ERROR_WRITE_PROCESS Process 出力失敗
E_CIO_ERROR_WRITE_RANKID 出力ランク以外

E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY index dfi ファイル名が未定義

E_CIO_ERROR_WRITE_PREFIX_EMPTY Prefix が未定義

E_CIO_ERROR_WRITE_INDEXFILE_OPENERROR proc dfi ファイルオープン失敗

E_CIO_ERROR_WRITE_FILEINFO FileInfo 出力失敗

E_CIO_ERROR_WRITE_UNIT Unit 出力失敗

E_CIO_ERROR_WRITE_TIMESLICE TimeSlice 出力失敗

E_CIO_ERROR_WRITE_FILEPATH FilePath 出力失敗

E_CIO_WARN_GETUNIT Unit の単位がない

cio_Define.h の 140 行で定義されています。

```
141   {
142     E_CIO_SUCCESS                          = 1
143 ,   E_CIO_ERROR                            = -1
144 ,   E_CIO_ERROR_READ_DFI_GLOBALORIGIN      = 1000
145 ,   E_CIO_ERROR_READ_DFI_GLOBALREGION      = 1001
146 ,   E_CIO_ERROR_READ_DFI_GLOBALVOXEL       = 1002
147 ,   E_CIO_ERROR_READ_DFI_GLOBALDIVISION    = 1003
148 ,   E_CIO_ERROR_READ_DFI_DIRECTORYPATH     = 1004
149 ,   E_CIO_ERROR_READ_DFI_TIMESLICEDIRECTORY = 1005
150 ,   E_CIO_ERROR_READ_DFI_PREFIX            = 1006
151 ,   E_CIO_ERROR_READ_DFI_FILEFORMAT        = 1007
152 ,   E_CIO_ERROR_READ_DFI_GUIDECELL        = 1008
153 ,   E_CIO_ERROR_READ_DFI_DATATYPE         = 1009
154 ,   E_CIO_ERROR_READ_DFI_ENDIAN           = 1010
155 ,   E_CIO_ERROR_READ_DFI_ARRAYSHAPE       = 1011
156 ,   E_CIO_ERROR_READ_DFI_COMPONENT        = 1012
157 ,   E_CIO_ERROR_READ_DFI_FILEPATH_PROCESS = 1013
158 ,   E_CIO_ERROR_READ_DFI_NO_RANK          = 1014
159 ,   E_CIO_ERROR_READ_DFI_ID               = 1015
160 ,   E_CIO_ERROR_READ_DFI_HOSTNAME         = 1016
161 ,   E_CIO_ERROR_READ_DFI_VOXELSIZE        = 1017
162 ,   E_CIO_ERROR_READ_DFI_HEADINDEX        = 1018
163 ,   E_CIO_ERROR_READ_DFI_TAILINDEX        = 1019
164 ,   E_CIO_ERROR_READ_DFI_NO_SLICE         = 1020
165 ,   E_CIO_ERROR_READ_DFI_STEP             = 1021
166 ,   E_CIO_ERROR_READ_DFI_TIME             = 1022
167 ,   E_CIO_ERROR_READ_DFI_NO_MINMAX        = 1023
168 ,   E_CIO_ERROR_READ_DFI_MIN              = 1024
169 ,   E_CIO_ERROR_READ_DFI_MAX              = 1025
170 ,   E_CIO_ERROR_READ_INDEXFILE_OPENERROR  = 1050
171 ,   E_CIO_ERROR_TEXTPARSER                = 1051
172 ,   E_CIO_ERROR_READ_FILEINFO             = 1052
173 ,   E_CIO_ERROR_READ_FILEPATH             = 1053
174 ,   E_CIO_ERROR_READ_UNIT                 = 1054
175 ,   E_CIO_ERROR_READ_TIMESLICE            = 1055
176 ,   E_CIO_ERROR_READ_PROCFILE_OPENERROR   = 1056
177 ,   E_CIO_ERROR_READ_DOMAIN               = 1057
178 ,   E_CIO_ERROR_READ_MPI                  = 1058
179 ,   E_CIO_ERROR_READ_PROCESS              = 1059
180 ,   E_CIO_ERROR_READ_FIELDDATA_FILE       = 1900
181 ,   E_CIO_ERROR_READ_SPH_FILE             = 2000
182 ,   E_CIO_ERROR_READ_SPH_REC1             = 2001
183 ,   E_CIO_ERROR_READ_SPH_REC2             = 2002
184 ,   E_CIO_ERROR_READ_SPH_REC3             = 2003
185 ,   E_CIO_ERROR_READ_SPH_REC4             = 2004
186 ,   E_CIO_ERROR_READ_SPH_REC5             = 2005
187 ,   E_CIO_ERROR_READ_SPH_REC6             = 2006
188 ,   E_CIO_ERROR_READ_SPH_REC7             = 2007
189 ,   E_CIO_ERROR_UNMATCH_VOXELSIZE         = 2050
190 ,   E_CIO_ERROR_NOMATCH_ENDIAN            = 2051
191 ,   E_CIO_ERROR_READ_BOV_FILE             = 2100
192 ,   E_CIO_ERROR_READ_FIELD_HEADER_RECORD  = 2102
193 ,   E_CIO_ERROR_READ_FIELD_DATA_RECORD    = 2103
194 ,   E_CIO_ERROR_READ_FIELD_AVERAGED_RECORD = 2104
195 //,   E_CIO_ERROR_DATATYPE                  = 2500 ///< DataType error
196 ,   E_CIO_ERROR_MISMATCH_NP_SUBDOMAIN     = 3003
197 ,   E_CIO_ERROR_INVALID_DIVNUM            = 3011
198 ,   E_CIO_ERROR_OPEN_SBDM                 = 3012
199 ,   E_CIO_ERROR_READ_SBDM_HEADER          = 3013
200 ,   E_CIO_ERROR_READ_SBDM_FORMAT          = 3014
201 ,   E_CIO_ERROR_READ_SBDM_DIV             = 3015
202 ,   E_CIO_ERROR_READ_SBDM_CONTENTS        = 3016
203 ,   E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO       = 3017
204 ,   E_CIO_ERROR_MAKEDIRECTORY            = 3100
205 ,   E_CIO_ERROR_OPEN_FIELDDATA           = 3101
206 ,   E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD = 3102
207 ,   E_CIO_ERROR_WRITE_FIELD_DATA_RECORD   = 3103
208 ,   E_CIO_ERROR_WRITE_FIELD_AVERAGED_RECORD = 3104
```

```
209 ,   E_CIO_ERROR_WRITE_SPH_REC1              = 3201
210 ,   E_CIO_ERROR_WRITE_SPH_REC2              = 3202
211 ,   E_CIO_ERROR_WRITE_SPH_REC3              = 3203
212 ,   E_CIO_ERROR_WRITE_SPH_REC4              = 3204
213 ,   E_CIO_ERROR_WRITE_SPH_REC5              = 3205
214 ,   E_CIO_ERROR_WRITE_SPH_REC6              = 3206
215 ,   E_CIO_ERROR_WRITE_SPH_REC7              = 3207
216 ,   E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY    = 3500
217 ,   E_CIO_ERROR_WRITE_PROCFILE_OPENERROR    = 3501
218 ,   E_CIO_ERROR_WRITE_DOMAIN                = 3502
219 ,   E_CIO_ERROR_WRITE_MPI                   = 3503
220 ,   E_CIO_ERROR_WRITE_PROCESS               = 3504
221 ,   E_CIO_ERROR_WRITE_RANKID                = 3505
222 ,   E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY   = 3510
223 ,   E_CIO_ERROR_WRITE_PREFIX_EMPTY          = 3511
224 ,   E_CIO_ERROR_WRITE_INDEXFILE_OPENERROR   = 3512
225 ,   E_CIO_ERROR_WRITE_FILEINFO              = 3513
226 ,   E_CIO_ERROR_WRITE_UNIT                  = 3514
227 ,   E_CIO_ERROR_WRITE_TIMESLICE             = 3515
228 ,   E_CIO_ERROR_WRITE_FILEPATH              = 3516
229 ,   E_CIO_WARN_GETUNIT                      = 4000
230   };
```

5.1.2.5 enum CIO::E_CIO_FORMAT

File 形式

列挙型の値

E_CIO_FMT_UNKNOWN 未定

E_CIO_FMT_SPH sph format

E_CIO_FMT_BOV bov format

E_CIO_FMT_AVS avs format

E_CIO_FMT_PLOT3D plot3d format

E_CIO_FMT_VTK plot3d vtk format

cio_Define.h の 59 行で定義されています。

```
60   {
61     E_CIO_FMT_UNKNOWN = -1,
62     E_CIO_FMT_SPH,
63 //FCONV 20131122.s
64     //E_CIO_FMT_BOV            ///< bov format
65     E_CIO_FMT_BOV,
66     E_CIO_FMT_AVS,
67     E_CIO_FMT_PLOT3D,
68     E_CIO_FMT_VTK
69 //FCONV 20131122.e
70   };
```

5.1.2.6 enum CIO::E_CIO_ONOFF

スイッチ on or off

列挙型の値

E_CIO_OFF off

E_CIO_ON on

cio_Define.h の 73 行で定義されています。

```
74   {
75     E_CIO_OFF = 0,
76     E_CIO_ON
77   };
```

5.1.2.7 enum CIO::E_CIO_OUTPUT_FNAME

列挙型の値

E_CIO_FNAME_DEFAULT 出力ファイル命名規約デフォルト (step_rank)

E_CIO_FNAME_STEP_RANK step_rank

E_CIO_FNAME_RANK_STEP rank_step

cio_Define.h の 131 行で定義されています。

```
132   {
133     E_CIO_FNAME_DEFAULT=-1,
134     E_CIO_FNAME_STEP_RANK=0,
135     E_CIO_FNAME_RANK_STEP
136   };
```

5.1.2.8 enum CIO::E_CIO_OUTPUT_TYPE

出力形式

列挙型の値

E_CIO_OUTPUT_TYPE_DEFAULT デフォルト（binary)

E_CIO_OUTPUT_TYPE_ASCII ascii 形式での出力

E_CIO_OUTPUT_TYPE_BINARY binary 形式での出力

E_CIO_OUTPUT_TYPE_FBINARY Fortran Binary での出力

cio_Define.h の 123 行で定義されています。

```
124   {
125     E_CIO_OUTPUT_TYPE_DEFAULT=-1,
126     E_CIO_OUTPUT_TYPE_ASCII=0,
127     E_CIO_OUTPUT_TYPE_BINARY,
128     E_CIO_OUTPUT_TYPE_FBINARY
129   };
```

5.1.2.9 enum CIO::E_CIO_READTYPE

読込みタイプコード

列挙型の値

E_CIO_SAMEDIV_SAMERES 同一分割＆同一密度

E_CIO_SAMEDIV_REFINEMENT 同一分割＆粗密

E_CIO_DIFFDIV_SAMERES MxN ＆同一密度

E_CIO_DIFFDIV_REFINEMENT MxN ＆粗密

E_CIO_READTYPE_UNKNOWN error

cio_Define.h の 112 行で定義されています。

```
113   {
114     E_CIO_SAMEDIV_SAMERES=1,
115     E_CIO_SAMEDIV_REFINEMENT,
116     E_CIO_DIFFDIV_SAMERES,
117     E_CIO_DIFFDIV_REFINEMENT,
118     E_CIO_READTYPE_UNKNOWN,
119   };
```

5.1.3 関数

5.1.3.1 std::string CIO::cioPath_ConnectPath (std::string *dirName,* std::string *fname*) [inline]

cio_PathUtil.h の 169 行で定義されています。

参照先 cioPath_getDelimChar(), と cioPath_getDelimString().

参照元 cio_DFI::Generate_DFI_Name(), cio_DFI::Generate_Directory_Path(), cio_DFI::ReadData(), cio_DFI::ReadInit(), と cio_DFI::WriteData().

```
170   {
171     std::string path = dirName;
172
173     const char *p = dirName.c_str();
174     if( p[strlen(p)-1] != CIO::cioPath_getDelimChar() )
175     {
176       path += CIO::cioPath_getDelimString();
177     }
178
179     path += fname;
180
181     return path;
182   }
```

5.1.3.2 std::string CIO::cioPath_DirName (const std::string & *path,* const char *dc* = cioPath_getDelimChar()) [inline]

cio_PathUtil.h の 67 行で定義されています。

参照先 cioPath_isAbsolute().

参照元 cio_DFI::Generate_DFI_Name(), cio_DFI::Generate_Directory_Path(), cio_DFI::MakeDirectorySub(), cio_DFI::ReadData(), cio_DFI::ReadInit(), cio_DFI::WriteData(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

```
68                                                                  {
69     char* name = strdup( path.c_str() );
70     char* p = name;
71
72     for ( ; ; ++p ) {
73       if ( ! *p ) {
74         if ( p > name ) {
75           char rs[2] = {dc, '\0'};
76           return rs;
77         } else {
78           char rs[3] = {'.', dc, '\0'};
79           return rs;
80         }
81       }
82       if ( *p != dc ) break;
83     }
84
85     for ( ; *p; ++p );
86     while ( *--p == dc ) continue;
87     *++p = '\0';
88
89     while ( --p >= name )
90       if ( *p == dc ) break;
91     ++p;
92     if ( p == name )
93     {
94       char rs[3] = {'.', dc, '\0'};
95       return rs;
96     }
97
98     while ( --p >= name )
99       if ( *p != dc ) break;
100     ++p;
101
102     *p = '\0';
103     if( p == name ) {
104       char rs[2] = {dc, '\0'};
105       return rs;
106     } else {
107       std::string s( name );
108       free( name );
109       if( !CIO::cioPath_isAbsolute(s) )
```

```
110       {
111         const char *q = s.c_str();
112         if( q[0] != '.' && q[1] != '/' )
113         {
114           char rs[3] = {'.', dc, '\0'};
115           s = std::string(rs) + s;
116         }
117       }
118       return s;
119     }
120   }
```

5.1.3.3 std::string CIO::cioPath_FileName (const std::string & *path*, const std::string & *addext* = `std::string("")`, const char *dc* = `cioPath_getDelimChar()`) [inline]

cio_PathUtil.h の 122 行で定義されています。

参照元 cio_DFI::Generate_DFI_Name(), cio_DFI::ReadInit(), cio_DFI::WriteData(), と cio_DFI::WriteProcDfiFile().

```
124                                                                                                 {
125     char* name = strdup( path.c_str() );
126     char* p = name;
127 
128     for ( ; ; ++p ) {
129       if ( ! *p ) {
130         if ( p > name ) {
131           char rs[2] = {dc, '\0'};
132           return rs;
133         } else
134           return "";
135       }
136       if ( *p != dc ) break;
137     }
138 
139     for ( ; *p; ++p ) continue;
140     while ( *--p == dc ) continue;
141     *++p = '\0';
142 
143     while ( --p >= name )
144       if ( *p == dc ) break;
145     ++p;
146 
147     bool add = false;
148     if ( addext.length() > 0 ) {
149       const int suffixlen = addext.length();
150       const int stringlen = strlen( p );
151       if ( suffixlen < stringlen ) {
152         const int off = stringlen - suffixlen;
153         if ( strcasecmp( p + off, addext.c_str()) != 0 )
154           add = true;
155       }
156       else
157       {
158           add = true;
159       }
160     }
161 
162     std::string s( p );
163     if( add ) s += addext;
164 
165     free( name );
166     return s;
167   }
```

5.1.3.4 char CIO::cioPath_getDelimChar ( ) [inline]

cio_PathUtil.h の 21 行で定義されています。

参照元 cioPath_ConnectPath(), cioPath_getDelimString(), と cioPath_isAbsolute().

```
22   {
23 #ifdef WIN32
24     return '\\';
25 #else
26     return '/';
27 #endif
28   }
```

5.1.3.5 std::string CIO::cioPath_getDelimString ( ) [inline]

cio_PathUtil.h の 30 行で定義されています。

参照先 cioPath_getDelimChar().

参照元 cioPath_ConnectPath().

```
31   {
32     const char dc = CIO::cioPath_getDelimChar();
33     char rs[2] = {dc, '\0'};
34     return rs;
35   }
```

5.1.3.6 bool CIO::cioPath_hasDrive (const std::string & *path*) [inline]

cio_PathUtil.h の 37 行で定義されています。

参照元 vfvPath_emitDrive().

```
37                                                    {
38     if ( path.size() < 2 ) return false;
39     char x = path[0];
40     if ( ((x >= 'A' && x <= 'Z' ) || (x >= 'a' && x <= 'z')) &&
41    path[1] == ':' )
42       return true;
43     return false;
44   }
```

5.1.3.7 bool CIO::cioPath_isAbsolute (const std::string & *path*) [inline]

cio_PathUtil.h の 57 行で定義されています。

参照先 cioPath_getDelimChar(), と vfvPath_emitDrive().

参照元 cioPath_DirName(), cio_DFI::Generate_Directory_Path(), cio_DFI::ReadData(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_avs_cord(), cio_DFI_AVS::write_avs_header(), cio_DFI_PLOT3D::write_GridData(), と cio_DFI::WriteData().

```
58   {
59     std::string xpath(path);
60     vfvPath_emitDrive(xpath);
61     char c1, c2;
62     c1 = xpath[0];
63     c2 = cioPath_getDelimChar();
64     return (c1 == c2);
65   }
```

5.1.3.8 std::string CIO::ExtractPathWithoutExt (const std::string & *fn*) [inline]

cio_PathUtil.h の 185 行で定義されています。

```
186   {
187     std::string::size_type pos;
188     if((pos = fn.find_last_of(".")) == std::string::npos){
189         return fn;
190     }
191
192     return fn.substr(0, pos);
193   }
```

5.1.3.9 std::string CIO::vfvPath_emitDrive (std::string & *path*) [inline]

cio_PathUtil.h の 46 行で定義されています。

参照先 cioPath_hasDrive().

参照元 cioPath_isAbsolute().

```
47   {
48     // returns drive (ex. 'C:')
49     if ( ! cioPath_hasDrive(path) ) return std::string();
50     std::string driveStr = path.substr(0, 2);
51     path = path.substr(2);
52     return driveStr;
53   }
```

Chapter 6

クラス

6.1 クラス cio_ActiveSubDomain

```
#include <cio_ActiveSubDomain.h>
```

Public メソッド

- cio_ActiveSubDomain ()
- cio_ActiveSubDomain (int pos[3])
- virtual ~cio_ActiveSubDomain ()
- virtual void clear ()
- void SetPos (int pos[3])
- const int * GetPos () const
- bool operator== (cio_ActiveSubDomain dom)
- bool operator!= (cio_ActiveSubDomain dom)

Private 変数

- int m_pos [3]
  領域分割内での位置

6.1.1 説明

ActiveSubDomian class

cio_ActiveSubDomain.h の 19 行で定義されています。

6.1.2 コンストラクタとデストラクタ

6.1.2.1 cio_ActiveSubDomain::cio_ActiveSubDomain ( )

デフォルトコンストラクタ

cio_ActiveSubDomain.C の 19 行で定義されています。

参照先 clear().

```
20 {
21   clear();
22 }
```

6.1.2.2 cio_ActiveSubDomain::cio_ActiveSubDomain (int *pos[3]*)

コンストラクタ

引数

in	*pos*	領域分割内での位置

cio_ActiveSubDomain.C の 26 行で定義されています。

参照先 SetPos().

```
27 {
28   SetPos(pos);
29 }
```

6.1.2.3 cio_ActiveSubDomain::∼cio_ActiveSubDomain ( ) [virtual]

デストラクタ

cio_ActiveSubDomain.C の 33 行で定義されています。

```
34 {
35 }
```

6.1.3 関数

6.1.3.1 void cio_ActiveSubDomain::clear ( ) [virtual]

情報のクリア

cio_ActiveSubDomain.C の 39 行で定義されています。

参照先 m_pos.

参照元 cio_ActiveSubDomain().

```
40 {
41   m_pos[0]=0;
42   m_pos[1]=0;
43   m_pos[2]=0;
44 }
```

6.1.3.2 const int ∗ cio_ActiveSubDomain::GetPos ( ) const

位置の取得

戻り値

位置情報整数配列のポインタ

cio_ActiveSubDomain.C の 57 行で定義されています。

参照先 m_pos.

参照元 cio_Process::CreateRankMap().

```
58 {
59   return m_pos;
60 }
```

6.1.3.3 bool cio_ActiveSubDomain::operator!= (cio_ActiveSubDomain *dom*)

比較演算子

引数

in	*dom*	比較対象の活性サブドメイン情報

戻り値

true	違う位置情報を持つ
false	同じ位置情報を持つ

cio_ActiveSubDomain.C の 74 行で定義されています。

参照先 m_pos.

```
75 {
76   if( m_pos[0] == dom.m_pos[0] ) return false;
77   if( m_pos[1] == dom.m_pos[1] ) return false;
78   if( m_pos[2] == dom.m_pos[2] ) return false;
79   return true;
80 }
```

6.1.3.4 bool cio_ActiveSubDomain::operator== (cio_ActiveSubDomain *dom*)

比較演算子

引数

in	*dom*	比較対象の活性サブドメイン情報

戻り値

true	同じ位置情報を持つ
false	違う位置情報を持つ

cio_ActiveSubDomain.C の 64 行で定義されています。

参照先 m_pos.

```
65 {
66   if( m_pos[0] != dom.m_pos[0] ) return false;
67   if( m_pos[1] != dom.m_pos[1] ) return false;
68   if( m_pos[2] != dom.m_pos[2] ) return false;
69   return true;
70 }
```

6.1.3.5 void cio_ActiveSubDomain::SetPos (int *pos[3]*)

位置のセット

引数

in	*pos*	領域分割内での位置

cio_ActiveSubDomain.C の 48 行で定義されています。

参照先 m_pos.

参照元 cio_ActiveSubDomain().

```
49 {
50   m_pos[0] = pos[0];
51   m_pos[1] = pos[1];
52   m_pos[2] = pos[2];
53 }
```

6.1.4 変数

6.1.4.1 int cio_ActiveSubDomain::m_pos[3] [private]

領域分割内での位置

cio_ActiveSubDomain.h の 63 行で定義されています。

参照元 clear(), GetPos(), operator!=(), operator==(), と SetPos().

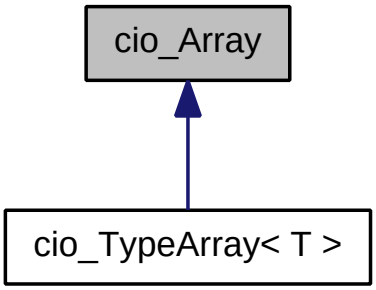
このクラスの説明は次のファイルから生成されました:

- cio_ActiveSubDomain.h
- cio_ActiveSubDomain.C

6.2 クラス cio_Array

`#include <cio_Array.h>`

cio_Array に対する継承グラフ



Public メソッド

- virtual ~cio_Array ()
  デストラクタ
- void * getData (bool extract=false)
  データポインタを取得
- CIO::E_CIO_DTYPE getDataType () const
  データタイプの取得
- char * getDataTypeString () const
  データタイプ文字列の取得
- CIO::E_CIO_ARRAYSHAPE getArrayShape () const
  配列形状の取得
- char * getArrayShapeString () const
  配列形状文字列の取得
- size_t getGc () const
  ガイドセル数を取得
- int getGcInt () const
  ガイドセル数を取得 *(int 版)*
- size_t getNcomp () const
  成分数を取得
- int getNcompInt () const
  成分数を取得 *(int 版)*

- const size_t * getArraySize ()
  格子数を取得
- const int * getArraySizeInt ()
  格子数を取得 *(int* **版***)*
- const int * getHeadIndex ()
  head **インデクスを取得**
- const int * getTailIndex ()
  tail **インデクスを取得**
- const size_t * _getArraySize ()
  ガイドセルを含んだ格子数を取得
- const int * _getArraySizeInt ()
  ガイドセルを含んだ格子数を取得 *(int* **版***)*
- size_t getArrayLength () const
  配列長を取得
- void setHeadIndex (int head[3])
  head/tail **をセット**
- virtual int copyArray (cio_Array *dst, bool ignoreGc=false)=0
  配列コピー *(***自信を** *dst* **にコピー。***head/tail* **を考慮した重複範囲をコピー***)*
- virtual int copyArray (int sta[3], int end[3], cio_Array *dst)=0
  範囲指定での配列コピー *(***自信を** *dst* **にコピー。***head/tail* **を考慮した重複範囲をコピー***)*
- virtual int copyArrayNcomp (cio_Array *dst, int comp, bool ignoreGc=false)=0
  指定成分の配列コピー *(***自信を** *dst* **にコピー。***head/tail* **を考慮した重複範囲をコピー***)*
- virtual int copyArrayNcomp (int sta[3], int end[3], cio_Array *dst, int comp)=0
  指定成分の範囲指定での配列コピー *(***自信を** *dst* **にコピー。***head/tail* **を考慮した重複範囲をコピー***)*
- virtual size_t readBinary (FILE *fp, bool bMatchEndian)=0
  配列サイズ分のバイナリデータを読み込み *(***戻り値は読み込んだ要素数***)*
- virtual size_t writeBinary (FILE *fp)=0
  配列サイズ分のバイナリデータを書き出す *(***戻り値は読み込んだ要素数***)*
- virtual size_t writeAscii (FILE *fp)=0
  配列サイズ分の *ascii* **データを書き出す** *(***戻り値は読み込んだ要素数***)*
- template<class T >
  instanceArray (T *data, CIO::E_CIO_ARRAYSHAPE shape, size_t ix, size_t jx, size_t kx, size_t gc, size_t ncomp)
- template<class T >
  instanceArray (T *data, CIO::E_CIO_ARRAYSHAPE shape, size_t sz[3], size_t gc, size_t ncomp)
- template<class T >
  instanceArray (T *data, CIO::E_CIO_ARRAYSHAPE shape, int ix, int jx, int kx, int gc, int ncomp)
- template<class T >
  instanceArray (T *data, CIO::E_CIO_ARRAYSHAPE shape, int sz[3], int gc, int ncomp)

Static Public メソッド

- static cio_Array * instanceArray (CIO::E_CIO_DTYPE dtype, CIO::E_CIO_ARRAYSHAPE shape, size_t ix, size_t jx, size_t kx, size_t gc, size_t ncomp=1)
  インスタンス
- static cio_Array * instanceArray (CIO::E_CIO_DTYPE dtype, CIO::E_CIO_ARRAYSHAPE shape, size_t sz[3], size_t gc, size_t ncomp=1)
  インスタンス
- static cio_Array * instanceArray (CIO::E_CIO_DTYPE dtype, CIO::E_CIO_ARRAYSHAPE shape, int ix, int jx, int kx, int gc, int ncomp=1)
  インスタンス

- static cio_Array * instanceArray (CIO::E_CIO_DTYPE dtype, CIO::E_CIO_ARRAYSHAPE shape, int sz[3], int gc, int ncomp=1)
  インスタンス
- template<class T >
  static cio_Array * instanceArray (T *data, CIO::E_CIO_ARRAYSHAPE shape, size_t ix, size_t jx, size_t kx, size_t gc, size_t ncomp=1)
  インスタンス
- template<class T >
  static cio_Array * instanceArray (T *data, CIO::E_CIO_ARRAYSHAPE shape, size_t sz[3], size_t gc, size_t ncomp=1)
  インスタンス
- template<class T >
  static cio_Array * instanceArray (T *data, CIO::E_CIO_ARRAYSHAPE shape, int ix, int jx, int kx, int gc, int ncomp=1)
  インスタンス
- template<class T >
  static cio_Array * instanceArray (T *data, CIO::E_CIO_ARRAYSHAPE shape, int sz[3], int gc, int ncomp=1)
  インスタンス
- static cio_Array * interp_coarse (cio_Array *src, int &err, bool head0start=true)
  粗密データの補間処理を行う

Protected メソッド

- cio_Array ()
  デフォルトコンストラクタ
- cio_Array (CIO::E_CIO_DTYPE dtype, CIO::E_CIO_ARRAYSHAPE shape, size_t ix, size_t jx, size_t kx, size_t gc, size_t ncomp=1)
  コンストラクタ

Protected 変数

- CIO::E_CIO_DTYPE m_dtype
  データタイプ
- CIO::E_CIO_ARRAYSHAPE m_shape
  配列形状
- size_t m_gc
  ガイドセル数
- size_t m_sz [3]
  格子数
- size_t m_Sz [4]
  ガイドセルを含んだ格子数
- size_t m_gcl [4]
  ガイドセル数 *(インデクス毎)*
- size_t m_ncomp
  成分数
- int m_gcl
  ガイドセル数 *(int)*
- int m_szl [3]
  格子数 *(int)*
- int m_Szl [4]
  ガイドセルを含んだ格子数 *(int)*

- int m_ncompI
  成分数 *(int)*
- int m_headIndex [4]
  head **インデックス**
- int m_tailIndex [4]
  tail **インデックス**

6.2.1 説明

cio_Array.h の 22 行で定義されています。

6.2.2 コンストラクタとデストラクタ

6.2.2.1 virtual cio_Array::~cio_Array ( ) [inline], [virtual]

デストラクタ

cio_Array.h の 59 行で定義されています。

```
60    {
61    }
```

6.2.2.2 cio_Array::cio_Array ( ) [inline], [protected]

デフォルトコンストラクタ

cio_Array.h の 372 行で定義されています。

参照先 CIO::E_CIO_ARRAYSHAPE_UNKNOWN, CIO::E_CIO_DTYPE_UNKNOWN, m_dtype, m_gc, m_gcI, m_-headIndex, m_ncomp, m_shape, m_sz, m_Sz, と m_tailIndex.

```
373   {
374     m_dtype = CIO::E_CIO_DTYPE_UNKNOWN;
375     m_shape = CIO::E_CIO_ARRAYSHAPE_UNKNOWN;
376     m_sz[0] = m_sz[1] = m_sz[2] = 0;
377     m_Sz[0] = m_Sz[1] = m_Sz[2] = m_Sz[3] = 0;
378     m_gc = 0;
379     m_gcI[0] = m_gcI[1] = m_gcI[2] = m_gcI[3] = 0;
380     m_ncomp = 1;
381     m_headIndex[0] = m_headIndex[1] = m_headIndex[2] = m_headIndex[3] = 0;
382     m_tailIndex[0] = m_tailIndex[1] = m_tailIndex[2] = m_tailIndex[3] = 0;
383   }
```

6.2.2.3 cio_Array::cio_Array (CIO::E_CIO_DTYPE *dtype*, CIO::E_CIO_ARRAYSHAPE *shape*, size_t *ix*, size_t *jx*, size_t *kx*, size_t *gc*, size_t *ncomp* = 1) [inline], [protected]

コンストラクタ

cio_Array.h の 386 行で定義されています。

参照先 CIO::E_CIO_IJKN, CIO::E_CIO_NIJK, m_dtype, m_gc, m_gcI, m_gcI, m_ncomp, m_ncompI, m_shape, m_sz, m_Sz, m_szI, m_SzI, と setHeadIndex().

```
393   {
394     m_sz[0] = m_szI[0] = ix;
395     m_sz[1] = m_szI[1] = jx;
396     m_sz[2] = m_szI[2] = kx;
397
398     switch(shape)
399     {
400     case CIO::E_CIO_IJKN:
401       m_Sz[0] = m_SzI[0] = ix+2*gc;
```

```
402       m_Sz[1] = m_SzI[1] = jx+2*gc;
403       m_Sz[2] = m_SzI[2] = kx+2*gc;
404       m_Sz[3] = m_SzI[3] = ncomp;
405       m_gcl[0] = gc;
406       m_gcl[1] = gc;
407       m_gcl[2] = gc;
408       m_gcl[3] = 0;
409       break;
410     case CIO::E_CIO_NIJK:
411       m_Sz[0] = m_SzI[0] = ncomp;
412       m_Sz[1] = m_SzI[1] = ix+2*gc;
413       m_Sz[2] = m_SzI[2] = jx+2*gc;
414       m_Sz[3] = m_SzI[3] = kx+2*gc;
415       m_gcl[0] = 0;
416       m_gcl[1] = gc;
417       m_gcl[2] = gc;
418       m_gcl[3] = gc;
419     }
420
421     m_gc = m_gcI = gc;
422     m_ncomp = m_ncompI = ncomp;
423     m_dtype = dtype;
424     m_shape = shape;
425
426     int head[3]={0,0,0};
427     setHeadIndex(head);
428   }
```

6.2.3 関数

6.2.3.1 const size_t* cio_Array::_getArraySize ( ) [inline]

ガイドセルを含んだ格子数を取得

cio_Array.h の 275 行で定義されています。

参照先 CIO::E_CIO_IJKN, CIO::E_CIO_NIJK, m_shape, と m_Sz.

```
276   {
277     switch(m_shape)
278     {
279     case CIO::E_CIO_IJKN:
280       return m_Sz;
281       break;
282     case CIO::E_CIO_NIJK:
283       return m_Sz + 1;
284       break;
285     }
286     return NULL;
287   }
```

6.2.3.2 const int* cio_Array::_getArraySizeInt ( ) [inline]

ガイドセルを含んだ格子数を取得 (int 版)

cio_Array.h の 290 行で定義されています。

参照先 CIO::E_CIO_IJKN, CIO::E_CIO_NIJK, m_shape, と m_SzI.

```
291   {
292     switch(m_shape)
293     {
294     case CIO::E_CIO_IJKN:
295       return m_SzI;
296       break;
297     case CIO::E_CIO_NIJK:
298       return m_SzI + 1;
299       break;
300     }
301     return NULL;
302   }
```

6.2.3.3 virtual int cio_Array::copyArray (cio_Array * *dst*, bool *ignoreGc* = false) [pure virtual]

配列コピー (自信を dst にコピー。head/tail を考慮した重複範囲をコピー)

cio_TypeArray< T >で実装されています。

参照元 cio_DFI_BOV::read_Datarecord(), cio_DFI_SPH::read_Datarecord(), cio_DFI::ReadData(), と cio_DFI::WriteData().

6.2.3.4 virtual int cio_Array::copyArray (int *sta[3]*, int *end[3]*, cio_Array * *dst*) [pure virtual]

範囲指定での配列コピー (自信を dst にコピー。head/tail を考慮した重複範囲をコピー)

cio_TypeArray< T >で実装されています。

6.2.3.5 virtual int cio_Array::copyArrayNcomp (cio_Array * *dst*, int *comp*, bool *ignoreGc* = false) [pure virtual]

指定成分の配列コピー (自信を dst にコピー。head/tail を考慮した重複範囲をコピー)

cio_TypeArray< T >で実装されています。

参照元 cio_DFI_BOV::read_Datarecord().

6.2.3.6 virtual int cio_Array::copyArrayNcomp (int *sta[3]*, int *end[3]*, cio_Array * *dst*, int *comp*) [pure virtual]

指定成分の範囲指定での配列コピー (自信を dst にコピー。head/tail を考慮した重複範囲をコピー)

cio_TypeArray< T >で実装されています。

6.2.3.7 size_t cio_Array::getArrayLength ( ) const [inline]

配列長を取得

cio_Array.h の 305 行で定義されています。

参照先 m_Sz.

参照元 cio_DFI_BOV::read_Datarecord(), と cio_DFI_SPH::read_Datarecord().

```
306   {
307     size_t nw = 1;
308     for( int i=0;i<4;i++ )
309     {
310       nw *= m_Sz[i];
311     }
312     return nw;
313   }
```

6.2.3.8 CIO::E_CIO_ARRAYSHAPE cio_Array::getArrayShape ( ) const [inline]

配列形状の取得

cio_Array.h の 188 行で定義されています。

参照先 m_shape.

参照元 cio_TypeArray< T >::copyArray(), cio_TypeArray< T >::copyArrayNcomp(), cio_DFI_BOV::read_Datarecord(), cio_DFI::setGridData(), cio_DFI::VolumeDataDivide(), cio_DFI_PLOT3D::write_Func(), と cio_DFI::WriteFieldData().

```
189   {
190     return m_shape;
191   }
```

6.2.3.9 char* cio_Array::getArrayShapeString ( ) const [inline]

配列形状文字列の取得

cio_Array.h の 194 行で定義されています。

参照先 CIO::E_CIO_IJKN, CIO::E_CIO_NIJK, と m_shape.

```
195   {
196     switch(m_shape)
197     {
198     case CIO::E_CIO_IJKN:
199       return "IJKN";
200       break;
201     case CIO::E_CIO_NIJK:
202       return "NIJK";
203       break;
204     }
205     return "Unknown";
206   }
```

6.2.3.10 const size_t* cio_Array::getArraySize ( ) [inline]

格子数を取得

cio_Array.h の 233 行で定義されています。

参照先 m_sz.

```
234   {
235     return m_sz;
236   }
```

6.2.3.11 const int* cio_Array::getArraySizeInt ( ) [inline]

格子数を取得 (int 版)

cio_Array.h の 239 行で定義されています。

参照先 m_szI.

参照元 cio_DFI::setGridData(), cio_DFI::VolumeDataDivide(), cio_DFI_VTK::write_DataRecord(), cio_DFI_AVS-::write_DataRecord(), cio_DFI_PLOT3D::write_DataRecord(), と cio_DFI::WriteFieldData().

```
240   {
241     return m_szI;
242   }
```

6.2.3.12 cio_Array::getData (bool *extract* = false)

データポインタを取得

cio_Array_inline.h の 244 行で定義されています。

参照先 CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_INT16, CIO::E_CIO_INT32, CIO::E_CIO_I-NT64, CIO::E_CIO_INT8, CIO::E_CIO_UINT16, CIO::E_CIO_UINT32, CIO::E_CIO_UINT64, CIO::E_CIO_UINT8, と cio_TypeArray< T >::getData().

参照元 interp_coarse(), cio_DFI::ReadData(), と cio_DFI_VTK::write_DataRecord().

```
245 {
246   switch( m_dtype )
247   {
248   case CIO::E_CIO_INT8:
249     {
250       cio_TypeArray<char> *ptr = dynamic_cast<cio_TypeArray<char>*>(this);
```

```
251       return ptr->getData( extract );
252     }
253     break;
254   case CIO::E_CIO_INT16:
255     {
256       cio_TypeArray<short> *ptr = dynamic_cast<cio_TypeArray<short>*>(this);
257       return ptr->getData( extract );
258     }
259     break;
260   case CIO::E_CIO_INT32:
261     {
262       cio_TypeArray<int> *ptr = dynamic_cast<cio_TypeArray<int>*>(this);
263       return ptr->getData( extract );
264     }
265     break;
266   case CIO::E_CIO_INT64:
267     {
268       cio_TypeArray<long long> *ptr = dynamic_cast<cio_TypeArray<long long>*>(this);
269       return ptr->getData( extract );
270     }
271     break;
272   case CIO::E_CIO_UINT8:
273     {
274       cio_TypeArray<unsigned char> *ptr = dynamic_cast<
    cio_TypeArray<unsigned char>*>(this);
275       return ptr->getData( extract );
276     }
277     break;
278   case CIO::E_CIO_UINT16:
279     {
280       cio_TypeArray<unsigned short> *ptr = dynamic_cast<
    cio_TypeArray<unsigned short>*>(this);
281       return ptr->getData( extract );
282     }
283     break;
284   case CIO::E_CIO_UINT32:
285     {
286       cio_TypeArray<unsigned int> *ptr = dynamic_cast<
    cio_TypeArray<unsigned int>*>(this);
287       return ptr->getData( extract );
288     }
289     break;
290   case CIO::E_CIO_UINT64:
291     {
292       cio_TypeArray<unsigned long long> *ptr = dynamic_cast<
    cio_TypeArray<unsigned long long>*>(this);
293       return ptr->getData( extract );
294     }
295     break;
296   case CIO::E_CIO_FLOAT32:
297     {
298       cio_TypeArray<float> *ptr = dynamic_cast<cio_TypeArray<float>*>(this);
299       return ptr->getData( extract );
300     }
301     break;
302   case CIO::E_CIO_FLOAT64:
303     {
304       cio_TypeArray<double> *ptr = dynamic_cast<cio_TypeArray<double>*>(this);
305       return ptr->getData( extract );
306     }
307     break;
308   }
309
310   return NULL;
311 }
```

6.2.3.13 CIO::E_CIO_DTYPE cio_Array::getDataType ( ) const [inline]

データタイプの取得

cio_Array.h の 143 行で定義されています。

参照先 m_dtype.

参照元 cio_TypeArray< T >::copyArray(), cio_TypeArray< T >::copyArrayNcomp(), interp_coarse(), cio_DFI_VTK::write_DataRecord(), cio_DFI_PLOT3D::write_DataRecord(), と cio_DFI::WriteFieldData().

```
144   {
145     return m_dtype;
146   }
```

6.2.3.14 char* cio_Array::getDataTypeString ( ) const [inline]

データタイプ文字列の取得

cio_Array.h の 149 行で定義されています。

参照先 CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_INT16, CIO::E_CIO_INT32, CIO::E_CIO_INT64, CIO::E_CIO_INT8, CIO::E_CIO_UINT16, CIO::E_CIO_UINT32, CIO::E_CIO_UINT64, CIO::E_CIO_UINT8, と m_dtype.

```
150   {
151     switch( m_dtype )
152     {
153     case CIO::E_CIO_INT8:
154       return "INT8";
155       break;
156     case CIO::E_CIO_INT16:
157       return "INT16";
158       break;
159     case CIO::E_CIO_INT32:
160       return "INT32";
161       break;
162     case CIO::E_CIO_INT64:
163       return "INT64";
164       break;
165     case CIO::E_CIO_UINT8:
166       return "UINT8";
167       break;
168     case CIO::E_CIO_UINT16:
169       return "UINT16";
170       break;
171     case CIO::E_CIO_UINT32:
172       return "UINT32";
173       break;
174     case CIO::E_CIO_UINT64:
175       return "UINT64";
176       break;
177     case CIO::E_CIO_FLOAT32:
178       return "FLOAT32";
179       break;
180     case CIO::E_CIO_FLOAT64:
181       return "FLOAT64";
182       break;
183     }
184     return "Unknown";
185   }
```

6.2.3.15 size_t cio_Array::getGc ( ) const [inline]

ガイドセル数を取得

cio_Array.h の 209 行で定義されています。

参照先 m_gc.

```
210   {
211     return m_gc;
212   }
```

6.2.3.16 int cio_Array::getGcInt ( ) const [inline]

ガイドセル数を取得 (int 版)

cio_Array.h の 215 行で定義されています。

参照先 m_gcI.

参照元 cio_TypeArray< T >::copyArray(), と cio_TypeArray< T >::copyArrayNcomp().

```
216   {
217     return m_gcI;
218   }
```

6.2.3.17 const int* cio_Array::getHeadIndex ( ) [inline]

head インデクスを取得

cio_Array.h の 245 行で定義されています。

参照先 CIO::E_CIO_IJKN, CIO::E_CIO_NIJK, m_headIndex, と m_shape.

参照元 cio_TypeArray< T >::copyArray(), と cio_TypeArray< T >::copyArrayNcomp().

```
246   {
247     switch(m_shape)
248     {
249     case CIO::E_CIO_IJKN:
250       return m_headIndex;
251       break;
252     case CIO::E_CIO_NIJK:
253       return m_headIndex + 1;
254       break;
255     }
256     return NULL;
257   }
```

6.2.3.18 size_t cio_Array::getNcomp ( ) const [inline]

成分数を取得

cio_Array.h の 221 行で定義されています。

参照先 m_ncomp.

参照元 cio_TypeArray< T >::copyArray(), cio_TypeArray< T >::copyArrayNcomp(), cio_DFI_BOV::read_Datarecord(), cio_DFI_VTK::write_DataRecord(), cio_DFI_PLOT3D::write_DataRecord(), と cio_DFI::WriteFieldData().

```
222   {
223     return m_ncomp;
224   }
```

6.2.3.19 int cio_Array::getNcompInt ( ) const [inline]

成分数を取得 (int 版)

cio_Array.h の 227 行で定義されています。

参照先 m_ncompI.

参照元 cio_DFI::setGridData(), と cio_DFI::VolumeDataDivide().

```
228   {
229     return m_ncompI;
230   }
```

6.2.3.20 const int* cio_Array::getTailIndex ( ) [inline]

tail インデクスを取得

cio_Array.h の 260 行で定義されています。

参照先 CIO::E_CIO_IJKN, CIO::E_CIO_NIJK, m_shape, と m_tailIndex.

参照元 cio_TypeArray< T >::copyArray(), と cio_TypeArray< T >::copyArrayNcomp().

```
261   {
262     switch(m_shape)
263     {
```

```
264     case CIO::E_CIO_IJKN:
265       return m_tailIndex;
266       break;
267     case CIO::E_CIO_NIJK:
268       return m_tailIndex + 1;
269       break;
270     }
271     return NULL;
272   }
```

6.2.3.21 cio_Array::instanceArray (CIO::E_CIO_DTYPE *dtype*, CIO::E_CIO_ARRAYSHAPE *shape*, size_t *ix*, size_t *jx*, size_t *kx*, size_t *gc*, size_t *ncomp* = 1) [static]

インスタンス

cio_Array_inline.h の 30 行で定義されています。

参照先 CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_INT16, CIO::E_CIO_INT32, CIO::E_CIO_INT64, CIO::E_CIO_INT8, CIO::E_CIO_UINT16, CIO::E_CIO_UINT32, CIO::E_CIO_UINT64, CIO::E_CIO_UINT8, m_gcl, と m_Sz.

参照元 interp_coarse(), cio_DFI::ReadData(), cio_DFI::ReadFieldData(), cio_DFI::WriteData(), と cio_DFI::WriteFieldData().

```
37 {
38   cio_Array *ptr = NULL;
39   switch( dtype )
40   {
41   case CIO::E_CIO_INT8:
42     ptr = new cio_TypeArray<char>(dtype,shape,ix,jx,kx,gc,ncomp);
43     break;
44   case CIO::E_CIO_INT16:
45     ptr = new cio_TypeArray<short>(dtype,shape,ix,jx,kx,gc,ncomp);
46     break;
47   case CIO::E_CIO_INT32:
48     ptr = new cio_TypeArray<int>(dtype,shape,ix,jx,kx,gc,ncomp);
49     break;
50   case CIO::E_CIO_INT64:
51     ptr = new cio_TypeArray<long long>(dtype,shape,ix,jx,kx,gc,ncomp);
52     break;
53   case CIO::E_CIO_UINT8:
54     ptr = new cio_TypeArray<unsigned char>(dtype,shape,ix,jx,kx,gc,ncomp);
55     break;
56   case CIO::E_CIO_UINT16:
57     ptr = new cio_TypeArray<unsigned short>(dtype,shape,ix,jx,kx,gc,ncomp);
58     break;
59   case CIO::E_CIO_UINT32:
60     ptr = new cio_TypeArray<unsigned int>(dtype,shape,ix,jx,kx,gc,ncomp);
61     break;
62   case CIO::E_CIO_UINT64:
63     ptr = new cio_TypeArray<unsigned long long>(dtype,shape,ix,jx,kx,gc,ncomp);
64     break;
65   case CIO::E_CIO_FLOAT32:
66     ptr = new cio_TypeArray<float>(dtype,shape,ix,jx,kx,gc,ncomp);
67     break;
68   case CIO::E_CIO_FLOAT64:
69     ptr = new cio_TypeArray<double>(dtype,shape,ix,jx,kx,gc,ncomp);
70     break;
71   }
72
73 #ifdef _CIO_DEBUG
74   if( ptr )
75   {
76     printf("dtype = %d\n",(int)dtype);
77     printf("shape = %d\n",(int)shape);
78     printf("ixjxkx = %d %d %d\n",(int)ix,(int)jx,(int)kx);
79     printf("gc = %d\n",(int)gc);
80     printf("ncomp = %d\n",(int)ncomp);
81     size_t *m_Sz=ptr->m_Sz;
82     size_t *m_gcl=ptr->m_gcl;
83     printf("Sz  = %d %d %d %d\n",(int)m_Sz[0],(int)m_Sz[1],(int)m_Sz[2],(int)m_Sz[3]);
84     printf("gcl = %d %d %d %d\n",(int)m_gcl[0],(int)m_gcl[1],(int)m_gcl[2],(int)m_gcl[3]);
85   }
86 #endif
87
88   return ptr;
89 }
```

6.2.3.22 cio_Array::instanceArray (CIO::E_CIO_DTYPE *dtype*, CIO::E_CIO_ARRAYSHAPE *shape*, size_t *sz[3]*, size_t *gc*, size_t *ncomp* = 1) [static]

インスタンス

cio_Array_inline.h の 93 行で定義されています。

```
98 {
99   return instanceArray(dtype,shape,sz[0],sz[1],sz[2],gc,ncomp);
100 }
```

6.2.3.23 cio_Array::instanceArray (CIO::E_CIO_DTYPE *dtype*, CIO::E_CIO_ARRAYSHAPE *shape*, int *ix*, int *jx*, int *kx*, int *gc*, int *ncomp* = 1) [static]

インスタンス

cio_Array_inline.h の 104 行で定義されています。

```
111 {
112   return instanceArray(dtype,shape,size_t(ix),size_t(jx),size_t(kx),size_t(gc),size_t(ncomp));
113 }
```

6.2.3.24 cio_Array::instanceArray (CIO::E_CIO_DTYPE *dtype*, CIO::E_CIO_ARRAYSHAPE *shape*, int *sz[3]*, int *gc*, int *ncomp* = 1) [static]

インスタンス

cio_Array_inline.h の 117 行で定義されています。

```
122 {
123   return instanceArray(dtype,shape,size_t(sz[0]),size_t(sz[1]),size_t(sz[2]),size_t(gc),size_t(ncomp));
124 }
```

6.2.3.25 template<class T > static cio_Array* cio_Array::instanceArray (T * *data*, CIO::E_CIO_ARRAYSHAPE *shape*, size_t *ix*, size_t *jx*, size_t *kx*, size_t *gc*, size_t *ncomp* = 1) [static]

インスタンス

6.2.3.26 template<class T > static cio_Array* cio_Array::instanceArray (T * *data*, CIO::E_CIO_ARRAYSHAPE *shape*, size_t *sz[3]*, size_t *gc*, size_t *ncomp* = 1) [static]

インスタンス

6.2.3.27 template<class T > static cio_Array* cio_Array::instanceArray (T * *data*, CIO::E_CIO_ARRAYSHAPE *shape*, int *ix*, int *jx*, int *kx*, int *gc*, int *ncomp* = 1) [static]

インスタンス

6.2.3.28 template<class T > cio_Array::instanceArray (T * *data*, CIO::E_CIO_ARRAYSHAPE *shape*, size_t *ix*, size_t *jx*, size_t *kx*, size_t *gc*, size_t *ncomp*)

cio_Array_inline.h の 129 行で定義されています。

参照先 CIO::E_CIO_DTYPE_UNKNOWN, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_INT16, CIO::E_CIO_INT32, CIO::E_CIO_INT64, CIO::E_CIO_INT8, CIO::E_CIO_UINT16, CIO::E_CIO_UINT32, CIO::E_CIO_UINT64, CIO::E_CIO_UINT8, m_gcl, と m_Sz.

```
136 {
137   cio_Array *ptr = NULL;
138   CIO::E_CIO_DTYPE dtype = CIO::E_CIO_DTYPE_UNKNOWN;
139 
140   if( typeid(data) == typeid(char*) )
141   {
142     dtype = CIO::E_CIO_INT8;
143   }
144   else if( typeid(data) == typeid(short*) )
145   {
146     dtype = CIO::E_CIO_INT16;
147   }
148   else if( typeid(data) == typeid(int*) )
149   {
150     dtype = CIO::E_CIO_INT32;
151   }
152   else if( typeid(data) == typeid(long long*) )
153   {
154     dtype = CIO::E_CIO_INT64;
155   }
156   else if( typeid(data) == typeid(unsigned char*) )
157   {
158     dtype = CIO::E_CIO_UINT8;
159   }
160   else if( typeid(data) == typeid(unsigned short*) )
161   {
162     dtype = CIO::E_CIO_UINT16;
163   }
164   else if( typeid(data) == typeid(unsigned int*) )
165   {
166     dtype = CIO::E_CIO_UINT32;
167   }
168   else if( typeid(data) == typeid(unsigned long long*) )
169   {
170     dtype = CIO::E_CIO_UINT64;
171   }
172   else if( typeid(data) == typeid(float*) )
173   {
174     dtype = CIO::E_CIO_FLOAT32;
175   }
176   else if( typeid(data) == typeid(double*) )
177   {
178     dtype = CIO::E_CIO_FLOAT64;
179   }
180 
181   if( dtype != CIO::E_CIO_DTYPE_UNKNOWN )
182   {
183     ptr = new cio_TypeArray<T>(data,dtype,shape,ix,jx,kx,gc,ncomp);
184   }
185 
186 #ifdef _CIO_DEBUG
187   if( ptr )
188   {
189     printf("dtype = %d\n",(int)dtype);
190     printf("shape = %d\n",(int)shape);
191     printf("ixjxkx = %d %d %d\n",(int)ix,(int)jx,(int)kx);
192     printf("gc = %d\n",(int)gc);
193     printf("ncomp = %d\n",(int)ncomp);
194     size_t *m_Sz=ptr->m_Sz;
195     size_t *m_gcl=ptr->m_gcl;
196     printf("Sz  = %d %d %d %d\n",(int)m_Sz[0],(int)m_Sz[1],(int)m_Sz[2],(int)m_Sz[3]);
197     printf("gcl = %d %d %d %d\n",(int)m_gcl[0],(int)m_gcl[1],(int)m_gcl[2],(int)m_gcl[3]);
198   }
199 #endif
200 
201   return ptr;
202 }
```

6.2.3.29 template<class T > static cio_Array* cio_Array::instanceArray (T * *data*, CIO::E_CIO_ARRAYSHAPE *shape*, int *sz[3]*, int *gc*, int *ncomp* = 1) [static]

インスタンス

6.2.3.30 template<class T > cio_Array::instanceArray (T * *data*, CIO::E_CIO_ARRAYSHAPE *shape*, size_t *sz[3]*, size_t *gc*, size_t *ncomp*)

cio_Array_inline.h の 207 行で定義されています。

```
212 {
```

```
213   return instanceArray(data,shape,sz[0],sz[1],sz[2],gc,ncomp);
214 }
```

6.2.3.31 template<class T > cio_Array::instanceArray (T * *data*, CIO::E_CIO_ARRAYSHAPE *shape*, int *ix*, int *jx*, int *kx*, int *gc*, int *ncomp*)

cio_Array_inline.h の 219 行で定義されています。

```
226 {
227   return instanceArray(data,shape,size_t(ix),size_t(jx),size_t(kx),size_t(gc),size_t(ncomp));
228 }
```

6.2.3.32 template<class T > cio_Array::instanceArray (T * *data*, CIO::E_CIO_ARRAYSHAPE *shape*, int *sz[3]*, int *gc*, int *ncomp*)

cio_Array_inline.h の 233 行で定義されています。

```
238 {
239   return instanceArray(data,shape,size_t(sz[0]),size_t(sz[1]),size_t(sz[2]),size_t(gc),size_t(ncomp));
240 }
```

6.2.3.33 cio_Array::interp_coarse (cio_Array * *src*, int & *err*, bool *head0start* = `true`) [static]

粗密データの補間処理を行う

cio_Array_inline.h の 580 行で定義されています。

参照先 cio_interp_ijkn_r4_(), cio_interp_ijkn_r8_(), cio_interp_nijk_r4_(), cio_interp_nijk_r8_(), CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_IJKN, getData(), getDataType(), instanceArray(), と setHeadIndex().

参照元 cio_DFI::ReadData().

```
581 {
582   err = 1;
583
584   // データタイプ
585   // 実数型のみ対応
586   CIO::E_CIO_DTYPE dtype = src->getDataType();
587   if( dtype != CIO::E_CIO_FLOAT32 && dtype != CIO::E_CIO_FLOAT64 )
588   {
589     err = -1;
590     return NULL;
591   }
592
593   // 配列形状
594   CIO::E_CIO_ARRAYSHAPE shape = src->getArrayShape();
595
596   // 成分数
597   // 成分数は 1 か 3 のみ対応
598   int ncomp = src->getNcomp();
599   if( ncomp != 1 && ncomp != 3 )
600   {
601     err = -1;
602     return NULL;
603   }
604
605   // その他の情報の取得
606   int gcS = src->getGc();
607   void *ptrS = src->getData();
608   const int *szS = src->getArraySizeInt();
609   const int *headS = src->getHeadIndex();
610   const int *tailS = src->getTailIndex();
611
612   // 密配列のインスタンス
613   int gcD = gcS*2;
614   int szD[3] = {szS[0]*2, szS[1]*2, szS[2]*2};
615   cio_Array *dst = cio_Array::instanceArray( dtype, shape, szD, gcD, ncomp );
616   void *ptrD = dst->getData();
617
```

```
618   // head インデクスのセット
619   int headD[3];
620   for( int i=0;i<3;i++ )
621   {
622     headD[i] = headS[i]*2;
623     if( !head0start )
624     {
625       headD[i] -= 1;
626     }
627   }
628   dst->setHeadIndex( headD );
629
630   // f90 コードのコール (配列形状、実数型毎)
631   if( shape == CIO::E_CIO_IJKN )
632   {
633     if( dtype == CIO::E_CIO_FLOAT32 )
634     {
635       cio_interp_ijkn_r4_(szS,&gcS,szD,&gcD,&ncomp,(float*)ptrS,(float*)ptrD);
636     }
637     else
638     {
639       cio_interp_ijkn_r8_(szS,&gcS,szD,&gcD,&ncomp,(double*)ptrS,(double*)ptrD);
640     }
641   }
642   else
643   {
644     if( dtype == CIO::E_CIO_FLOAT32 )
645     {
646       cio_interp_nijk_r4_(szS,&gcS,szD,&gcD,&ncomp,(float*)ptrS,(float*)ptrD);
647     }
648     else
649     {
650       cio_interp_nijk_r8_(szS,&gcS,szD,&gcD,&ncomp,(double*)ptrS,(double*)ptrD);
651     }
652   }
653
654   return dst;
655 }
```

6.2.3.34 virtual size_t cio_Array::readBinary (FILE * *fp*, bool *bMatchEndian*) [pure virtual]

配列サイズ分のバイナリデータを読み込み (戻り値は読み込んだ要素数)

cio_TypeArray< T >で実装されています。

参照元 cio_DFI_BOV::read_Datarecord(), と cio_DFI_SPH::read_Datarecord().

6.2.3.35 void cio_Array::setHeadIndex (int *head[3]*) [inline]

head/tail をセット

cio_Array.h の 316 行で定義されています。

参照先 CIO::E_CIO_IJKN, CIO::E_CIO_NIJK, m_headIndex, m_shape, m_sz, と m_tailIndex.

参照元 cio_Array(), interp_coarse(), cio_DFI_BOV::read_Datarecord(), cio_DFI_SPH::read_Datarecord(), cio_DFI::ReadData(), と cio_DFI::ReadFieldData().

```
317   {
318     switch(m_shape)
319     {
320     case CIO::E_CIO_IJKN:
321       m_headIndex[0] = head[0];
322       m_headIndex[1] = head[1];
323       m_headIndex[2] = head[2];
324       m_headIndex[3] = 0;
325       m_tailIndex[0] = m_headIndex[0] + m_sz[0] - 1;
326       m_tailIndex[1] = m_headIndex[1] + m_sz[1] - 1;
327       m_tailIndex[2] = m_headIndex[2] + m_sz[2] - 1;
328       m_tailIndex[3] = 0;
329       break;
330     case CIO::E_CIO_NIJK:
331       m_headIndex[0] = 0;
332       m_headIndex[1] = head[0];
333       m_headIndex[2] = head[1];
334       m_headIndex[3] = head[2];
335       m_tailIndex[0] = 0;
```

```
336       m_tailIndex[1] = m_headIndex[1] + m_sz[0] - 1;
337       m_tailIndex[2] = m_headIndex[2] + m_sz[1] - 1;
338       m_tailIndex[3] = m_headIndex[3] + m_sz[2] - 1;
339     }
340   }
```

6.2.3.36 virtual size_t cio_Array::writeAscii (FILE * *fp*) [pure virtual]

配列サイズ分の ascii データを書き出す (戻り値は読み込んだ要素数)

cio_TypeArray< T >で実装されています。

参照元 cio_DFI_VTK::write_DataRecord().

6.2.3.37 virtual size_t cio_Array::writeBinary (FILE * *fp*) [pure virtual]

配列サイズ分のバイナリデータを書き出す (戻り値は読み込んだ要素数)

cio_TypeArray< T >で実装されています。

参照元 cio_DFI_BOV::write_DataRecord(), cio_DFI_AVS::write_DataRecord(), と cio_DFI_SPH::write_DataRecord().

6.2.4 変数

6.2.4.1 CIO::E_CIO_DTYPE cio_Array::m_dtype [protected]

データタイプ

cio_Array.h の 438 行で定義されています。

参照元 cio_Array(), getDataType(), と getDataTypeString().

6.2.4.2 size_t cio_Array::m_gc [protected]

ガイドセル数

cio_Array.h の 444 行で定義されています。

参照元 cio_Array(), cio_TypeArray< T >::cio_TypeArray(), と getGc().

6.2.4.3 int cio_Array::m_gcl [protected]

ガイドセル数 (int)

cio_Array.h の 460 行で定義されています。

参照元 cio_Array(), と getGcInt().

6.2.4.4 size_t cio_Array::m_gcl[4] [protected]

ガイドセル数 (インデクス毎)

cio_Array.h の 453 行で定義されています。

参照元 cio_Array(), と instanceArray().

6.2.4.5 int cio_Array::m_headIndex[4] [protected]

head インデックス

cio_Array.h の 473 行で定義されています。

参照元 cio_Array(), getHeadIndex(), と setHeadIndex().

6.2.4.6 size_t cio_Array::m_ncomp [protected]

成分数

cio_Array.h の 456 行で定義されています。

参照元 cio_Array(), cio_TypeArray< T >::cio_TypeArray(), と getNcomp().

6.2.4.7 int cio_Array::m_ncompI [protected]

成分数 (int)

cio_Array.h の 469 行で定義されています。

参照元 cio_Array(), と getNcompInt().

6.2.4.8 CIO::E_CIO_ARRAYSHAPE cio_Array::m_shape [protected]

配列形状

cio_Array.h の 441 行で定義されています。

参照元 _getArraySize(), _getArraySizeInt(), cio_Array(), getArrayShape(), getArrayShapeString(), getHeadIndex(), getTailIndex(), と setHeadIndex().

6.2.4.9 size_t cio_Array::m_sz[3] [protected]

格子数

cio_Array.h の 447 行で定義されています。

参照元 cio_Array(), cio_TypeArray< T >::cio_TypeArray(), getArraySize(), と setHeadIndex().

6.2.4.10 size_t cio_Array::m_Sz[4] [protected]

ガイドセルを含んだ格子数

cio_Array.h の 450 行で定義されています。

参照元 _getArraySize(), cio_Array(), getArrayLength(), と instanceArray().

6.2.4.11 int cio_Array::m_szI[3] [protected]

格子数 (int)

cio_Array.h の 463 行で定義されています。

参照元 cio_Array(), と getArraySizeInt().

6.2.4.12 int cio_Array::m_SzI[4] [protected]

ガイドセルを含んだ格子数 (int)

cio_Array.h の 466 行で定義されています。

参照元 _getArraySizeInt(), と cio_Array().

6.2.4.13 int cio_Array::m_tailIndex[4] [protected]

tail インデックス

cio_Array.h の 476 行で定義されています。

参照元 cio_Array(), getTailIndex(), と setHeadIndex().

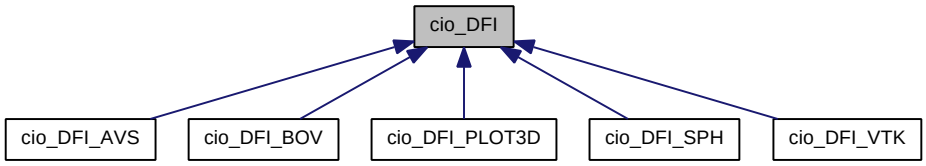
このクラスの説明は次のファイルから生成されました:

- cio_Array.h
- cio_Array_inline.h

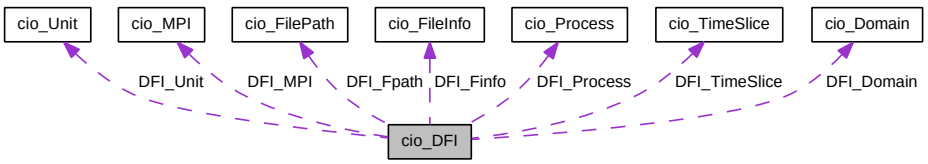
6.3 クラス cio_DFI

`#include <cio_DFI.h>`

cio_DFI に対する継承グラフ



cio_DFI のコラボレーション図



Public メソッド

- cio_DFI ()
- ~cio_DFI ()
- const cio_FileInfo * GetcioFileInfo ()
  cioFileInfo **クラスのポインタを取得**
- const cio_FilePath * GetcioFilePath ()
  cio_FilePath **クラスのポインタを取得**
- void SetcioFilePath (cio_FilePath FPath)
  cio_FilePath **クラスのセット**
- const cio_Unit * GetcioUnit ()
  cio_Unit **クラスのポインタを取得**
- void SetcioUnit (cio_Unit unit)
  cio_Unit **クラスのセット**
- const cio_Domain * GetcioDomain ()
  cio_Domain **クラスのポインタ取得**

- void SetcioDomain (cio_Domain domain)

  cio_Domain **クラスのセット**
- const cio_MPI * GetcioMPI ()

  cio_MPI **クラスのポインタ取得**
- void SetcioMPI (cio_MPI mpi)

  cio_MPI **クラスセット**
- const cio_TimeSlice * GetcioTimeSlice ()

  cio_TimeSlice **クラスのポインタ取得**
- void SetcioTimeSlice (cio_TimeSlice TSlice)

  cio_TimeSlice **クラスセット**
- const cio_Process * GetcioProcess ()

  cio_Process **クラスのポインタ取得**
- void SetcioProcess (cio_Process Process)

  cio_Process **クラスセット**
- std::string Generate_FieldFileName (int RankID, int step, const bool mio)

  フィールドデータ（*SPH,BOV***) ファイル名の作成** *(***ディレクトリパスが付加されている**)
- void set_RankID (const int rankID)

  RankID **をセットする**
- void set_output_type (CIO::E_CIO_OUTPUT_TYPE output_type)

  出力形式 *(ascii,binary,FortranBinary)* **をセット**
- void set_output_fname (CIO::E_CIO_OUTPUT_FNAME output_fname)

  出力ファイル命名規約 *(step_rank,rank_step)* **をセット**
- std::string get_dfi_fname ()
- template<class TimeT , class TimeAvrT >
  void * ReadData (CIO::E_CIO_ERRORCODE &ret, const unsigned step, const int gc, const int Gvoxel[3], const int Gdivision[3], const int head[3], const int tail[3], TimeT &time, const bool mode, unsigned &step_avr, TimeAvrT &time_avr)

  read field data record (template function)
- template<class T , class TimeT , class TimeAvrT >
  CIO::E_CIO_ERRORCODE ReadData (T *val, const unsigned step, const int gc, const int Gvoxel[3], const int Gdivision[3], const int head[3], const int tail[3], TimeT &time, const bool mode, unsigned &step_avr, TimeAvrT &time_avr)

  read field data record (template function)
- CIO::E_CIO_ERRORCODE ReadData (cio_Array *val, const unsigned step, const int gc, const int Gvoxel[3], const int Gdivision[3], const int head[3], const int tail[3], double &time, const bool mode, unsigned &step_avr, double &time_avr)

  read field data record
- template<class T , class TimeT , class TimeAvrT >
  CIO::E_CIO_ERRORCODE WriteData (const unsigned step, TimeT time, const int sz[3], const int nComp, const int gc, T *val, T *minmax=NULL, bool avr_mode=true, unsigned step_avr=0, TimeAvrT time_avr=0.0)

  write field data record (template function)
- CIO::E_CIO_ERRORCODE WriteData (const unsigned step, const int gc, double time, cio_Array *val, double *minmax, const bool avr_mode, const unsigned step_avr, double time_avr)

  write field data record
- CIO::E_CIO_ERRORCODE WriteProcDfiFile (const MPI_Comm comm, bool out_host=false)

  proc DFI **ファイル出力コントロール** *(float)*
- std::string GetArrayShapeString ()

  配列形状を文字列で返す
- CIO::E_CIO_ARRAYSHAPE GetArrayShape ()

  配列形状を返す
- std::string GetDataTypeString ()

  get DataType **（データタイプの取り出し関数）**

- CIO::E_CIO_DTYPE GetDataType ()

  get DataType **（データタイプの取り出し関数）**
- int GetNumComponent ()

  get Number of Component **（成分数の取り出し関数）**
- int * GetDFIGlobalVoxel ()

  DFI Domain **の***GlobalVoxel* **の取り出し**
- int * GetDFIGlobalDivision ()

  DFI Domain **の***GlobalDivision* **の取り出し**
- void AddUnit (const std::string Name, const std::string Unit, const double reference, const double difference=0.0, const bool BsetDiff=false)

  Uuit **をセットする**
- CIO::E_CIO_ERRORCODE GetUnitElem (const std::string Name, cio_UnitElem &unit)

  UuitElem **を取得する**
- CIO::E_CIO_ERRORCODE GetUnit (const std::string Name, std::string &unit, double &ref, double &diff, bool &bSetDiff)

  UnitElem **のメンバ変数毎に取得する**
- void SetTimeSliceFlag (const CIO::E_CIO_ONOFF ONOFF)

  TimeSlice OnOff **フラグをセットする**
- void setComponentVariable (int pcomp, std::string compName)

  FileInfo **の成分名を登録する**
- std::string getComponentVariable (int pcomp)

  FileInfo **の成分名を取得する**
- CIO::E_CIO_ERRORCODE getVectorMinMax (const unsigned step, double &vec_min, double &vec_max)

  DFI **に出力されている** *minmax* **の合成値を取得**
- CIO::E_CIO_ERRORCODE getMinMax (const unsigned step, const int compNo, double &min_value, double &max_value)
- CIO::E_CIO_ERRORCODE CheckReadRank (cio_Domain dfi_domain, const int head[3], const int tail[3], CIO::E_CIO_READTYPE readflag, vector< int > &readRankList)

  読込みランクリストの作成
- void setIntervalStep (int interval_step, int base_step=0, int start_step=0, int last_step=-1)

  出力インターバルステップの登録
- void setIntervalTime (double interval_time, double dt, double base_time=0.0, double start_time=0.0, double last_time=-1.0)

  インターバルタイムの登録
- bool normalizeTime (const double scale)

  インターバルの計算に使われる全ての時間をスケールで無次元化する
- void normalizeBaseTime (const double scale)

  インターバルの *base_time* **をスケールで無次元化する**
- void normalizeIntervalTime (const double scale)

  インターバルの *interval* **をスケールで無次元化する**
- void normalizeStartTime (const double scale)

  インターバルの *start_time* **をスケールで無次元化する**
- void normalizeLastTime (const double scale)

  インターバルの *last_time* **をスケールで無次元化する**
- void normalizeDelteT (const double scale)

  インターバルの*DetlaT* **をスケールで無次元化する**
- virtual cio_Array * ReadFieldData (std::string fname, const unsigned step, double &time, const int sta[3], const int end[3], const int DFI_head[3], const int DFI_tail[3], bool avr_mode, unsigned &avr_step, double &avr_time, CIO::E_CIO_ERRORCODE &ret)

  read field data record(sph or bov)
- virtual CIO::E_CIO_ERRORCODE read_HeaderRecord (FILE *fp, bool matchEndian, unsigned step, const int head[3], const int tail[3], int gc, int voxsize[3], double &time)=0

フィールドデータファイルのヘッダーレコード読込み

- virtual CIO::E_CIO_ERRORCODE read_Datarecord (FILE *fp, bool matchEndian, cio_Array *buf, int head[3], int nz, cio_Array *&src)=0

  フィールドデータファイルのデータレコード読込み

- virtual CIO::E_CIO_ERRORCODE read_averaged (FILE *fp, bool matchEndian, unsigned step, unsigned &avr_step, double &avr_time)=0

  sph **ファイルの***Average* **データレコードの読込み**

- template<class T1 , class T2 >
  bool setGridData (cio_TypeArray< T1 > *P, cio_TypeArray< T2 > *S)

  セル中心データを格子点に値をセット

- template<class T >
  void VolumeDataDivide (cio_TypeArray< T > *P)

  内部の格子点のデータを重み付けでで割る

- int MakeDirectory (const std::string path)

  ディレクトリパスの作成 *(MakeDirectorySub* **を呼出して作成***)*

- int MakeDirectoryPath ()

  ディレクトリパスの作成 *(MakeDirectory* **関数を呼出して作成***)*

- std::string Generate_Directory_Path ()

  dfi **のパスと***DirectoryPath* **を連結する関数**

- template<class TimeT , class TimeAvrT >
  CIO_INLINE void * ReadData (CIO::E_CIO_ERRORCODE &ret, const unsigned step, const int gc, const int Gvoxel[3], const int Gdivision[3], const int head[3], const int tail[3], TimeT &time, const bool mode, unsigned &step_avr, TimeAvrT &time_avr)

- template<class T , class TimeT , class TimeAvrT >
  CIO_INLINE CIO::E_CIO_ERRORCODE ReadData (T *val, const unsigned step, const int gc, const int Gvoxel[3], const int Gdivision[3], const int head[3], const int tail[3], TimeT &time, const bool mode, unsigned &step_avr, TimeAvrT &time_avr)

- template<class T , class TimeT , class TimeAvrT >
  CIO_INLINE CIO::E_CIO_ERRORCODE WriteData (const unsigned step, TimeT time, const int sz[3], const int nComp, const int gc, T *val, T *minmax, const bool avr_mode, const unsigned step_avr, TimeAvrT time_avr)

- template<class T1 , class T2 >
  CIO_INLINE bool setGridData (cio_TypeArray< T1 > *P, cio_TypeArray< T2 > *S)

- template<class T >
  CIO_INLINE void VolumeDataDivide (cio_TypeArray< T > *P)

Static Public メソッド

- static cio_DFI * ReadInit (const MPI_Comm comm, const std::string dfifile, const int G_Voxel[3], const int G_Div[3], CIO::E_CIO_ERRORCODE &ret)

  read **インスタンス**

- static std::string Generate_DFI_Name (const std::string prefix)

  出力*DFI* **ファイル名を作成する**

- static std::string Generate_FileName (std::string prefix, int RankID, int step, std::string ext, CIO::E_CIO_OUTPUT_FNAME output_fname, bool mio, CIO::E_CIO_ONOFF TimeSliceDirFlag)

  ファイル名生成

- static cio_DFI * WriteInit (const MPI_Comm comm, const std::string DfiName, const std::string Path, const std::string prefix, const CIO::E_CIO_FORMAT format, const int GCell, const CIO::E_CIO_DTYPE DataType, const CIO::E_CIO_ARRAYSHAPE ArrayShape, const int nComp, const std::string proc_fname, const int G_size[3], const float pitch[3], const float G_origin[3], const int division[3], const int head[3], const int tail[3], const std::string hostname, const CIO::E_CIO_ONOFF TSliceOnOff)

  write **インスタンス** *float* **型**

- static cio_DFI ∗ WriteInit (const MPI_Comm comm, const std::string DfiName, const std::string Path, const std::string prefix, const CIO::E_CIO_FORMAT format, const int GCell, const CIO::E_CIO_DTYPE DataType, const CIO::E_CIO_ARRAYSHAPE ArrayShape, const int nComp, const std::string proc_fname, const int G_size[3], const double pitch[3], const double G_origin[3], const int division[3], const int head[3], const int tail[3], const std::string hostname, const CIO::E_CIO_ONOFF TSliceOnOff)

  write **インスタンス** *double* **型**
- static CIO::E_CIO_DTYPE ConvDatatypeS2E (const std::string datatype)

  データタイプを文字列から *e_num* **番号に変換**
- static std::string ConvDatatypeE2S (const CIO::E_CIO_DTYPE Dtype)

  データタイプを *e_num* **番号から文字列に変換**
- static int MakeDirectorySub (std::string path)

  ディレクトリパスの作成 *(system* **関数** *mkdir* **で作成***)*
- static std::string getVersionInfo ()

Protected メソッド

- virtual CIO::E_CIO_ERRORCODE WriteFieldData (std::string fname, const unsigned step, double time, cio_Array ∗val, const bool mode, const unsigned step_avr, const double time_avr)

  write field data record (double)
- virtual CIO::E_CIO_ERRORCODE write_HeaderRecord (FILE ∗fp, const unsigned step, const double time, const int RankID)=0

  SPH **ヘッダファイルの出力**
- virtual CIO::E_CIO_ERRORCODE write_DataRecord (FILE ∗fp, cio_Array ∗val, const int gc, const int RankID)=0

  SPH **データレコードの出力**
- virtual CIO::E_CIO_ERRORCODE write_averaged (FILE ∗fp, const unsigned step_avr, const double time_avr)=0

  Average **レコードの出力**
- virtual bool write_ascii_header (const unsigned step, const double time)

  ascii **ヘッダーレコード出力** *(bov,avs)*
- void cio_Create_dfiProcessInfo (const MPI_Comm comm, cio_Process &G_Process)

  Create Process.
- CIO::E_CIO_READTYPE CheckReadType (const int G_voxel[3], const int DFI_GlobalVoxel[3], const int G_Div[3], const int DFI_GlobalDivision[3])

  読込み判定判定
- void CreateReadStartEnd (bool isSame, const int head[3], const int tail[3], const int gc, const int DFI_head[3], const int DFI_tail[3], const int DFI_gc, const CIO::E_CIO_READTYPE readflag, int copy_sta[3], int copy_end[3], int read_sta[3], int read_end[3])

  フィールドデータの読込み範囲を求める
- CIO::E_CIO_ERRORCODE WriteIndexDfiFile (const std::string dfi_name)

  index DFI **ファイル出力**

Static Protected メソッド

- static int get_cio_Datasize (CIO::E_CIO_DTYPE Dtype)

  データタイプ毎のサイズを取得

Protected 変数

- MPI_Comm m_comm
  MPI **コミュニケータ**
- std::string m_directoryPath
  index dfi **ファイルのディレクトリパス**
- std::string m_indexDfiName
  index dfi **ファイル名**
- CIO::E_CIO_READTYPE m_read_type
  読込みタイプ
- int m_RankID
  ランク番号
- cio_FileInfo DFI_Finfo
  FileInfo class.
- cio_FilePath DFI_Fpath
  FilePath class.
- cio_Unit DFI_Unit
  Unit class.
- cio_Domain DFI_Domain
  Domain class.
- cio_MPI DFI_MPI
  MPI class.
- cio_TimeSlice DFI_TimeSlice
  TimeSlice class.
- cio_Process DFI_Process
  Process class.
- vector< int > m_readRankList
  読込みランクリスト
- bool m_bgrid_interp_flag
  節点への補間フラグ
- CIO::E_CIO_OUTPUT_TYPE m_output_type
  出力形式 *(ascii,binary,FortarnBinary)*
- CIO::E_CIO_OUTPUT_FNAME m_output_fname
  出力ファイル命名規約 *(step_rank,rank_step)*

6.3.1 説明

CIO main class

cio_DFI.h の 45 行で定義されています。

6.3.2 コンストラクタとデストラクタ

6.3.2.1 cio_DFI::cio_DFI ( )

コンストラクタ

cio_DFI.C の 27 行で定義されています。

参照先 CIO::E_CIO_FNAME_DEFAULT, CIO::E_CIO_OUTPUT_TYPE_DEFAULT, CIO::E_CIO_READTYPE_UNKNOWN, m_output_fname, m_output_type, m_RankID, と m_read_type.

```
28 {
29
30  m_read_type = CIO::E_CIO_READTYPE_UNKNOWN;
31  m_RankID = 0;
32
33  m_output_type = CIO::E_CIO_OUTPUT_TYPE_DEFAULT;
34  m_output_fname = CIO::E_CIO_FNAME_DEFAULT;
35
36 }
```

6.3.2.2 cio_DFI::∼cio_DFI ( )

デストラクタ

cio_DFI.C の 41 行で定義されています。

```
42 {
43
44 }
```

6.3.3 関数

6.3.3.1 void cio_DFI::AddUnit (const std::string *Name,* const std::string *Unit,* const double *reference,* const double *difference* = `0.0`, const bool *BsetDiff* = `false`)

Uuit をセットする

引数

in	*Name*	追加する単位系 ("Length","Velocity",,,,)
in	*Unit*	単位ラベル ("M","CM","MM","M/S",,,)
in	*reference*	規格化したスケール値
in	*difference*	差の値
in	*BsetDiff*	difference の有無

UnitElem の生成

UnilList へのセット

cio_DFI.C の 983 行で定義されています。

参照先 DFI_Unit, と cio_Unit::UnitList.

```
988 {
989
991   cio_UnitElem unit =  cio_UnitElem(Name,Unit,reference,difference,BsetDiff);
992
994   DFI_Unit.UnitList.insert(map<std::string,cio_UnitElem>::value_type(Name,unit));
995
996 }
```

6.3.3.2 CIO::E_CIO_ERRORCODE cio_DFI::CheckReadRank (cio_Domain *dfi_domain,* const int *head[3],* const int *tail[3],* CIO::E_CIO_READTYPE *readflag,* vector< int > & *readRankList*)

読込みランクリストの作成

RankList があるかないか判定しないときは新規にRankList を生成し それをもとにランクマップの生成、読込みランクリスト readRankList を生成する

引数

in	*dfi_domain*	DFI の domain 情報
in	*head*	ソルバーのHeadIndex
in	*tail*	ソルバーのTailIndex
in	*readflag*	読込み方法
out	*readRankList*	読込みランクリスト

戻り値

error code

cio_DFI.C の 1069 行で定義されています。

参照先 cio_Process::CheckReadRank(), と DFI_Process.

```
1074 {
1075
1076   return DFI_Process.CheckReadRank(dfi_domain,head,tail,readflag,readRankList);
1077
1078 }
```

6.3.3.3 CIO::E_CIO_READTYPE cio_DFI::CheckReadType (const int *G_voxel[3]*, const int *DFI_GlobalVoxel[3]*, const int *G_Div[3]*, const int *DFI_GlobalDivision[3]*) [protected]

読込み判定判定

引数

in	*G_voxel*	計算空間全体のボクセルサイズ（自）
in	*DFI_GlobalVoxel*	計算空間全体のボクセルサイズ（DFI）
in	*G_Div*	分割数（自）
in	*DFI_Global-Division*	分割数（DFI）

戻り値

読込みタイプコード

cio_DFI.C の 649 行で定義されています。

参照先 CIO::E_CIO_DIFFDIV_REFINEMENT, CIO::E_CIO_DIFFDIV_SAMERES, CIO::E_CIO_READTYPE_UNKNOWN, CIO::E_CIO_SAMEDIV_REFINEMENT, と CIO::E_CIO_SAMEDIV_SAMERES.

参照元 ReadData(), と ReadInit().

```
653 {
654
655   bool isSameDiv=true;
656
657   //分割数チェック
658   for(int i=0; i<3; i++ ) {
659     if( DFI_GlobalDivision[i] != G_Div[i] ) {
660       isSameDiv = false;
661     }
662   }
663
664   if( isSameDiv ) {
665     if( G_voxel[0] == DFI_GlobalVoxel[0]   &&
666         G_voxel[1] == DFI_GlobalVoxel[1]   &&
667         G_voxel[2] == DFI_GlobalVoxel[2]   ) return CIO::E_CIO_SAMEDIV_SAMERES;
668
669     if( G_voxel[0] == DFI_GlobalVoxel[0]*2 &&
670         G_voxel[1] == DFI_GlobalVoxel[1]*2 &&
671         G_voxel[2] == DFI_GlobalVoxel[2]*2 ) return
    CIO::E_CIO_SAMEDIV_REFINEMENT;
672   } else {
673     if( G_voxel[0] == DFI_GlobalVoxel[0]   &&
674         G_voxel[1] == DFI_GlobalVoxel[1]   &&
```

```
675         G_voxel[2] == DFI_GlobalVoxel[2]   ) return CIO::E_CIO_DIFFDIV_SAMERES;
676
677     if( G_voxel[0] == DFI_GlobalVoxel[0]*2 &&
678         G_voxel[1] == DFI_GlobalVoxel[1]*2 &&
679         G_voxel[2] == DFI_GlobalVoxel[2]*2 ) return
      CIO::E_CIO_DIFFDIV_REFINEMENT;
680   }
681
682   return CIO::E_CIO_READTYPE_UNKNOWN;
683 }
```

6.3.3.4 void cio_DFI::cio_Create_dfiProcessInfo (const MPI_Comm *comm*, cio_Process & *G_Process*) [protected]

Create Process.

引数

in	*comm*	MPI コミュニケータ
out	*G_Process*	Process class

cio_DFI.C の 596 行で定義されています。

参照先 DFI_Process, cio_Rank::HeadIndex, MPI_Comm_rank(), MPI_Comm_size(), MPI_Gather(), MPI_INT, cio_Rank::RankID, cio_Process::RankList, cio_Rank::TailIndex, と cio_Rank::VoxelSize.

参照元 WriteProcDfiFile().

```
598 {
599
600   cio_Rank G_Rank;
601
602   int RankID;
603   MPI_Comm_rank( comm, &RankID );
604
605   int nrank;
606   MPI_Comm_size( comm, &nrank );
607
608   if( nrank > 1 ) {
609     int *headtail = NULL;
610     if( RankID == 0 ) {
611       headtail = new int[6*nrank];
612     }
613
614     int sbuff[6];
615     for(int i=0; i<3; i++) {
616       sbuff[i]   = DFI_Process.RankList[RankID].HeadIndex[i];
617       sbuff[i+3] = DFI_Process.RankList[RankID].TailIndex[i];
618     }
619
620     MPI_Gather(sbuff,6,MPI_INT,headtail,6,MPI_INT,0,comm);
621
622     if( RankID == 0 ) {
623       for(int i=0; i<nrank; i++) {
624         G_Rank.RankID=i;
625         for(int j=0; j<3; j++) {
626           G_Rank.HeadIndex[j]=headtail[i*6+j];
627           G_Rank.TailIndex[j]=headtail[i*6+j+3];
628           G_Rank.VoxelSize[j]=G_Rank.TailIndex[j]-G_Rank.HeadIndex[j]+1;
629         }
630         G_Process.RankList.push_back(G_Rank);
631       }
632     }
633
634     if ( RankID == 0 ) delete [] headtail;
635
636   } else {
637     G_Rank.RankID=0;
638     for(int i=0; i<3; i++) {
639       G_Rank.HeadIndex[i]=DFI_Process.RankList[0].HeadIndex[i];
640       G_Rank.TailIndex[i]=DFI_Process.RankList[0].TailIndex[i];
641       G_Rank.VoxelSize[i]=G_Rank.TailIndex[i]-G_Rank.HeadIndex[i]+1;
642     }
643     G_Process.RankList.push_back(G_Rank);
644   }
645 }
```

6.3.3.5 std::string cio_DFI::ConvDatatypeE2S (const CIO::E_CIO_DTYPE *Dtype*) [static]

データタイプを e_num 番号から文字列に変換

引数

in	*Dtype*	データタイプ

戻り値

データタイプ (string)

cio_DFI.C の 547 行で定義されています。

参照先 D_CIO_FLOAT32, D_CIO_FLOAT64, D_CIO_INT16, D_CIO_INT32, D_CIO_INT64, D_CIO_INT8, D_CIO_UINT16, D_CIO_UINT32, D_CIO_UINT64, D_CIO_UINT8, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_INT16, CIO::E_CIO_INT32, CIO::E_CIO_INT64, CIO::E_CIO_INT8, CIO::E_CIO_UINT16, CIO::E_CIO_UINT32, CIO::E_CIO_UINT64, と CIO::E_CIO_UINT8.

参照元 GetDataTypeString(), と cio_FileInfo::Write().

```
548 {
549   if     ( Dtype == CIO::E_CIO_INT8    ) return D_CIO_INT8;
550   else if( Dtype == CIO::E_CIO_INT16   ) return D_CIO_INT16;
551   else if( Dtype == CIO::E_CIO_INT32   ) return D_CIO_INT32;
552   else if( Dtype == CIO::E_CIO_INT64   ) return D_CIO_INT64;
553   else if( Dtype == CIO::E_CIO_UINT8   ) return D_CIO_UINT8;
554   else if( Dtype == CIO::E_CIO_UINT16  ) return D_CIO_UINT16;
555   else if( Dtype == CIO::E_CIO_UINT32  ) return D_CIO_UINT32;
556   else if( Dtype == CIO::E_CIO_UINT64  ) return D_CIO_UINT64;
557   else if( Dtype == CIO::E_CIO_FLOAT32 ) return D_CIO_FLOAT32;
558   else if( Dtype == CIO::E_CIO_FLOAT64 ) return D_CIO_FLOAT64;
559   else return "dummy";
560
561 }
```

6.3.3.6 CIO::E_CIO_DTYPE cio_DFI::ConvDatatypeS2E (const std::string *datatype*) [static]

データタイプを文字列から e_num 番号に変換

引数

in	*datatype*	dfi から取得したデータタイプ

戻り値

データタイプ (E_CIO_DTYPE)

cio_DFI.C の 528 行で定義されています。

参照先 CIO::E_CIO_DTYPE_UNKNOWN, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_INT16, CIO::E_CIO_INT32, CIO::E_CIO_INT64, CIO::E_CIO_INT8, CIO::E_CIO_UINT16, CIO::E_CIO_UINT32, CIO::E_CIO_UINT64, と CIO::E_CIO_UINT8.

参照元 cio_FileInfo::Read().

```
529 {
530
531   if     ( !strcasecmp(datatype.c_str(),"Int8"   ) ) return CIO::E_CIO_INT8;
532   else if( !strcasecmp(datatype.c_str(),"Int16"  ) ) return CIO::E_CIO_INT16;
533   else if( !strcasecmp(datatype.c_str(),"Int32"  ) ) return CIO::E_CIO_INT32;
534   else if( !strcasecmp(datatype.c_str(),"Int64"  ) ) return CIO::E_CIO_INT64;
535   else if( !strcasecmp(datatype.c_str(),"UInt8"  ) ) return CIO::E_CIO_UINT8;
536   else if( !strcasecmp(datatype.c_str(),"UInt16" ) ) return CIO::E_CIO_UINT16;
537   else if( !strcasecmp(datatype.c_str(),"UInt32" ) ) return CIO::E_CIO_UINT32;
538   else if( !strcasecmp(datatype.c_str(),"UInt64" ) ) return CIO::E_CIO_UINT64;
539   else if( !strcasecmp(datatype.c_str(),"Float32") ) return CIO::E_CIO_FLOAT32;
540   else if( !strcasecmp(datatype.c_str(),"Float64") ) return CIO::E_CIO_FLOAT64;
541
542   return CIO::E_CIO_DTYPE_UNKNOWN;
543 }
```

6.3.3.7 void cio_DFI::CreateReadStartEnd (bool *isSame,* const int *head[3],* const int *tail[3],* const int *gc,* const int *DFI_head[3],* const int *DFI_tail[3],* const int *DFI_gc,* const CIO::E_CIO_READTYPE *readflag,* int *copy_sta[3],* int *copy_end[3],* int *read_sta[3],* int *read_end[3]*) [protected]

フィールドデータの読込み範囲を求める

引数

in	*isSame*	粗密フラグ true:密、false:粗
in	*head*	計算領域の開始位置 (自)
in	*tail*	計算領域の終了位置 (自)
in	*gc*	仮想セル数 (自)
in	*DFI_head*	計算領域の開始位置 (DFI)
in	*DFI_tail*	計算領域の終了位置 (DFI)
in	*DFI_gc*	仮想セル数 (DFI)
in	*readflag*	読込み方法
out	*copy_sta*	コピー開始位置
out	*copy_end*	コピー終了位置
out	*read_sta*	読込み開始位置
out	*read_end*	読込み終了位置

cio_DFI.C の 687 行で定義されています。

参照先 DFI_Domain, と cio_Domain::GlobalVoxel.

参照元 ReadData().

```
699 {
700
701   int src_head[3],src_tail[3],src_gc;
702   if( !isSame ) {
703   // 粗密のとき密に変換、ガイドセルは倍にする
704     src_gc = DFI_gc*2;
705     for(int i=0; i<3; i++) {
706       src_head[i]=DFI_head[i]*2-1;
707       src_tail[i]=DFI_tail[i]*2;
708     }
709   } else {
710   // 粗密でない時各値をコピー
711     src_gc = DFI_gc;
712     for(int i=0; i<3; i++) {
713       src_head[i]=DFI_head[i];
714       src_tail[i]=DFI_tail[i];
715     }
716   }
717
718 //スタート、エンドをセット
719   for(int i=0; i<3; i++) {
720     copy_sta[i] = max(head[i],src_head[i]);
721     copy_end[i] = min(tail[i],src_tail[i]);
722
723     //仮想セルが読込みの実セル内のときの処理（スタート）
724     if( copy_sta[i] == 1 ) {
725       copy_sta[i] -= min(gc,src_gc);
726     } else if( head[i]>src_head[i] ) {
727       copy_sta[i] = max(head[i]-gc,src_head[i]);
728     }
729
730     //仮想セルが読込みの実セル内のときの処理
731     if( ( isSame  && copy_end[i] == DFI_Domain.GlobalVoxel[i] )  ||
732         ( !isSame && copy_end[i] == DFI_Domain.GlobalVoxel[i]*2 ) ) {
733       copy_end[i] += min(gc,src_gc);
734     } else if( tail[i]<src_tail[i] ) {
735       copy_end[i] = min(tail[i]+gc,src_tail[i]);
736     }
737
738     //read satrt/end のセット
739     if( !isSame ) {
740       if( copy_sta[i]>0 ) read_sta[i] = (copy_sta[i]+1)/2;
741       else                read_sta[i] =  copy_sta[i]/2;
742
743       if( copy_end[i]>0 ) read_end[i] = (copy_end[i]+1)/2;
744       else                read_end[i] =  copy_end[i]/2;
745
746     } else {
747       read_sta[i] = copy_sta[i];
748       read_end[i] = copy_end[i];
749     }
750
751   }
752 }
```

6.3.3.8 std::string cio_DFI::Generate_DFI_Name (const std::string *prefix*) [static]

出力DFI ファイル名を作成する

引数

in	*prefix*	ファイル接頭文字

戻り値

DFI ファイル名

cio_DFI.C の 958 行で定義されています。

参照先 CIO::cioPath_ConnectPath(), CIO::cioPath_DirName(), と CIO::cioPath_FileName().

```
959 {
960
961   // directory path
962   std::string dirName  = CIO::cioPath_DirName(prefix);
963
964   // file extension
965   std::string dfiname = CIO::cioPath_FileName(prefix,".dfi");
966
967   // filename
968   std::string fname = CIO::cioPath_ConnectPath( dirName, dfiname );
969
970 #if 0 // for debug
971   printf("prefix    =%s\n", prefix.c_str() );
972   printf("  dirName =%s\n", dirName.c_str() );
973   printf("  dfiname =%s\n", dfiname.c_str() );
974   printf("  fname   =%s\n", fname.c_str() );
975   printf("\n");
976 #endif
977
978   return fname;
979 }
```

6.3.3.9 std::string cio_DFI::Generate_Directory_Path ( )

dfi のパスとDirectoryPath を連結する関数

戻り値

パス名

cio_DFI.C の 929 行で定義されています。

参照先 CIO::cioPath_ConnectPath(), CIO::cioPath_DirName(), CIO::cioPath_isAbsolute(), DFI_Finfo, CIO::E_CIO_ON, m_directoryPath, m_indexDfiName, と cio_FileInfo::TimeSliceDirFlag.

参照元 MakeDirectoryPath().

```
930 {
931
932   // dfi のパスと DirectoryPath を連結する関数
933   // ただし、絶対パスのときは dfi のパスは無視
934   // CIO::cioPath_isAbsolute が true のとき絶対パス
935   // DirectoryPath + TimeSliceDir
936   std::string path = m_directoryPath;
937   if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON )
938   {
939     //path = CIO::cioPath_ConnectPath(path, m_timeSliceDir);
940     path = CIO::cioPath_ConnectPath(path, "");
941   }
942
943   // absolute path
944   if( CIO::cioPath_isAbsolute(path) )
945   {
946     return path;
947   }
948
949   // relative path
950   std::string dfidir = CIO::cioPath_DirName(m_indexDfiName);
951   path = CIO::cioPath_ConnectPath(dfidir, path);
952   return path;
953
954 }
```

6.3.3.10 std::string cio_DFI::Generate_FieldFileName (int *RankID,* int *step,* const bool *mio*)

フィールドデータ（SPH,BOV) ファイル名の作成 (ディレクトリパスが付加されている）

引数

in	*RankID*	ランク番号
in	*step*	読込みステップ番号
in	*mio*	並列判定フラグ（逐次 or 並列の判定用）

戻り値

生成されたファイル名

cio_DFI.C の 756 行で定義されています。

参照先 D_CIO_EXT_BOV, D_CIO_EXT_FUNC, D_CIO_EXT_SPH, D_CIO_EXT_VTK, DFI_Finfo, cio_FileInfo::DirectoryPath, CIO::E_CIO_FMT_AVS, CIO::E_CIO_FMT_BOV, CIO::E_CIO_FMT_PLOT3D, CIO::E_CIO_FMT_SPH, CIO::E_CIO_FMT_VTK, CIO::E_CIO_ON, cio_FileInfo::FileFormat, cio_FileInfo::Prefix, と cio_FileInfo::TimeSliceDirFlag.

参照元 ReadData(), と WriteData().

```
759 {
760
761   if( DFI_Finfo.DirectoryPath.empty() ) return NULL;
762   if( DFI_Finfo.Prefix.empty() ) return NULL;
763
764   std::string fmt;
765   if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_SPH ) {
766     fmt=D_CIO_EXT_SPH;
767   } else if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_BOV ) {
768     fmt=D_CIO_EXT_BOV;
769 //FCONV 20131122.s
770   } else if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_AVS ) {
771     //fmt=D_CIO_EXT_SPH;
772     fmt=D_CIO_EXT_BOV;
773   } else if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_VTK ) {
774     fmt=D_CIO_EXT_VTK;
775   } else if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_PLOT3D ) {
776     fmt=D_CIO_EXT_FUNC;
777 //FCONV 20131122.e
778   }
779
780   int len = DFI_Finfo.DirectoryPath.size() + DFI_Finfo.Prefix.size() + fmt.size() + 25;
781   // id(6) + step(10) + 1(\0) + "_"(2) + "."(1)+"id"(2)
782   if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON ) len += 11;
783
784   char* tmp = new char[len];
785   memset(tmp, 0, sizeof(char)*len);
786
787   if( mio ) {
788     if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON ) {
789       sprintf(tmp, "%s/%010d/%s_%010d_id%06d.%s",DFI_Finfo.DirectoryPath.c_str(),step,
    DFI_Finfo.Prefix.c_str(),
790             step,RankID,fmt.c_str());
791     } else {
792       sprintf(tmp, "%s/%s_%010d_id%06d.%s",DFI_Finfo.DirectoryPath.c_str(),
    DFI_Finfo.Prefix.c_str(),
793             step,RankID,fmt.c_str());
794     }
795   } else {
796     if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON ) {
797       sprintf(tmp, "%s/%010d/%s_%010d.%s",DFI_Finfo.DirectoryPath.c_str(),step,
    DFI_Finfo.Prefix.c_str(),
798             step,fmt.c_str());
799     } else {
800       sprintf(tmp, "%s/%s_%010d.%s",DFI_Finfo.DirectoryPath.c_str(),DFI_Finfo.
    Prefix.c_str(),
801             step,fmt.c_str());
802     }
803   }
804
805   std::string fname(tmp);
806   if( tmp ) delete [] tmp;
807
808   return fname;
809 }
```

6.3.3.11 std::string cio_DFI::Generate_FileName (std::string *prefix,* int *RankID,* int *step,* std::string *ext,* CIO::E_CIO_OUTPUT_FNAME *output_fname,* bool *mio,* CIO::E_CIO_ONOFF *TimeSliceDirFlag*) [static]

ファイル名生成

引数

in	*prefix*	ベースファイル名
in	*RankID*	ランク番号
in	*step*	出力ステップ番号（負のとき、ステップ番号が付加されない）
in	*ext*	拡張子
in	*output_fname*	step_rank,rank_step 指示
in	*mio*	並列判定フラグ
in	*TimeSliceDir-Flag*	Time Slice 毎の出力指示

戻り値

生成されたファイル名

cio_DFI.C の 814 行で定義されています。

参照先 CIO::E_CIO_FNAME_RANK_STEP, と CIO::E_CIO_ON.

参照元 cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_avs_cord(), cio_DFI_AVS::write_avs_header(), cio_DFI_PLOT3D::write_GridData(), と WriteData().

```
821 {
822
823   int len = prefix.size()+ext.size()+100;
824   char* tmp = new char[len];
825   memset(tmp, 0, sizeof(char)*len);
826
827   //step 出力なしのファイル名生成
828   if( step < 0 )
829   {
830     if( mio ) {
831       sprintf(tmp,"%s_id%06d.%s",prefix.c_str(),RankID,ext.c_str());
832     } else {
833       sprintf(tmp,"%s.%s",prefix.c_str(),ext.c_str());
834     }
835     std::string fname(tmp);
836     if( tmp ) delete [] tmp;
837     return fname;
838   }
839
840   //RankID 出力なしのファイル名生成
841   if( !mio ) {
842     sprintf(tmp,"%s_%010d.%s",prefix.c_str(),step,ext.c_str());
843     std::string fname(tmp);
844     if( tmp ) delete [] tmp;
845     return fname;
846   }
847
848   //step_rank
849   if( output_fname != CIO::E_CIO_FNAME_RANK_STEP )
850   {
851     if( TimeSliceDirFlag == CIO::E_CIO_ON ) {
852       sprintf(tmp,"%010d/%s_%010d_id%06d.%s",step,prefix.c_str(),step,RankID,ext.c_str());
853     } else {
854       sprintf(tmp,"%s_%010d_id%06d.%s",prefix.c_str(),step,RankID,ext.c_str());
855     }
856   } else if( output_fname == CIO::E_CIO_FNAME_RANK_STEP )
857   {
858   //rank_step
859     if( TimeSliceDirFlag == CIO::E_CIO_ON ) {
860       sprintf(tmp,"%010d/%s_id%06d_%010d.%s",step,prefix.c_str(),RankID,step,ext.c_str());
861     } else {
862       sprintf(tmp,"%s_id%06d_%010d.%s",prefix.c_str(),RankID,step,ext.c_str());
863     }
864   }
865
866   std::string fname(tmp);
867   if( tmp ) delete [] tmp;
868   return fname;
869
870 }
```

6.3.3.12 int cio_DFI::get_cio_Datasize (CIO::E_CIO_DTYPE *Dtype*) [static], [protected]

データタイプ毎のサイズを取得

引数

in	*Dtype*	データタイプ (Int8,Int16,,,,etc)

戻り値

データサイズ
0 エラー

cio_DFI.C の 564 行で定義されています。

参照先 CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_INT16, CIO::E_CIO_INT32, CIO::E_CIO_INT64, CIO::E_CIO_INT8, CIO::E_CIO_UINT16, CIO::E_CIO_UINT32, CIO::E_CIO_UINT64, と CIO::E_CIO_UINT8.

参照元 cio_DFI_BOV::write_DataRecord(), cio_DFI_AVS::write_DataRecord(), と cio_DFI_SPH::write_DataRecord().

```
565 {
566
567   if     ( Dtype == CIO::E_CIO_INT8    ) return sizeof(char);
568   else if( Dtype == CIO::E_CIO_INT16   ) return sizeof(short);
569   else if( Dtype == CIO::E_CIO_INT32   ) return sizeof(int);
570   else if( Dtype == CIO::E_CIO_INT64   ) return sizeof(long long);
571   else if( Dtype == CIO::E_CIO_UINT8   ) return sizeof(unsigned char);
572   else if( Dtype == CIO::E_CIO_UINT16  ) return sizeof(unsigned short);
573   else if( Dtype == CIO::E_CIO_UINT32  ) return sizeof(unsigned int);
574   else if( Dtype == CIO::E_CIO_UINT64  ) return sizeof(unsigned long long);
575   else if( Dtype == CIO::E_CIO_FLOAT32 ) return sizeof(float);
576   else if( Dtype == CIO::E_CIO_FLOAT64 ) return sizeof(double);
577   else return 0;
578
579 }
```

6.3.3.13 std::string cio_DFI::get_dfi_fname ( ) [inline]

cio_DFI.h の 313 行で定義されています。

参照先 m_indexDfiName.

```
314   { return m_indexDfiName; };
```

6.3.3.14 CIO::E_CIO_ARRAYSHAPE cio_DFI::GetArrayShape ( )

配列形状を返す

戻り値

配列形状（e_num 番号)

cio_DFI.C の 500 行で定義されています。

参照先 cio_FileInfo::ArrayShape, と DFI_Finfo.

```
501 {
502   return (CIO::E_CIO_ARRAYSHAPE)DFI_Finfo.ArrayShape;
503 }
```

6.3.3.15 std::string cio_DFI::GetArrayShapeString ( )

配列形状を文字列で返す

戻り値

配列形状（文字列)

cio_DFI.C の 491 行で定義されています。

参照先 cio_FileInfo::ArrayShape, D_CIO_IJNK, D_CIO_NIJK, DFI_Finfo, CIO::E_CIO_IJKN, と CIO::E_CIO_NIJK.

```
492 {
493   if( DFI_Finfo.ArrayShape == CIO::E_CIO_IJKN ) return D_CIO_IJNK;
494   if( DFI_Finfo.ArrayShape == CIO::E_CIO_NIJK ) return D_CIO_NIJK;
495   return " ";
496 }
```

6.3.3.16 const cio_Domain * cio_DFI::GetcioDomain ( )

cio_Domain クラスのポインタ取得

戻り値

cio_Domain クラスポインタ

cio_DFI.C の 265 行で定義されています。

参照先 DFI_Domain.

```
266 {
267   return &DFI_Domain;
268 }
```

6.3.3.17 const cio_FileInfo * cio_DFI::GetcioFileInfo ( )

cioFileInfo クラスのポインタを取得

戻り値

cio_FileInfo クラスポインタ

cio_DFI.C の 227 行で定義されています。

参照先 DFI_Finfo.

```
228 {
229   return &DFI_Finfo;
230 }
```

6.3.3.18 const cio_FilePath * cio_DFI::GetcioFilePath ( )

cio_FilePath クラスのポインタを取得

戻り値

cio_FilePath クラスポインタ

cio_DFI.C の 242 行で定義されています。

参照先 DFI_Fpath.

```
243 {
244   return &DFI_Fpath;
245 }
```

6.3.3.19 const cio_MPI * cio_DFI::GetcioMPI ( )

cio_MPI クラスのポインタ取得

戻り値

cio_MPI クラスポインタ

cio_DFI.C の 282 行で定義されています。

参照先 DFI_MPI.

```
283 {
284   return &DFI_MPI;
285 }
```

6.3.3.20 const cio_Process * cio_DFI::GetcioProcess ( )

cio_Process クラスのポインタ取得

戻り値

cio_Process クラスポインタ

cio_DFI.C の 313 行で定義されています。

参照先 DFI_Process.

```
314 {
315   return &DFI_Process;
316 }
```

6.3.3.21 const cio_TimeSlice * cio_DFI::GetcioTimeSlice ( )

cio_TimeSlice クラスのポインタ取得

戻り値

cio_TimeSlice クラスポインタ

cio_DFI.C の 298 行で定義されています。

参照先 DFI_TimeSlice.

```
299 {
300   return &DFI_TimeSlice;
301 }
```

6.3.3.22 const cio_Unit * cio_DFI::GetcioUnit ( )

cio_Unit クラスのポインタを取得

戻り値

cio_Unit クラスポインタ

cio_DFI.C の 250 行で定義されています。

参照先 DFI_Unit.

```
251 {
252   return &DFI_Unit;
253 }
```

6.3.3.23 std::string cio_DFI::getComponentVariable (int *pcomp*)

FileInfo の成分名を取得する

引数

in	*pcomp*	成分位置 0:u, 1:v, 2:w

戻り値

成分名

cio_DFI.C の 1036 行で定義されています。

参照先 DFI_Finfo, と cio_FileInfo::getComponentVariable().

参照元 cio_DFI_AVS::write_avs_header().

```
1037 {
1038
1039   return DFI_Finfo.getComponentVariable(pcomp);
1040
1041 }
```

6.3.3.24 CIO::E_CIO_DTYPE cio_DFI::GetDataType ( )

get DataType （データタイプの取り出し関数）

戻り値

データタイプ (e_num 番号)

cio_DFI.C の 514 行で定義されています。

参照先 cio_FileInfo::DataType, と DFI_Finfo.

参照元 cio_DFI_BOV::write_ascii_header(), と cio_DFI_AVS::write_avs_header().

```
515 {
516   return (CIO::E_CIO_DTYPE)DFI_Finfo.DataType;
517 }
```

6.3.3.25 std::string cio_DFI::GetDataTypeString ( )

get DataType （データタイプの取り出し関数）

戻り値

データタイプ（文字列)

cio_DFI.C の 507 行で定義されています。

参照先 ConvDatatypeE2S(), cio_FileInfo::DataType, と DFI_Finfo.

参照元 cio_DFI_AVS::write_avs_header().

```
508 {
509   return ConvDatatypeE2S((CIO::E_CIO_DTYPE)DFI_Finfo.DataType);
510 }
```

6.3.3.26 int * cio_DFI::GetDFIGlobalDivision ( )

DFI Domain のGlobalDivision の取り出し

戻り値

GlobalDivision のポインタ

cio_DFI.C の 590 行で定義されています。

参照先 DFI_Domain, と cio_Domain::GlobalDivision.

```
591 {
592   return DFI_Domain.GlobalDivision;
593 }
```

6.3.3.27 int * cio_DFI::GetDFIGlobalVoxel ( )

DFI Domain のGlobalVoxel の取り出し

戻り値

GlobalVoxel のポインタ

cio_DFI.C の 583 行で定義されています。

参照先 DFI_Domain, と cio_Domain::GlobalVoxel.

```
584 {
585   return DFI_Domain.GlobalVoxel;
586 }
```

6.3.3.28 CIO::E_CIO_ERRORCODE cio_DFI::getMinMax (const unsigned *step*, const int *compNo*, double & *min_value*, double & *max_value*)

brief DFI に出力されている minmax を取得

引数

in	*step*	取得するステップ
in	*compNo*	成分No(0〜n)
out	*min_value*	取得した min
out	*max_value*	取得した max

戻り値

error code 取得出来たときは E_CIO_SUCCESS

cio_DFI.C の 1056 行で定義されています。

参照先 DFI_TimeSlice, と cio_TimeSlice::getMinMax().

```
1060 {
1061
1062   return DFI_TimeSlice.getMinMax(step,compNo,min_value,max_value);
1063
1064 }
```

6.3.3.29 int cio_DFI::GetNumComponent ( )

get Number of Component （成分数の取り出し関数）

戻り値

成分数

cio_DFI.C の 521 行で定義されています。

参照先 cio_FileInfo::Component, と DFI_Finfo.

```
522 {
523   return DFI_Finfo.Component;
524 }
```

6.3.3.30 CIO::E_CIO_ERRORCODE cio_DFI::GetUnit (const std::string *Name*, std::string & *unit*, double & *ref*, double & *diff*, bool & *bSetDiff*)

UnitElem のメンバ変数毎に取得する

引数

in	*Name*	取得する単位系
out	*unit*	単位文字列
out	*ref*	reference
out	*diff*	difference
out	*bSetDiff*	difference の有無（true:あり false:なし）

戻り値

error code

cio_DFI.C の 1008 行で定義されています。

参照先 DFI_Unit, と cio_Unit::GetUnit().

```
1013 {
1014   return DFI_Unit.GetUnit(Name, unit, ref, diff, bSetDiff);
1015 }
```

6.3.3.31 CIO::E_CIO_ERRORCODE cio_DFI::GetUnitElem (const std::string *Name*, cio_UnitElem & *unit*)

UuitElem を取得する

引数

in	*Name*	取得する単位系
out	*unit*	取得した cio_UnitElem

戻り値

error code

cio_DFI.C の 1000 行で定義されています。

参照先 DFI_Unit, と cio_Unit::GetUnitElem().

```
1002 {
1003   return DFI_Unit.GetUnitElem(Name, unit);
1004 }
```

6.3.3.32 CIO::E_CIO_ERRORCODE cio_DFI::getVectorMinMax (const unsigned *step*, double & *vec_min*, double & *vec_max*)

DFI に出力されている minmax の合成値を取得

引数

in	*step*	取得するステップ
out	*vec_min*	取得した minmax の合成値
out	*vec_max*	取得した minmax の合成値

戻り値

error code 取得出来たときは E_CIO_SUCCESS

cio_DFI.C の 1045 行で定義されています。

参照先 DFI_TimeSlice, と cio_TimeSlice::getVectorMinMax().

```
1048 {
1049
1050   return DFI_TimeSlice.getVectorMinMax(step,vec_min,vec_max);
1051
1052 }
```

6.3.3.33 static std::string cio_DFI::getVersionInfo ( ) [inline], [static]

バージョンを出力する

cio_DFI.h の 1009 行で定義されています。

参照先 CIO_VERSION_NO.

```
1010   {
1011     std::string str(CIO_VERSION_NO);
1012     return str;
1013   }
```

6.3.3.34 int cio_DFI::MakeDirectory (const std::string *path*)

ディレクトリパスの作成 (MakeDirectorySub を呼出して作成)

引数

in	*path*	パス

戻り値

error code

cio_DFI.C の 876 行で定義されています。

参照先 MakeDirectorySub().

参照元 MakeDirectoryPath(), と WriteData().

```
877 {
878   int ret = MakeDirectorySub(path);
879   if( ret != 0 )
880   {
881      // 既存以外のエラー
882      if ( EEXIST != errno )
883      {
884         printf( "\tError(errno)=[%s]\n", strerror(errno) );
885         return 0;
886      }
887   }
888
889   // failed
890   return 1;
891 }
```

6.3.3.35 int cio_DFI::MakeDirectoryPath ( )

ディレクトリパスの作成 (MakeDirectory 関数を呼出して作成)

戻り値

error code

cio_DFI.C の 895 行で定義されています。

参照先 Generate_Directory_Path(), と MakeDirectory().

```
896 {
897   // DirectoryPath with TimeSlice
898   std::string path = Generate_Directory_Path();
899
900   return MakeDirectory(path);
901 }
```

6.3.3.36 int cio_DFI::MakeDirectorySub (std::string *path*) [static]

ディレクトリパスの作成 (system 関数 mkdir で作成)

引数

in	*path*	パス

戻り値

error code

cio_DFI.C の 904 行で定義されています。

参照先 CIO::cioPath_DirName().

参照元 MakeDirectory().

```
905 {
906
907   umask(022);
908
909   int ret = mkdir(path.c_str(), 0777);
910   if( ret != 0 )
911   {
912     if( errno == EEXIST ) return 0;
913
914     std::string parent = CIO::cioPath_DirName(path);
915     int ret2 = MakeDirectorySub( parent );
916     if( ret2 != 0 )
917     {
918       return ret2;
919     }
920     ret = MakeDirectorySub( path );
921   }
922
923   return ret;
924
925 }
```

6.3.3.37 void cio_DFI::normalizeBaseTime (const double *scale*)

インターバルの base_time をスケールで無次元化する

引数

in	*scale*	スケール

6.3.3.38 void cio_DFI::normalizeDelteT (const double *scale*)

インターバルのDetlaT をスケールで無次元化する

引数

in	*scale*	スケール

6.3.3.39 void cio_DFI::normalizeIntervalTime (const double *scale*)

インターバルの interval をスケールで無次元化する

引数

in	*scale*	スケール

6.3.3.40 void cio_DFI::normalizeLastTime (const double *scale*)

インターバルの last_time をスケールで無次元化する

引数

in	*scale*	スケール

6.3.3.41 void cio_DFI::normalizeStartTime (const double *scale*)

インターバルの start_time をスケールで無次元化する

引数

in	*scale*	スケール

6.3.3.42 bool cio_DFI::normalizeTime (const double *scale*)

インターバルの計算に使われる全ての時間をスケールで無次元化する

(base_time, interval_time, start_time, last_time)

引数

in	*scale*	スケール return mode がStep のときは false を返す、無次元化しない

6.3.3.43 virtual CIO::E_CIO_ERRORCODE cio_DFI::read_averaged (FILE * *fp*, bool *matchEndian*, unsigned *step*, unsigned & *avr_step*, double & *avr_time*) [pure virtual]

sph ファイルのAverage データレコードの読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*step*	読込み step 番号
out	*avr_step*	平均ステップ
out	*avr_time*	平均タイム

cio_DFI_SPH, cio_DFI_PLOT3D, cio_DFI_AVS, cio_DFI_VTK, と cio_DFI_BOVで実装されています。

参照元 ReadFieldData().

6.3.3.44 virtual CIO::E_CIO_ERRORCODE cio_DFI::read_Datarecord (FILE * *fp*, bool *matchEndian*, cio_Array * *buf*, int *head[3]*, int *nz*, cio_Array *& *src*) [pure virtual]

フィールドデータファイルのデータレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*buf*	読込み用バッファ
in	*head*	読込みバッファHeadIndex
in	*nz*	z 方向のボクセルサイズ（実セル＋ガイドセル＊２）
out	*src*	読み込んだデータを格納した配列のポインタ

cio_DFI_SPH, cio_DFI_PLOT3D, cio_DFI_AVS, cio_DFI_VTK, と cio_DFI_BOVで実装されています。

参照元 ReadFieldData().

6.3.3.45 virtual CIO::E_CIO_ERRORCODE cio_DFI::read_HeaderRecord (FILE * *fp*, bool *matchEndian*, unsigned *step*, const int *head[3]*, const int *tail[3]*, int *gc*, int *voxsize[3]*, double & *time*) [pure virtual]

フィールドデータファイルのヘッダーレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*step*	ステップ番号
in	*head*	dfi のHeadIndex
in	*tail*	dfi のTailIndex
in	*gc*	dfi のガイドセル数
out	*voxsize*	voxsize
out	*time*	時刻

戻り値

true:出力成功 false:出力失敗

cio_DFI_SPH, cio_DFI_PLOT3D, cio_DFI_AVS, cio_DFI_VTK, と cio_DFI_BOVで実装されています。

参照元 ReadFieldData().

6.3.3.46 template<class TimeT , class TimeAvrT > CIO_INLINE void* cio_DFI::ReadData (CIO::E_CIO_ERRORCODE & *ret*, const unsigned *step*, const int *gc*, const int *Gvoxel[3]*, const int *Gdivision[3]*, const int *head[3]*, const int *tail[3]*, TimeT & *time*, const bool *mode*, unsigned & *step_avr*, TimeAvrT & *time_avr*)

cio_DFI_inline.h の 36 行で定義されています。

参照先 cio_FileInfo::ArrayShape, cio_FileInfo::Component, cio_FileInfo::DataType, DFI_Finfo, CIO::E_CIO_SUCCESS, cio_Array::getData(), cio_Array::instanceArray(), と ReadData().

```
47 {
48
49    int sz[3];
50    for(int i=0; i<3; i++) sz[i]=tail[i]-head[i]+1;
51    cio_Array *data = cio_Array::instanceArray
52                      ( DFI_Finfo.DataType
53                      , DFI_Finfo.ArrayShape
54                      , sz
55                      , gc
56                      , DFI_Finfo.Component);
57
58    double d_time = (double)time;
59    double d_time_avr = (double)time_avr;
60
61 //   int ret = ReadData(data, step, gc, Gvoxel, Gdivision, head, tail,
62    ret = ReadData(data, step, gc, Gvoxel, Gdivision, head, tail,
63                         d_time, mode, step_avr, d_time_avr);
64
65    if( ret != CIO::E_CIO_SUCCESS ) {
66      delete data;
67      return NULL;
68    }
69
70 //   T* ptr = (T*)data->getData(true);
71    void* ptr = data->getData(true);
72    delete data;
73    time = d_time;
74    time_avr = d_time_avr;
75
76    return ptr;
77 }
```

6.3.3.47 template<class T , class TimeT , class TimeAvrT > CIO_INLINE CIO::E_CIO_ERRORCODE cio_DFI::ReadData (T * *val*, const unsigned *step*, const int *gc*, const int *Gvoxel[3]*, const int *Gdivision[3]*, const int *head[3]*, const int *tail[3]*, TimeT & *time*, const bool *mode*, unsigned & *step_avr*, TimeAvrT & *time_avr*)

cio_DFI_inline.h の 83 行で定義されています。

参照先 cio_FileInfo::ArrayShape, cio_FileInfo::Component, DFI_Finfo, CIO::E_CIO_SUCCESS, cio_Array::instanceArray(), と ReadData().

```
94 {
95
96    int sz[3];
97    for(int i=0; i<3; i++) sz[i]=tail[i]-head[i]+1;
98
99    cio_Array *data = cio_Array::instanceArray
100                     ( val
101                     , DFI_Finfo.ArrayShape
102                     , sz
103                     , gc
104                     , DFI_Finfo.Component);
105
106    double d_time = (double)time;
107    double d_time_avr = (double)time_avr;
108
109    CIO::E_CIO_ERRORCODE ret;
110    ret = ReadData(data, step, gc, Gvoxel, Gdivision, head, tail,
111                   d_time, mode, step_avr, d_time_avr);
112
113    if( ret == CIO::E_CIO_SUCCESS ) {
114      time = d_time;
115      time_avr = d_time_avr;
116    }
117
118    //data->getData(true);
119    delete data;
120
121    return ret;
122 }
```

6.3.3.48 template<class TimeT , class TimeAvrT > void* cio_DFI::ReadData (CIO::E_CIO_ERRORCODE & *ret*, const unsigned *step*, const int *gc*, const int *Gvoxel[3]*, const int *Gdivision[3]*, const int *head[3]*, const int *tail[3]*, TimeT & *time*, const bool *mode*, unsigned & *step_avr*, TimeAvrT & *time_avr*)

read field data record (template function)

読み込んだデータのポインタを戻り値として返す

引数

out	*ret*	終了コード 1:正常、1 以外：エラー
in	*step*	入力ステップ番号
in	*gc*	仮想セル数
in	*Gvoxel*	グローバルボクセルサイズ
in	*Gdivision*	領域分割数
in	*head*	計算領域の開始位置
in	*tail*	計算領域の終了位置
out	*time*	読み込んだ時間
in	*mode*	平均ステップ&時間読込みフラグ　false : 読込み true : 読み込まない
out	*step_avr*	平均ステップ
out	*time_avr*	平均時間

戻り値

読みんだフィールドデータのポンタ

参照元 ReadData().

6.3.3.49 template<class T , class TimeT , class TimeAvrT > CIO::E_CIO_ERRORCODE cio_DFI::ReadData (T * *val*, const unsigned *step*, const int *gc*, const int *Gvoxel[3]*, const int *Gdivision[3]*, const int *head[3]*, const int *tail[3]*, TimeT & *time*, const bool *mode*, unsigned & *step_avr*, TimeAvrT & *time_avr*)

read field data record (template function)

引数で渡された配列ポインタにデータを読込む

引数

out	*val*	読み込んだデータポインタ
in	*step*	入力ステップ番号
in	*gc*	仮想セル数
in	*Gvoxel*	グローバルボクセルサイズ
in	*Gdivision*	領域分割数
in	*head*	計算領域の開始位置
in	*tail*	計算領域の終了位置
out	*time*	読み込んだ時間
in	*mode*	平均ステップ&時間読込みフラグ　false : 読込み true : 読み込まない
out	*step_avr*	平均ステップ
out	*time_avr*	平均時間

戻り値

終了コード 1:正常 1 以外:エラー

6.3.3.50 CIO::E_CIO_ERRORCODE cio_DFI::ReadData (cio_Array * *val*, const unsigned *step*, const int *gc*, const int *Gvoxel[3]*, const int *Gdivision[3]*, const int *head[3]*, const int *tail[3]*, double & *time*, const bool *mode*, unsigned & *step_avr*, double & *time_avr*)

read field data record

template ReadData 関数で型に応じた配列を確保した後、呼び出される

引数

out	*val*	読み込み先の配列をポインタで渡す
in	*step*	読み込むステップ番号
in	*gc*	仮想セル数
in	*Gvoxel*	グローバルボクセルサイズ
in	*Gdivision*	領域分割数
in	*head*	計算領域の開始位置
in	*tail*	計算領域の終了位置
out	*time*	読み込んだ時間
in	*mode*	平均ステップ&時間読込みフラグ　false : 読込み true : 読み込まない
out	*step_avr*	平均ステップ
out	*time_avr*	平均時間

戻り値

終了コード 1:正常 1 以外:エラー

dts にHead/Tail をセット

index DFI ファイルの ディレクトリパスを取得

< DFI ファイルの並列フラグ

< 粗密フラグ true:密 false:粗

< 読込み判定フラグ

読込みフラグ取得

粗密フラグセット

読込みランクリストの生成

<Process が 1 より大きい時並列

ファイル名の生成

読込み領域 start end の取得

読込み方法の取得

フィールドデータの読込み

読込めたファイル名の出力（ランク 0 のみ)

src にHead/Tail をセット

粗密処理

cio_DFI_Read.C の 20 行で定義されています。

参照先 cio_Process::CheckReadRank(), CheckReadType(), CIO::cioPath_ConnectPath(), CIO::cioPath_DirName(), CIO::cioPath_isAbsolute(), cio_Array::copyArray(), CreateReadStartEnd(), DFI_Domain, DFI_Finfo, DFI_Process, cio_FileInfo::DirectoryPath, CIO::E_CIO_DIFFDIV_REFINEMENT, CIO::E_CIO_SAMEDIV_REFINEMENT, CIO::E_CIO_SUCCESS, Generate_FieldFileName(), cio_Domain::GlobalDivision, cio_Domain::GlobalVoxel, cio_FileInfo::GuideCell, cio_Array::interp_coarse(), m_indexDfiName, m_RankID, m_readRankList, cio_Process::RankList, ReadFieldData(), と cio_Array::setHeadIndex().

```
31 {
32
33   CIO::E_CIO_ERRORCODE ret;
34
37   int Shead[3];
38   for(int i=0; i<3; i++) Shead[i] = head[i];
39   dst->setHeadIndex(Shead);
40
42   std::string dir = CIO::cioPath_DirName(m_indexDfiName);
43
44   bool mio = false;
45   bool isSame =true;
```

```
46   CIO::E_CIO_READTYPE readflag;
47
49   readflag = CheckReadType(Gvoxel, DFI_Domain.GlobalVoxel,
50                            Gdivision, DFI_Domain.GlobalDivision);
51
53   if( readflag == CIO::E_CIO_SAMEDIV_REFINEMENT || readflag ==
      CIO::E_CIO_DIFFDIV_REFINEMENT ) isSame = false;
54
56   ret = DFI_Process.CheckReadRank(DFI_Domain, head, tail, readflag,
      m_readRankList);
57   if( ret != CIO::E_CIO_SUCCESS ) {
58     printf("error code : %d\n",(int)ret);
59     return ret;
60   }
61
62   if( DFI_Process.RankList.size() > 1 ) mio = true;
63
64   for(int i=0; i<m_readRankList.size(); i++) {
65     int n = m_readRankList[i];
66     int ID= DFI_Process.RankList[n].RankID;
67
69     std::string fname;
70     if( CIO::cioPath_isAbsolute(DFI_Finfo.DirectoryPath) ){
71       fname = Generate_FieldFileName(ID,step,mio);
72     } else {
73       std::string tmp = Generate_FieldFileName(ID,step,mio);
74       fname = CIO::cioPath_ConnectPath( dir, tmp );
75     }
76
77     int copy_sta[3],copy_end[3],read_sta[3],read_end[3];
78
80     CreateReadStartEnd(isSame,head, tail, gc, DFI_Process.RankList[n].HeadIndex,
81                        DFI_Process.RankList[n].TailIndex,
82                        DFI_Finfo.GuideCell, readflag,
83                        copy_sta, copy_end, read_sta, read_end);
84
89     cio_Array* src = ReadFieldData(fname, step, time, read_sta, read_end,
90                                    DFI_Process.RankList[n].HeadIndex,
91                                    DFI_Process.RankList[n].TailIndex,
92                                    avr_mode, avr_step, avr_time, ret);
93     if( ret != CIO::E_CIO_SUCCESS ) {
94       delete src;
95       return ret;
96     }
97
99     if( m_RankID == 0 ) {
100       printf("\t[%s] has read :\tstep=%d  time=%e ]\n",fname.c_str(), step, time);
101    }
102
103
105    src->setHeadIndex(read_sta);
106
108    if( !isSame ) {
109      cio_Array *temp = src;
110      int err;
111      src = cio_Array::interp_coarse(temp,err,false);
112      delete temp;
113    }
114
115    src->copyArray(copy_sta,copy_end,dst);
116    delete src;
117
118  }
119
120  return CIO::E_CIO_SUCCESS;
121
122 }
```

6.3.3.51 cio_Array * cio_DFI::ReadFieldData (std::string *fname*, const unsigned *step*, double & *time*, const int *sta[3]*, const int *end[3]*, const int *DFI_head[3]*, const int *DFI_tail[3]*, bool *avr_mode*, unsigned & *avr_step*, double & *avr_time*, CIO::E_CIO_ERRORCODE & *ret*) [virtual]

read field data record(sph or bov)

引数

in	*fname*	FieldData ファイル名
in	*step*	読込みステップ番号
out	*time*	読み込んだ時間
in	*sta*	読込みスタート位置
in	*end*	読込みエンド位置
in	*DFI_head*	dfi のHeadIndex
in	*DFI_tail*	dfi のTailIndex
in	*avr_mode*	平均ステップ&時間読込みフラグ　false : 読込み

true : 読み込まない

引数

out	*avr_step*	平均ステップ
out	*avr_time*	平均時間
out	*ret*	終了コード

戻り値

読み込んだ配列のポインタ

ファイルオープン

Endian セット

ヘッダーレコードの読込み

< voxsize - 2*gc : 実セル数

cio_DFI_Read.C の 126 行で定義されています。

参照先 cio_FileInfo::ArrayShape, cio_FileInfo::Component, cio_FileInfo::DataType, DFI_Finfo, CIO::E_CIO_BIG, CIO::E_CIO_ENDIANTYPE_UNKNOWN, CIO::E_CIO_ERROR_OPEN_FIELDDATA, CIO::E_CIO_ERROR_READ_FIELD_AVERAGED_RECORD, CIO::E_CIO_ERROR_READ_FIELD_DATA_RECORD, CIO::E_CIO_ERROR_READ_FIELD_HEADER_RECORD, CIO::E_CIO_ERROR_READ_FIELDDATA_FILE, CIO::E_CIO_IJKN, CIO::E_CIO_LITTLE, CIO::E_CIO_NIJK, CIO::E_CIO_SUCCESS, cio_FileInfo::Endian, cio_FileInfo::GuideCell, cio_Array::instanceArray(), read_averaged(), read_Datarecord(), read_HeaderRecord(), と cio_Array::setHeadIndex().

参照元 ReadData().

```
137 {
138
139   ret = CIO::E_CIO_SUCCESS;
140
141
142   if( !fname.c_str() || !DFI_Finfo.Component ) {
143     ret = CIO::E_CIO_ERROR_READ_FIELDDATA_FILE;
144     return NULL;
145   }
146
148   FILE* fp;
149   if( !(fp=fopen(fname.c_str(),"rb")) ) {
150     printf("Can't open file. (%s)\n",fname.c_str());
151     ret = CIO::E_CIO_ERROR_OPEN_FIELDDATA;
152     return NULL;
153   }
154
156   int idumy = 1;
157   char* cdumy = (char*)(&idumy);
158   CIO::E_CIO_ENDIANTYPE Endian=CIO::E_CIO_ENDIANTYPE_UNKNOWN;
159   if( cdumy[0] == 0x01 ) Endian = CIO::E_CIO_LITTLE;
160   if( cdumy[0] == 0x00 ) Endian = CIO::E_CIO_BIG;
161
162   bool matchEndian = true;
163   if( Endian != DFI_Finfo.Endian ) matchEndian = false;
164
165   //RealType real_type;
166   int voxsize[3];
168   ret = read_HeaderRecord(fp, matchEndian, step, DFI_head, DFI_tail,
169                           DFI_Finfo.GuideCell, voxsize, time);
170   if( ret != CIO::E_CIO_SUCCESS )
171   {
172     ret = CIO::E_CIO_ERROR_READ_FIELD_HEADER_RECORD;
```

```
173     printf("**** read error\n");
174     fclose(fp);
175     return NULL;
176   }
177
178   int sz[3];
179   for(int i=0; i<3; i++) sz[i]=voxsize[i]-2*DFI_Finfo.GuideCell;
180
181   int szB[3],headB[3];
182   for(int i=0; i<3; i++) {
183     szB[i] = voxsize[i];
184     headB[i] = DFI_head[i] - DFI_Finfo.GuideCell;
185   }
186   //1層ずつ読み込むので、バッファの z サイズは1にしておく
187   szB[2]=1;
188
189 //FCONV 20121216.s
190   //読み込みバッファ
191   cio_Array* buf=NULL;
192   //配列形状が IJKN のときは成分数を1にしてインスタンスする
193   if( DFI_Finfo.ArrayShape == CIO::E_CIO_NIJK ) {
194     buf = cio_Array::instanceArray
195                   ( DFI_Finfo.DataType
196                   , DFI_Finfo.ArrayShape
197                   , szB
198                   , 0
199                   , DFI_Finfo.Component );
200   } else if( DFI_Finfo.ArrayShape == CIO::E_CIO_IJKN ) {
201     buf = cio_Array::instanceArray
202                   ( DFI_Finfo.DataType
203                   , DFI_Finfo.ArrayShape
204                   , szB
205                   , 0
206                   , 1 );
207   }
208 //FCONV 20121216.e
209
210   int szS[3];
211   int headS[3];
212   for(int i=0; i<3; i++) {
213     szS[i]=end[i]-sta[i]+1;
214     headS[i]=sta[i];
215   }
216
217   cio_Array* src = cio_Array::instanceArray
218                   ( DFI_Finfo.DataType
219                   , DFI_Finfo.ArrayShape
220                   , szS
221                   , 0
222                   , DFI_Finfo.Component );
223   src->setHeadIndex( headS );
224
225
226   //data 読込み
227   //if( !read_Datarecord(fp, matchEndian, buf, headB, voxsize[2], src ) ) {
228   ret = read_Datarecord(fp, matchEndian, buf, headB, voxsize[2], src );
229   if( ret != CIO::E_CIO_SUCCESS) {
230     ret = CIO::E_CIO_ERROR_READ_FIELD_DATA_RECORD;
231     fclose(fp);
232     printf("ERROR Data Record Read error!!!!\n");
233     delete buf;
234     return NULL;
235   }
236
237   //read average
238   if( !avr_mode ) {
239     //if( !read_averaged(fp, matchEndian, step, avr_step, avr_time) )
240     ret = read_averaged(fp, matchEndian, step, avr_step, avr_time);
241     if( ret !=CIO::E_CIO_SUCCESS )
242     {
243       ret = CIO::E_CIO_ERROR_READ_FIELD_AVERAGED_RECORD;
244       delete buf;
245       return src;
246     }
247   }
248
249   fclose(fp);
250   delete buf;
251
252   return src;
253
254 }
```

6.3.3.52 **cio_DFI** ∗ cio_DFI::ReadInit (const **MPI_Comm** *comm,* const std::string *dfifile,* const int *G_Voxel[3],* const int *G_Div[3],* **CIO::E_CIO_ERRORCODE** & *ret*) [static]

read インスタンス

引数

in	*comm*	MPI コミュニケータ
in	*dfifile*	DFI ファイル名
in	*G_Voxel*	計算空間全体のボクセルサイズ
in	*G_Div*	計算空間の領域分割数
out	*ret*	終了コード

戻り値

インスタンスされたクラスのポインタ

DFI のディレクトリパスの取得

index.dfi read

TP インスタンス

入力ファイル index.dfi をセット

Fileinfo の読込み

FilePath の読込み

Unit の読込み

TimeSlice の読込み

TextParser の破棄

proc.dfi file name の取得

proc.dfi read

TP インスタンス

入力ファイル proc.dfi をセット

Domain の読込み

MPI の読込み

Process の読込み

TextParser の破棄

dfi のインスタンス

cio_DFI.C の 48 行で定義されています。

参照先 CheckReadType(), CIO::cioPath_ConnectPath(), CIO::cioPath_DirName(), CIO::cioPath_FileName(), DFI_Domain, CIO::E_CIO_DIFFDIV_REFINEMENT, CIO::E_CIO_DIFFDIV_SAMERES, CIO::E_CIO_ERROR_INVALID_DIVNUM, CIO::E_CIO_ERROR_READ_DOMAIN, CIO::E_CIO_ERROR_READ_FILEINFO, CIO::E_CIO_ERROR_READ_FILEPATH, CIO::E_CIO_ERROR_READ_INDEXFILE_OPENERROR, CIO::E_CIO_ERROR_READ_MPI, CIO::E_CIO_ERROR_READ_PROCESS, CIO::E_CIO_ERROR_READ_PROCFILE_OPENERROR, CIO::E_CIO_ERROR_READ_TIMESLICE, CIO::E_CIO_ERROR_READ_UNIT, CIO::E_CIO_ERROR_TEXTPARSER, CIO::E_CIO_FMT_BOV, CIO::E_CIO_FMT_SPH, CIO::E_CIO_READTYPE_UNKNOWN, CIO::E_CIO_SAMEDIV_REFINEMENT, CIO::E_CIO_SAMEDIV_SAMERES, CIO::E_CIO_SUCCESS, cio_FileInfo::FileFormat, cio_TextParser::getTPinstance(), cio_Domain::GlobalDivision, cio_Domain::GlobalVoxel, m_comm, m_indexDfiName, m_RankID, m_read_type, MPI_Comm_rank(), cio_FilePath::ProcDFIFile, cio_FilePath::Read(), cio_MPI::Read(), cio_Domain::Read(), cio_FileInfo::Read(), cio_TimeSlice::Read(), cio_Process::Read(), cio_Unit::Read(), cio_TextParser::readTPfile(), と cio_TextParser::remove().

```
53 {
54
56   std::string dirName = CIO::cioPath_DirName(DfiName);
57
58   int RankID;
59   MPI_Comm_rank( comm, &RankID );
60
61   cio_TextParser tpCntl;
```

```
62
65   tpCntl.getTPinstance();
66
67   FILE*fp = NULL;
68   if( !(fp=fopen(DfiName.c_str(),"rb")) ) {
69     printf("Can't open file. (%s)\n",DfiName.c_str());
70     ret = CIO::E_CIO_ERROR_READ_INDEXFILE_OPENERROR;
71     return NULL;
72   }
73   fclose(fp);
74
76   int ierror = 0;
77   ierror = tpCntl.readTPfile(DfiName);
78   if ( ierror )
79   {
80     printf("\tinput file not found '%s'\n",DfiName.c_str());
81     ret = CIO::E_CIO_ERROR_TEXTPARSER;
82     return NULL;
83   }
84
86   cio_FileInfo F_info;
87   if( F_info.Read(tpCntl) != CIO::E_CIO_SUCCESS )
88   {
89     printf("\tFileInfo Data Read error %s\n",DfiName.c_str());
90     ret = CIO::E_CIO_ERROR_READ_FILEINFO;
91     return NULL;
92   }
93
95   cio_FilePath F_path;
96   if( F_path.Read(tpCntl) != CIO::E_CIO_SUCCESS )
97   {
98     printf("\tFilePath Data Read error %s\n",DfiName.c_str());
99     ret = CIO::E_CIO_ERROR_READ_FILEPATH;
100     return NULL;
101   }
102
104   cio_Unit unit;
105   if( unit.Read(tpCntl) != CIO::E_CIO_SUCCESS )
106   {
107     printf("\tUnit Data Read error %s\n",DfiName.c_str());
108     ret = CIO::E_CIO_ERROR_READ_UNIT;
109     return NULL;
110   }
111
113   cio_TimeSlice TimeSlice;
114   if( TimeSlice.Read(tpCntl) != CIO::E_CIO_SUCCESS )
115   {
116     printf("\tTimeSlice Data Read error %s\n",DfiName.c_str());
117     ret = CIO::E_CIO_ERROR_READ_TIMESLICE;
118     return NULL;
119   }
120
122   tpCntl.remove();
123
125   std::string dfiname  = CIO::cioPath_FileName(F_path.ProcDFIFile,".dfi");
126   std::string procfile = CIO::cioPath_ConnectPath(dirName,dfiname);
127
130   tpCntl.getTPinstance();
131
132   fp = NULL;
133   if( !(fp=fopen(procfile.c_str(),"rb")) ) {
134     printf("Can't open file. (%s)\n",procfile.c_str());
135     ret = CIO::E_CIO_ERROR_READ_PROCFILE_OPENERROR;
136     return NULL;
137   }
138   fclose(fp);
139
141   ierror = tpCntl.readTPfile(procfile);
142   if ( ierror )
143   {
144     printf("\tinput file not found '%s'\n",procfile.c_str());
145     ret = CIO::E_CIO_ERROR_TEXTPARSER;
146     return NULL;
147   }
148
150   cio_Domain domain;
151   if( domain.Read(tpCntl) != CIO::E_CIO_SUCCESS )
152   {
153     printf("\tDomain Data Read error %s\n",procfile.c_str());
154     ret = CIO::E_CIO_ERROR_READ_DOMAIN;
155     return NULL;
156   }
157
159   cio_MPI mpi;
160   if( mpi.Read(tpCntl,domain) != CIO::E_CIO_SUCCESS )
161   {
162     printf("\tMPI Data Read error %s\n",procfile.c_str());
```

```
163     ret = CIO::E_CIO_ERROR_READ_MPI;
164     return NULL;
165   }
166
168   cio_Process process;
169   if( process.Read(tpCntl) != CIO::E_CIO_SUCCESS )
170   {
171     printf("\tProcess Data Read error %s\n",procfile.c_str());
172     ret = CIO::E_CIO_ERROR_READ_PROCESS;
173     return NULL;
174   }
175
177   tpCntl.remove();
178
180   cio_DFI *dfi = NULL;
181   if( F_info.FileFormat == CIO::E_CIO_FMT_SPH ) {
182     dfi = new cio_DFI_SPH(F_info, F_path, unit, domain, mpi, TimeSlice, process);
183   } else if( F_info.FileFormat == CIO::E_CIO_FMT_BOV ) {
184     dfi = new cio_DFI_BOV(F_info, F_path, unit, domain, mpi, TimeSlice, process);
185   } else {
186     return NULL;
187   }
188
189   //読込みタイプのチェック
190   dfi->m_read_type = dfi->CheckReadType(G_Voxel,dfi->DFI_Domain.GlobalVoxel,
191                                         G_Div,dfi->DFI_Domain.GlobalDivision);
192   if( dfi->m_read_type == CIO::E_CIO_READTYPE_UNKNOWN ) {
193     //printf("\tDimension size error (%d %d %d)\n",
194     //       G_Voxel[0], G_Voxel[1], G_Voxel[2]);
195     ret = CIO::E_CIO_ERROR_INVALID_DIVNUM;
196     dfi->m_comm = comm;
197     dfi->m_indexDfiName = DfiName;
198     dfi->m_RankID = RankID;
199     return dfi;
200   }
201
202 #if 0
203   if( dfi->m_start_type == E_CIO_SAMEDIV_SAMERES ) {
204     printf("***** SAMEDIV_SAMERES\n");
205   } else if( dfi->m_start_type == E_CIO_SAMEDIV_REFINEMENT ) {
206     printf("***** SAMEDIV_REFINEMENT\n");
207   } else if( dfi->m_start_type == E_CIO_DIFFDIV_SAMERES ) {
208     printf("***** DIFFDIV_SAMERES\n");
209   } else if( dfi->m_start_type == E_CIO_DIFFDIV_REFINEMENT ) {
210     printf("***** DIFFDIV_REFINEMENT\n");
211   }
212 #endif
213
214   dfi->m_comm = comm;
215   dfi->m_indexDfiName = DfiName;
216   dfi->m_RankID = RankID;
217
218   ret = CIO::E_CIO_SUCCESS;
219
220   return dfi;
221
222 }
```

6.3.3.53 void cio_DFI::set_output_fname (CIO::E_CIO_OUTPUT_FNAME *output_fname*) [inline]

出力ファイル命名規約 (step_rank,rank_step) をセット

引数

in	*output_fname*	出力ファイル命名規約

cio_DFI.h の 310 行で定義されています。

参照先 m_output_fname.

```
311   { m_output_fname = output_fname; };
```

6.3.3.54 void cio_DFI::set_output_type (CIO::E_CIO_OUTPUT_TYPE *output_type*) [inline]

出力形式 (ascii,binary,FortranBinary) をセット

引数

in	*output_type*	出力形式

cio_DFI.h の 303 行で定義されています。

参照先 m_output_type.

```
304   {  m_output_type = output_type; };
```

6.3.3.55 void cio_DFI::set_RankID (const int *rankID*) [inline]

RankID をセットする

引数

in	*rankID*	RankID

cio_DFI.h の 297 行で定義されています。

参照先 m_RankID.

```
298   { m_RankID = rankID; };
```

6.3.3.56 void cio_DFI::SetcioDomain (cio_Domain *domain*)

cio_Domain クラスのセット

cio_DFI.C の 273 行で定義されています。

参照先 DFI_Domain.

```
274 {
275   DFI_Domain = domain;
276 }
```

6.3.3.57 void cio_DFI::SetcioFilePath (cio_FilePath *FPath*)

cio_FilePath クラスのセット

cio_DFI.C の 234 行で定義されています。

参照先 DFI_Fpath.

```
235 {
236   DFI_Fpath = FPath;
237 }
```

6.3.3.58 void cio_DFI::SetcioMPI (cio_MPI *mpi*)

cio_MPI クラスセット

cio_DFI.C の 290 行で定義されています。

参照先 DFI_MPI.

```
291 {
292   DFI_MPI = mpi;
293 }
```

6.3.3.59 void cio_DFI::SetcioProcess (cio_Process *Process*)

cio_Process クラスセット

cio_DFI.C の 321 行で定義されています。

参照先 DFI_Process.

```
322 {
323   DFI_Process = Process;
324 }
```

6.3.3.60 void cio_DFI::SetcioTimeSlice (cio_TimeSlice *TSlice*)

cio_TimeSlice クラスセット

cio_DFI.C の 305 行で定義されています。

参照先 DFI_TimeSlice.

```
306 {
307   DFI_TimeSlice = TSlice;
308 }
```

6.3.3.61 void cio_DFI::SetcioUnit (cio_Unit *unit*)

cio_Unit クラスのセット

cio_DFI.C の 257 行で定義されています。

参照先 DFI_Unit.

```
258 {
259   DFI_Unit = unit;
260 }
```

6.3.3.62 void cio_DFI::setComponentVariable (int *pcomp*, std::string *compName*)

FileInfo の成分名を登録する

引数

in	*pcomp*	成分位置 0:u, 1:v, 2:w
in	*compName*	成分名 "u","v","w",,,

cio_DFI.C の 1027 行で定義されています。

参照先 DFI_Finfo, と cio_FileInfo::setComponentVariable().

```
1028 {
1029
1030   DFI_Finfo.setComponentVariable(pcomp, compName);
1031
1032 }
```

6.3.3.63 template<class T1 , class T2 > CIO_INLINE bool cio_DFI::setGridData (cio_TypeArray< T1 > * *P*, cio_TypeArray< T2 > * *S*)

<0,0,0>

<1,0,0>

<1,0,1>

<0,0,1>

<0,1,0>

<1,1,0>

<1,1,1>

<0,1,1>

<0,0,0>

<1,0,0>

<1,0,1>

<0,0,1>

<0,1,0>

<1,1,0>

<1,1,1>

<0,1,1>

cio_DFI_inline.h の 184 行で定義されています。

参照先 CIO::E_CIO_NIJK, cio_Array::getArrayShape(), cio_Array::getArraySizeInt(), cio_TypeArray< T >::getData(), cio_Array::getNcompInt(), cio_TypeArray< T >::val(), と VolumeDataDivide().

```
186 {
187
188   if( P->getArrayShape() != S->getArrayShape() ) return false;
189
190   //成分数をセット
191   if( P->getNcompInt() != S->getNcompInt() ) return false;
192   int nComp = P->getNcompInt();
193
194   //S(セル中心)の配列サイズを取得セット
195   //T2* data = S->getData();
196   const int* size = S->getArraySizeInt();
197   int ix = size[0];
198   int jx = size[1];
199   int kx = size[2];
200
201   //P(格子点)の配列サイズを取得セット
202   T1* d    = P->getData();
203   const int* Psz = P->getArraySizeInt();
204   int id = Psz[0];
205   int jd = Psz[1];
206   int kd = Psz[2];
207
208   //P の配列をゼロクリア
209   size_t dsize = (size_t)(id*jd*kd*nComp);
210   for (size_t l=0; l<dsize; l++) d[l]=0.0;
211
212   //S(セル中心)のデータを P(格子点)に加える
213   //NIJK の処理
214   if( P->getArrayShape() == CIO::E_CIO_NIJK ) {
215     for (int km=0; km<kx; km++) {
216     for (int jm=0; jm<jx; jm++) {
217     for (int im=0; im<ix; im++) {
218     for (int n=0; n<nComp; n++) {
219       P->val(n, im  ,jm  ,km  ) = P->val(n, im  ,jm  ,km  )+S->val(n, im,jm,km);
220       P->val(n, im+1,jm  ,km  ) = P->val(n, im+1,jm  ,km  )+S->val(n, im,jm,km);
221       P->val(n, im+1,jm  ,km+1) = P->val(n, im+1,jm  ,km+1)+S->val(n, im,jm,km);
222       P->val(n, im  ,jm  ,km+1) = P->val(n, im  ,jm  ,km+1)+S->val(n, im,jm,km);
223       P->val(n, im  ,jm+1,km  ) = P->val(n, im  ,jm+1,km  )+S->val(n, im,jm,km);
224       P->val(n, im+1,jm+1,km  ) = P->val(n, im+1,jm+1,km  )+S->val(n, im,jm,km);
225       P->val(n, im+1,jm+1,km+1) = P->val(n, im+1,jm+1,km+1)+S->val(n, im,jm,km);
226       P->val(n, im  ,jm+1,km+1) = P->val(n, im  ,jm+1,km+1)+S->val(n, im,jm,km);
227     }}}}
228   } else {
229   //IJKN の処理
230     for (int n=0; n<nComp; n++) {
231     for (int km=0; km<kx; km++) {
232     for (int jm=0; jm<jx; jm++) {
233     for (int im=0; im<ix; im++) {
234       P->val(im  ,jm  ,km  ,n) = P->val(im  ,jm  ,km  ,n)+S->val(im,jm,km,n);
235       P->val(im+1,jm  ,km  ,n) = P->val(im+1,jm  ,km  ,n)+S->val(im,jm,km,n);
```

```
236       P->val(im+1,jm  ,km+1,n) = P->val(im+1,jm  ,km+1,n)+S->val(im,jm,km,n);
237       P->val(im  ,jm  ,km+1,n) = P->val(im  ,jm  ,km+1,n)+S->val(im,jm,km,n);
238       P->val(im  ,jm+1,km  ,n) = P->val(im  ,jm+1,km  ,n)+S->val(im,jm,km,n);
239       P->val(im+1,jm+1,km  ,n) = P->val(im+1,jm+1,km  ,n)+S->val(im,jm,km,n);
240       P->val(im+1,jm+1,km+1,n) = P->val(im+1,jm+1,km+1,n)+S->val(im,jm,km,n);
241       P->val(im  ,jm+1,km+1,n) = P->val(im  ,jm+1,km+1,n)+S->val(im,jm,km,n);
242     }}}}
243   }
244
245   //内部の格子点のデータを重み付けで割る
246   VolumeDataDivide(P);
247
248   return true;
249
250 }
```

6.3.3.64 template<class T1 , class T2 > bool cio_DFI::setGridData (cio_TypeArray< T1 > * *P*, cio_TypeArray< T2 > * *S*)

セル中心データを格子点に値をセット

引数

out	*P*	格子点データ
in	*S*	セル中心 data

参照元 WriteFieldData().

6.3.3.65 void cio_DFI::setIntervalStep (int *interval_step*, int *base_step* = 0, int *start_step* = 0, int *last_step* = -1)

出力インターバルステップの登録

登録しない（本メソッドがコールされない）場合はCIO でのインターバル 制御は行わない

引数

in	*interval_step*	インターバルステップ
in	*base_step*	基準となるステップ（デフォルト 0 ステップ）
in	*start_step*	セッション開始ステップ（デフォルト 0 ステップ）
in	*last_step*	セッション最終ステップ（デフォルト、-1：最終ステップで出力しない）

6.3.3.66 void cio_DFI::setIntervalTime (double *interval_time*, double *dt*, double *base_time* = 0.0, double *start_time* = 0.0, double *last_time* = -1.0)

インターバルタイムの登録

引数

in	*interval_time*	出力インターバルタイム
in	*dt*	計算の時間間隔
in	*base_time*	基準となるタイム（デフォルト 0.0 タイム）
in	*start_time*	セッション開始タイム（デフォルト 0.0 タイム）
in	*last_time*	せっしょん最終タイム（デフォルト、-1.0：最終タイムで出力しない）

6.3.3.67 void cio_DFI::SetTimeSliceFlag (const CIO::E_CIO_ONOFF *ONOFF*)

TimeSlice OnOff フラグをセットする

引数

in	*ONOFF*	

cio_DFI.C の 1020 行で定義されています。

参照先 DFI_Finfo, と cio_FileInfo::TimeSliceDirFlag.

```
1021 {
1022   DFI_Finfo.TimeSliceDirFlag = ONOFF;
1023 }
```

6.3.3.68 template<class T > CIO_INLINE void cio_DFI::VolumeDataDivide (cio_TypeArray< T > * *P*)

cio_DFI_inline.h の 257 行で定義されています。

参照先 CIO::E_CIO_NIJK, cio_Array::getArrayShape(), cio_Array::getArraySizeInt(), cio_Array::getNcompInt(), と cio_TypeArray< T >::val().

```
258 {
259   int i,j,k,n;
260   const int* szP = P->getArraySizeInt();
261   int id = szP[0];
262   int jd = szP[1];
263   int kd = szP[2];
264
265   int ncomp = P->getNcompInt();
266
267   //NIJK
268   if( P->getArrayShape() == CIO::E_CIO_NIJK ) {
269
270     //I
271     for (k=0; k<kd;    k++){
272     for (j=0; j<jd;    j++){
273     for (i=1; i<id-1;  i++){
274     for (n=0; n<ncomp; n++){
275       P->val(n,i,j,k) = P->val(n,i,j,k)*0.5;
276     }}}}
277
278     //J
279     for (k=0; k<kd;    k++){
280     for (j=1; j<jd-1;  j++){
281     for (i=0; i<id;    i++){
282     for (n=0; n<ncomp; n++){
283       P->val(n,i,j,k) = P->val(n,i,j,k)*0.5;
284     }}}}
285
286     //K
287     for (k=1; k<kd-1;  k++){
288     for (j=0; j<jd;    j++){
289     for (i=0; i<id;    i++){
290     for (n=0; n<ncomp; n++){
291       P->val(n,i,j,k) = P->val(n,i,j,k)*0.5;
292     }}}}
293
294   //IJKN
295   } else {
296
297     //I
298     for (n=0; n<ncomp; n++){
299     for (k=0; k<kd;    k++){
300     for (j=0; j<jd;    j++){
301     for (i=1; i<id-1;  i++){
302       P->val(i,j,k,n) = P->val(i,j,k,n)*0.5;
303     }}}}
304
305     //J
306     for (n=0; n<ncomp; n++){
307     for (k=0; k<kd;    k++){
308     for (j=1; j<jd-1;  j++){
309     for (i=0; i<id;    i++){
310       P->val(i,j,k,n) = P->val(i,j,k,n)*0.5;
311     }}}}
312
313     //K
314     for (n=0; n<ncomp; n++){
315     for (k=1; k<kd-1;  k++){
316     for (j=0; j<jd;    j++){
317     for (i=0; i<id;    i++){
```

```
318         P->val(i,j,k,n) = P->val(i,j,k,n)*0.5;
319       }}}}
320
321     }
322 }
```

6.3.3.69 template<class T > void cio_DFI::VolumeDataDivide (cio_TypeArray< T > * *P*)

内部の格子点のデータを重み付けでで割る

引数

out	*P*	格子点 data

参照元 setGridData().

6.3.3.70 virtual bool cio_DFI::write_ascii_header (const unsigned *step*, const double *time*) [inline], [protected], [virtual]

ascii ヘッダーレコード出力 (bov,avs)

引数

in	*step*	step 番号
in	*time*	time

cio_DFI_AVS, と cio_DFI_BOVで再定義されています。

cio_DFI.h の 881 行で定義されています。

参照元 WriteData().

```
883   { return true; };
```

6.3.3.71 virtual CIO::E_CIO_ERRORCODE cio_DFI::write_averaged (FILE * *fp*, const unsigned *step_avr*, const double *time_avr*) [protected], [pure virtual]

Average レコードの出力

引数

in	*fp*	ファイルポインタ
in	*step_avr*	平均ステップ番号
in	*time_avr*	平均時刻

戻り値

true:出力成功 false:出力失敗

cio_DFI_PLOT3D, cio_DFI_SPH, cio_DFI_AVS, cio_DFI_VTK, と cio_DFI_BOVで実装されています。

参照元 WriteFieldData().

6.3.3.72 virtual CIO::E_CIO_ERRORCODE cio_DFI::write_DataRecord (FILE * *fp*, cio_Array * *val*, const int *gc*, const int *RankID*) [protected], [pure virtual]

SPH データレコードの出力

引数

in	*fp*	ファイルポインタ
in	*val*	データポインタ
in	*gc*	ガイドセル
in	*RankID*	ランク番号

戻り値

true:出力成功 false:出力失敗

cio_DFI_PLOT3D, cio_DFI_SPH, cio_DFI_AVS, cio_DFI_VTK, と cio_DFI_BOVで実装されています。

参照元 WriteFieldData().

6.3.3.73 virtual CIO::E_CIO_ERRORCODE cio_DFI::write_HeaderRecord (FILE * *fp*, const unsigned *step*, const double *time*, const int *RankID*) [protected], [pure virtual]

SPH ヘッダファイルの出力

引数

in	*fp*	ファイルポインタ
in	*step*	ステップ番号
in	*time*	時刻
in	*RankID*	ランク番号

戻り値

true:出力成功 false:出力失敗

cio_DFI_PLOT3D, cio_DFI_SPH, cio_DFI_AVS, cio_DFI_VTK, と cio_DFI_BOVで実装されています。

参照元 WriteFieldData().

6.3.3.74 template<class T , class TimeT , class TimeAvrT > CIO_INLINE CIO::E_CIO_ERRORCODE cio_DFI::WriteData (const unsigned *step*, TimeT *time*, const int *sz[3]*, const int *nComp*, const int *gc*, T * *val*, T * *minmax*, const bool *avr_mode*, const unsigned *step_avr*, TimeAvrT *time_avr*)

cio_DFI_inline.h の 129 行で定義されています。

参照先 cio_FileInfo::ArrayShape, cio_FileInfo::Component, DFI_Finfo, DFI_Process, cio_Array::instanceArray(), m-_RankID, cio_Process::RankList, と WriteData().

```
139 {
140
141   cio_Array *data = cio_Array::instanceArray
142                     ( val
143                     , DFI_Finfo.ArrayShape
144                     , DFI_Process.RankList[m_RankID].VoxelSize[0]
145                     , DFI_Process.RankList[m_RankID].VoxelSize[1]
146                     , DFI_Process.RankList[m_RankID].VoxelSize[2]
147                     , gc
148                     , DFI_Finfo.Component);
149
150   double d_time = (double)time;
151   double d_time_avr = (double)time_avr;
152   double *d_minmax=NULL;
153   if( minmax ) {
154     if( DFI_Finfo.Component>1 ) {
155       d_minmax = new double[DFI_Finfo.Component*2+2];
156       for(int i=0; i<DFI_Finfo.Component*2+2; i++) {
157         d_minmax[i] = minmax[i];
158       }
159     } else {
160       d_minmax = new double[2];
```

```
161       d_minmax[0] = minmax[0];
162       d_minmax[1] = minmax[1];
163     }
164   }
165
166   CIO::E_CIO_ERRORCODE ret;
167   ret = WriteData(step, gc, d_time, data, d_minmax, avr_mode, step_avr, d_time_avr);
168
169   //val = (T*)data->getData(true);
170   //data->getData(true);
171
172   if( d_minmax ) delete [] d_minmax;
173
174   delete data;
175   return ret;
176
177 }
```

6.3.3.75 template<class T , class TimeT , class TimeAvrT > CIO::E_CIO_ERRORCODE cio_DFI::WriteData (const unsigned *step*, TimeT *time*, const int *sz[3]*, const int *nComp*, const int *gc*, T * *val*, T * *minmax* = NULL, bool *avr_mode* = true, unsigned *step_avr* = 0, TimeAvrT *time_avr* = 0.0)

write field data record (template function)

スカラーのとき、minmax[0]=min minmax[1]=max ベクトルのとき、minmax[0] =成分 1 の minX minmax[1] = 成分 1 の maxX ... minmax[2n-2]=成分 n の minX minmax[2n-1]=成分 n の maxX minmax[2n]=合成値の min minmax[2n+1]=合成値の max

引数

in	*step*	出力ステップ番号
in	*time*	出力時刻
in	*sz*	val の実ボクセルサイズ
in	*nComp*	val の成分数（1or3)
in	*gc*	val の仮想セル数
in	*val*	出力データポインタ
in	*minmax*	フィールデータのMinMax
in	*avr_mode*	平均ステップ&時間出力　false : 出力 true : 出力しない
in	*step_avr*	平均ステップ
in	*time_avr*	平均時間

参照元 WriteData().

6.3.3.76 CIO::E_CIO_ERRORCODE cio_DFI::WriteData (const unsigned *step*, const int *gc*, double *time*, cio_Array * *val*, double * *minmax*, const bool *avr_mode*, const unsigned *step_avr*, double *time_avr*)

write field data record

template WriteData 関数で方に応じた配列を確保した後、呼び出される

引数

in	*step*	出力ステップ番号
in	*gc*	仮想セル数
in	*time*	出力時刻
in	*val*	出力データポインタ
in	*minmax*	フィールデータのMinMax
in	*avr_mode*	平均ステップ&時間出力　false : 出力 true : 出力しない
in	*step_avr*	平均ステップ

in	*time_avr*	平均時間

cio_DFI_Write.C の 207 行で定義されています。

参照先 cio_TimeSlice::AddSlice(), cio_FileInfo::ArrayShape, CIO::cioPath_ConnectPath(), CIO::cioPath_DirName(), CIO::cioPath_FileName(), CIO::cioPath_isAbsolute(), cio_FileInfo::Component, cio_Array::copyArray(), D_CIO_EXT_BOV, D_CIO_EXT_FUNC, D_CIO_EXT_SPH, D_CIO_EXT_VTK, cio_FileInfo::DataType, DFI_Finfo, DFI_MPI, DFI_Process, DFI_TimeSlice, cio_FileInfo::DirectoryPath, CIO::E_CIO_ERROR, CIO::E_CIO_ERROR_MAKEDIRECTORY, CIO::E_CIO_FMT_AVS, CIO::E_CIO_FMT_BOV, CIO::E_CIO_FMT_PLOT3D, CIO::E_CIO_FMT_SPH, CIO::E_CIO_FMT_VTK, CIO::E_CIO_FNAME_RANK_STEP, CIO::E_CIO_SUCCESS, cio_FileInfo::FileFormat, Generate_FieldFileName(), Generate_FileName(), cio_FileInfo::GuideCell, cio_Array::instanceArray(), m_directoryPath, m_indexDfiName, m_output_fname, m_RankID, MakeDirectory(), cio_MPI::NumberOfRank, cio_FileInfo::Prefix, cio_Process::RankList, cio_FileInfo::TimeSliceDirFlag, write_ascii_header(), WriteFieldData(), と WriteIndexDfiFile().

```
215 {
216
217   //printf("WriteData RankID : %d\n",m_RankID);
218
219   bool mio=false;
220   if( DFI_MPI.NumberOfRank > 1 ) mio=true;
221
222   std::string outFile,tmp;
223 //FCONV 20131128.s
224   if( m_output_fname != CIO::E_CIO_FNAME_RANK_STEP ) {
225     tmp = Generate_FieldFileName(m_RankID,step,mio);
226     if( CIO::cioPath_isAbsolute(DFI_Finfo.DirectoryPath) ){
227       outFile = tmp;
228     } else {
229       outFile = m_directoryPath + "/"+ tmp;
230     }
231   } else {
232     std::string ext;
233     if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_SPH ) {
234       ext = D_CIO_EXT_SPH;
235     } else if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_BOV ) {
236       ext = D_CIO_EXT_BOV;
237     } else if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_AVS ) {
238       //ext = D_CIO_EXT_SPH;
239       ext = D_CIO_EXT_BOV;
240     } else if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_VTK ) {
241       ext = D_CIO_EXT_VTK;
242     } else if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_PLOT3D ) {
243       ext = D_CIO_EXT_FUNC;
244     }
245     tmp = Generate_FileName(DFI_Finfo.Prefix,
246                             m_RankID,
247                             step,ext,
248                             m_output_fname,
249                             mio,
250                             DFI_Finfo.TimeSliceDirFlag);
251     if( CIO::cioPath_isAbsolute(DFI_Finfo.DirectoryPath) ){
252       outFile = DFI_Finfo.DirectoryPath +"/"+ tmp;
253     } else {
254       outFile = m_directoryPath + "/" + DFI_Finfo.DirectoryPath +"/"+ tmp;
255     }
256   }
257 //FCONV 20131128.e
258
259   std::string dir = CIO::cioPath_DirName(outFile);
260
261   if( MakeDirectory(dir) != 1 ) return CIO::E_CIO_ERROR_MAKEDIRECTORY;
262
263   cio_Array *outArray = val;
264   if( gc != DFI_Finfo.GuideCell ) {
265     //出力用バッファのインスタンス
266     outArray = cio_Array::instanceArray
267                ( DFI_Finfo.DataType
268                , DFI_Finfo.ArrayShape
269                , DFI_Process.RankList[m_RankID].VoxelSize
270                , DFI_Finfo.GuideCell
271                , DFI_Finfo.Component);
272     //配列のコピー val -> outArray
273     int ret = val->copyArray(outArray);
274   }
275
276   // フィールドデータの出力
277   CIO::E_CIO_ERRORCODE err = CIO::E_CIO_SUCCESS;
278   err = WriteFieldData(outFile, step, time, outArray, avr_mode, step_avr, time_avr);
279
280   //出力バッファのメモリ解放
```

```
281   if( val != outArray ) {
282     delete outArray;
283   }
284
285   if( err != CIO::E_CIO_SUCCESS ) return err;
286
287 //FCONV 20131218.s
288   if( m_indexDfiName != "" ) {
289     //index dfi ファイルのディレクトリ作成
290     cio_DFI::MakeDirectory(m_directoryPath);
291     std::string dfiname = CIO::cioPath_FileName(m_indexDfiName,".dfi");
292     std::string fname = CIO::cioPath_ConnectPath( m_directoryPath, dfiname );
293
294     //Slice へのセット
295     DFI_TimeSlice.AddSlice(step, time, minmax, DFI_Finfo.Component, avr_mode,
296                            step_avr, time_avr);
297
298     //index dfi のファイル出力
299     if( m_RankID == 0 ) {
300       err = WriteIndexDfiFile(fname);
301     }
302   }
303 //FCONV 20131218.e
304 //FCONV 20131125.s
305   if( !write_ascii_header(step,time) ) return CIO::E_CIO_ERROR;
306 //FCONV 20131125.e
307
308   return err;
309 }
```

6.3.3.77 CIO::E_CIO_ERRORCODE cio_DFI::WriteFieldData (std::string *fname*, const unsigned *step*, double *time*, cio_Array * *val*, const bool *mode*, const unsigned *step_avr*, const double *time_avr*) [protected], [virtual]

write field data record (double)

引数

in	*fname*	出力フィールドファイル名
in	*step*	出力ステップ番号
in	*time*	出力時刻
in	*val*	出力データポインタ
in	*mode*	平均ステップ&時間出力　false : 出力 true : 出力しない
in	*step_avr*	平均ステップ
in	*time_avr*	平均時間

戻り値

error code

cio_DFI_Write.C の 314 行で定義されています。

参照先 DFI_Finfo, CIO::E_CIO_ERROR_OPEN_FIELDDATA, CIO::E_CIO_ERROR_WRITE_FIELD_AVERAGED_RECORD, CIO::E_CIO_ERROR_WRITE_FIELD_DATA_RECORD, CIO::E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_INT16, CIO::E_CIO_INT32, CIO::E_CIO_INT8, CIO::E_CIO_SUCCESS, cio_Array::getArrayShape(), cio_Array::getArraySizeInt(), cio_Array::getDataType(), cio_Array::getNcomp(), cio_FileInfo::GuideCell, cio_Array::instanceArray(), m_bgrid_interp_flag, m_RankID, setGridData(), write_averaged(), write_DataRecord(), と write_HeaderRecord().

参照元 WriteData().

```
321 {
322
323   FILE* fp;
324   if( (fp = fopen(fname.c_str(),"wb")) == NULL ) {
325     fprintf(stderr,"Can't open file.(%s)\n",fname.c_str());
326     return CIO::E_CIO_ERROR_OPEN_FIELDDATA;
327   }
328
329   //printf("field file name : %s\n",fname.c_str());
330
```

```
331   //ヘッダー出力
332   if( write_HeaderRecord(fp, step, time, m_RankID) != CIO::E_CIO_SUCCESS ) {
333     fclose(fp);
334     return CIO::E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD;
335   }
336
337   cio_Array *outArray = val;
338
339   //格子点補間処理ありの場合、図心データから格子点への補間を行う
340   if( m_bgrid_interp_flag ) {
341     //配列サイズの取得
342     const int *szVal = val->getArraySizeInt();
343     //配列成分の取得
344     int nComp = val->getNcomp();
345     //格子点データ配列サイズのセット
346     int szOut[3];
347     for(int i=0; i<3; i++) szOut[i]=szVal[i]+1;
348     //出力バッファのインスタンス
349     outArray =  cio_Array::instanceArray
350                           (val->getDataType(),
351                            val->getArrayShape(),
352                            szOut,
353                            0,
354                            nComp);
355
356     //char
357     if( val->getDataType() == CIO::E_CIO_INT8 ) {
358       cio_TypeArray<char> *V = dynamic_cast<cio_TypeArray<char>*>(val);
359       cio_TypeArray<char> *P = dynamic_cast<cio_TypeArray<char>*>(outArray);
360       setGridData(P,V);
361     //short
362     } else if( val->getDataType() == CIO::E_CIO_INT16 ) {
363       cio_TypeArray<short> *V = dynamic_cast<cio_TypeArray<short>*>(val);
364       cio_TypeArray<short> *P = dynamic_cast<cio_TypeArray<short>*>(outArray);
365       setGridData(P,V);
366     //int
367     } else if( val->getDataType() == CIO::E_CIO_INT32 ) {
368       cio_TypeArray<int> *V = dynamic_cast<cio_TypeArray<int>*>(val);
369       cio_TypeArray<int> *P = dynamic_cast<cio_TypeArray<int>*>(outArray);
370       setGridData(P,V);
371     //float
372     } else if( val->getDataType() == CIO::E_CIO_FLOAT32 ) {
373       cio_TypeArray<float> *V = dynamic_cast<cio_TypeArray<float>*>(val);
374       cio_TypeArray<float> *P = dynamic_cast<cio_TypeArray<float>*>(outArray);
375       setGridData(P,V);
376     //double
377     } else if( val->getDataType() == CIO::E_CIO_FLOAT64 ) {
378       cio_TypeArray<double> *V = dynamic_cast<cio_TypeArray<double>*>(val);
379       cio_TypeArray<double> *P = dynamic_cast<cio_TypeArray<double>*>(outArray);
380       setGridData(P,V);
381     }
382
383   }
384
385   //型変換
386   //ここに追加予定
387
388
389   //データ出力
390   //if( write_DataRecord(fp, val, DFI_Finfo.GuideCell, m_RankID) != CIO::E_CIO_SUCCESS) {
391   if( write_DataRecord(fp, outArray, DFI_Finfo.GuideCell, m_RankID) !=
      CIO::E_CIO_SUCCESS) {
392     fclose(fp);
393     return CIO::E_CIO_ERROR_WRITE_FIELD_DATA_RECORD;
394   }
395
396   //average 出力
397   if( !avr_mode ) {
398     if( write_averaged(fp, step_avr, time_avr) != CIO::E_CIO_SUCCESS ) {
399       fclose(fp);
400       return CIO::E_CIO_ERROR_WRITE_FIELD_AVERAGED_RECORD;
401     }
402   }
403
404   fclose(fp);
405
406   return CIO::E_CIO_SUCCESS;
407
408 }
```

6.3.3.78 CIO::E_CIO_ERRORCODE cio_DFI::WriteIndexDfiFile (const std::string *dfi_name*) [protected]

index DFI ファイル出力

引数

in	*dfi_name*	DFI ファイル名

戻り値

true:出力成功 false:出力失敗

cio_DFI_Write.C の 20 行で定義されています。

参照先 DFI_Finfo, DFI_Fpath, DFI_TimeSlice, DFI_Unit, CIO::E_CIO_ERROR_WRITE_FILEINFO, CIO::E_CIO_ERROR_WRITE_FILEPATH, CIO::E_CIO_ERROR_WRITE_INDEXFILE_OPENERROR, CIO::E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY, CIO::E_CIO_ERROR_WRITE_PREFIX_EMPTY, CIO::E_CIO_ERROR_WRITE_TIMESLICE, CIO::E_CIO_ERROR_WRITE_UNIT, CIO::E_CIO_SUCCESS, cio_FileInfo::Prefix, cio_FilePath::Write(), cio_FileInfo::Write(), cio_TimeSlice::Write(), と cio_Unit::Write().

参照元 WriteData().

```
21 {
22
23   if ( dfi_name.empty() ) return CIO::E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY;
24   if ( DFI_Finfo.Prefix.empty() ) return CIO::E_CIO_ERROR_WRITE_PREFIX_EMPTY;
25
26   FILE* fp = NULL;
27
28   // File exist ?
29   bool flag = false;
30   if ( fp = fopen(dfi_name.c_str(), "r") )
31   {
32     flag = true;
33     fclose(fp);
34   }
35
36   if( !(fp = fopen(dfi_name.c_str(), "w")) )
37   {
38     fprintf(stderr, "Can't open file.(%s)\n", dfi_name.c_str());
39     return CIO::E_CIO_ERROR_WRITE_INDEXFILE_OPENERROR;
40   }
41
42   //FileInfo {} の出力
43   if( DFI_Finfo.Write(fp, 0) != CIO::E_CIO_SUCCESS )
44   {
45     fclose(fp);
46     return CIO::E_CIO_ERROR_WRITE_FILEINFO;
47   }
48
49   //FilePath {} の出力
50   if( DFI_Fpath.Write(fp, 1) != CIO::E_CIO_SUCCESS )
51   {
52     fclose(fp);
53     return CIO::E_CIO_ERROR_WRITE_FILEPATH;
54   }
55
56
57   //Unit {} の出力
58   if( DFI_Unit.Write(fp, 0) != CIO::E_CIO_SUCCESS )
59   {
60     fclose(fp);
61     return CIO::E_CIO_ERROR_WRITE_UNIT;
62   }
63
64   //TimeSlice {} の出力
65   if ( DFI_TimeSlice.Write(fp, 1) != CIO::E_CIO_SUCCESS )
66   {
67     fclose(fp);
68     return CIO::E_CIO_ERROR_WRITE_TIMESLICE;
69   }
70
71   return CIO::E_CIO_SUCCESS;
72
73 }
```

6.3.3.79 **cio_DFI** ∗ cio_DFI::WriteInit (const **MPI_Comm** *comm,* const std::string *DfiName,* const std::string *Path,* const std::string *prefix,* const **CIO::E_CIO_FORMAT** *format,* const int *GCell,* const **CIO::E_CIO_DTYPE** *DataType,* const **CIO::E_CIO_ARRAYSHAPE** *ArrayShape,* const int *nComp,* const std::string *proc_fname,* const int *G_size[3],* const float *pitch[3],* const float *G_origin[3],* const int *division[3],* const int *head[3],* const int *tail[3],* const std::string *hostname,* const **CIO::E_CIO_ONOFF** *TSliceOnOff*) [static]

write インスタンス float 型

引数

in	*comm*	MPI コミュニケータ
in	*DfiName*	DFI ファイル名
in	*Path*	フィールドデータのディレクトリ
in	*prefix*	ベースファイル名
in	*format*	ファイルフォーマット
in	*GCell*	出力仮想セル数
in	*DataType*	データタイプ
in	*ArrayShape*	配列形状
in	*nComp*	成分数
in	*proc_fname*	proc.dfi ファイル名
in	*G_size*	グローバルボクセルサイズ
in	*pitch*	ピッチ
in	*G_origin*	原点座標値
in	*division*	領域分割数
in	*head*	計算領域の開始位置
in	*tail*	計算領域の終了位置
in	*hostname*	ホスト名
in	*TSliceOnOff*	TimeSlice フラグ

戻り値

インスタンスされたクラスのポインタ

cio_DFI.C の 328 行で定義されています。

```
346 {
347
348   // float 型を double 型に変換して double 版 WriteInit 関数を呼ぶ
349
350   double d_pch[3],d_org[3];
351   for(int i=0; i<3; i++) {
352     d_pch[i]=(double)pitch[i];
353     d_org[i]=(double)G_origin[i];
354   }
355
356   return WriteInit(comm,
357                    DfiName,
358                    Path,
359                    prefix,
360                    format,
361                    GCell,
362                    DataType,
363                    ArrayShape,
364                    nComp,
365                    proc_fname,
366                    G_size,
367                    d_pch,
368                    d_org,
369                    division,
370                    head,
371                    tail,
372                    hostname,
373                    TSliceOnOff);
374
375 }
```

6.3.3.80 cio_DFI * cio_DFI::WriteInit (const MPI_Comm *comm*, const std::string *DfiName*, const std::string *Path*, const std::string *prefix*, const CIO::E_CIO_FORMAT *format*, const int *GCell*, const CIO::E_CIO_DTYPE *DataType*, const CIO::E_CIO_ARRAYSHAPE *ArrayShape*, const int *nComp*, const std::string *proc_fname*, const int *G_size[3]*, const double *pitch[3]*, const double *G_origin[3]*, const int *division[3]*, const int *head[3]*, const int *tail[3]*, const std::string *hostname*, const CIO::E_CIO_ONOFF *TSliceOnOff*) [static]

write インスタンス double 型

引数

in	*comm*	MPI コミュニケータ
in	*DfiName*	DFI ファイル名
in	*Path*	フィールドデータのディレクトリ
in	*prefix*	ベースファイル名
in	*format*	ファイルフォーマット
in	*GCell*	出力仮想セル数
in	*DataType*	データタイプ
in	*ArrayShape*	配列形状
in	*nComp*	成分数
in	*proc_fname*	proc.dfi ファイル名
in	*G_size*	グローバルボクセルサイズ
in	*pitch*	ピッチ
in	*G_origin*	原点座標値
in	*division*	領域分割数
in	*head*	計算領域の開始位置
in	*tail*	計算領域の終了位置
in	*hostname*	ホスト名
in	*TSliceOnOff*	TimeSlice フラグ

戻り値

インスタンスされたクラスのポインタ

cio_DFI.C の 379 行で定義されています。

参照先 cio_FileInfo::ArrayShape, CIO::cioPath_DirName(), cio_FileInfo::Component, cio_FileInfo::DataType, cio_FileInfo::DirectoryPath, CIO::E_CIO_BIG, CIO::E_CIO_FMT_AVS, CIO::E_CIO_FMT_BOV, CIO::E_CIO_FMT_PLOT3D, CIO::E_CIO_FMT_SPH, CIO::E_CIO_FMT_VTK, CIO::E_CIO_LITTLE, cio_FileInfo::Endian, cio_FileInfo::FileFormat, cio_Domain::GlobalDivision, cio_Domain::GlobalOrigin, cio_Domain::GlobalRegion, cio_Domain::GlobalVoxel, cio_FileInfo::GuideCell, m_comm, m_directoryPath, m_indexDfiName, m_RankID, MPI_Comm_rank(), MPI_Comm_size(), cio_MPI::NumberOfGroup, cio_MPI::NumberOfRank, cio_FileInfo::Prefix, cio_FilePath::ProcDFIFile, cio_Process::RankList, と cio_FileInfo::TimeSliceDirFlag.

```
397 {
398
399   cio_DFI *dfi = NULL;
400
401   int RankID;
402   MPI_Comm_rank( comm, &RankID );
403
404   int nrank;
405   MPI_Comm_size( comm, &nrank );
406
407   cio_FileInfo out_F_info;
408   out_F_info.DirectoryPath    = Path;
409   out_F_info.TimeSliceDirFlag = TSliceOnOff;
410   out_F_info.Prefix           = prefix;
411   out_F_info.FileFormat       = format;
412   out_F_info.GuideCell        = GCell;
413   out_F_info.DataType         = DataType;
414   out_F_info.ArrayShape       = ArrayShape;
415   out_F_info.Component        = nComp;
416
417   int idumy = 1;
418   char* cdumy = (char*)(&idumy);
419   if( cdumy[0] == 0x01 ) out_F_info.Endian = CIO::E_CIO_LITTLE;
420   if( cdumy[0] == 0x00 ) out_F_info.Endian = CIO::E_CIO_BIG;
421
422   cio_FilePath out_F_path;
423   out_F_path.ProcDFIFile = proc_fname;
424
425   cio_Unit out_unit;
426
427   cio_MPI out_mpi;
428   out_mpi.NumberOfRank = nrank;
429   out_mpi.NumberOfGroup = 1;
430
431   cio_Domain out_domain;
```

```
432   cio_Process out_Process;
433   cio_Rank out_Rank;
434
435   for(int i=0; i<nrank; i++ ) {
436      out_Process.RankList.push_back(out_Rank);
437   }
438
439   out_Process.RankList[RankID].RankID=RankID;
440   out_Process.RankList[RankID].HostName=hostname;
441   for(int i=0; i<3; i++) {
442     out_Process.RankList[RankID].HeadIndex[i]=head[i];
443     out_Process.RankList[RankID].TailIndex[i]=tail[i];
444     out_Process.RankList[RankID].VoxelSize[i]=tail[i]-head[i]+1;
445   }
446
447   for(int i=0; i<3; i++) {
448     out_domain.GlobalVoxel[i]  = G_size[i];
449     out_domain.GlobalDivision[i] = division[i];
450     out_domain.GlobalOrigin[i] = G_origin[i];
451     out_domain.GlobalRegion[i] = pitch[i]*G_size[i];
452   }
453
454   cio_TimeSlice out_TSlice;
455
456   char tmpname[512];
457   memset(tmpname,0x00,sizeof(char)*512);
458   if( gethostname(tmpname, 512) != 0 ) printf("*** error gethostname() \n");
459
460   if( out_F_info.FileFormat == CIO::E_CIO_FMT_SPH ) {
461     dfi = new cio_DFI_SPH(out_F_info, out_F_path, out_unit, out_domain, out_mpi,
462                           out_TSlice, out_Process);
463   } else if( out_F_info.FileFormat == CIO::E_CIO_FMT_BOV ) {
464     dfi = new cio_DFI_BOV(out_F_info, out_F_path, out_unit, out_domain, out_mpi,
465                           out_TSlice, out_Process);
466 //FCONV 20131122.s
467   } else if( out_F_info.FileFormat == CIO::E_CIO_FMT_AVS ) {
468     dfi = new cio_DFI_AVS(out_F_info, out_F_path, out_unit, out_domain, out_mpi,
469                           out_TSlice, out_Process);
470   } else if( out_F_info.FileFormat == CIO::E_CIO_FMT_PLOT3D ) {
471     dfi = new cio_DFI_PLOT3D(out_F_info, out_F_path, out_unit, out_domain, out_mpi,
472                           out_TSlice, out_Process);
473   } else if( out_F_info.FileFormat == CIO::E_CIO_FMT_VTK ) {
474     dfi = new cio_DFI_VTK(out_F_info, out_F_path, out_unit, out_domain, out_mpi,
475                           out_TSlice, out_Process);
476 //FCONV 20131122.e
477   } else return NULL;
478
479
480   dfi->m_indexDfiName = DfiName;
481   dfi->m_directoryPath = CIO::cioPath_DirName(DfiName);
482   dfi->m_comm = comm;
483   dfi->m_RankID = RankID;
484
485   return dfi;
486
487 }
```

6.3.3.81 CIO::E_CIO_ERRORCODE cio_DFI::WriteProcDfiFile (const MPI_Comm *comm*, bool *out_host* = `false`)

proc DFI ファイル出力コントロール (float)

引数

in	*comm*	MPI コミュニケータ
in	*out_host*	ホスト名出力フラグ

戻り値

true:出力成功 false:出力失敗 proc DFI ファイル出力コントロール

引数

in	*comm*	MPI コミュニケータ
in	*out_host*	ホスト名出力フラグ

戻り値

true:出力成功 false:出力失敗

cio_DFI_Write.C の 103 行で定義されています。

参照先 cio_Create_dfiProcessInfo(), CIO::cioPath_DirName(), CIO::cioPath_FileName(), DFI_Domain, DFI_Fpath, DFI_Process, CIO::E_CIO_ERROR_WRITE_DOMAIN, CIO::E_CIO_ERROR_WRITE_MPI, CIO::E_CIO_ERROR_WRITE_PROCESS, CIO::E_CIO_ERROR_WRITE_PROCFILE_OPENERROR, CIO::E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY, CIO::E_CIO_SUCCESS, cio_Domain::GlobalDivision, cio_Domain::GlobalOrigin, cio_Domain::GlobalRegion, cio_Domain::GlobalVoxel, m_indexDfiName, MPI_CHAR, MPI_Comm_rank(), MPI_Comm_size(), MPI_COMM_WORLD, MPI_Gather(), cio_MPI::NumberOfGroup, cio_MPI::NumberOfRank, cio_FilePath::ProcDFIFile, cio_Process::RankList, cio_MPI::Write(), cio_Domain::Write(), と cio_Process::Write().

```
106 {
107
108   //proc ファイル名の生成
109   std::string procFileName = CIO::cioPath_DirName(m_indexDfiName)+"/"+
      CIO::cioPath_FileName(DFI_Fpath.ProcDFIFile,".dfi");
110
111   if( procFileName.empty() ) return CIO::E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY;
112
113   int RankID;
114   MPI_Comm_rank( comm, &RankID );
115
116   int nrank;
117   MPI_Comm_size( comm, &nrank );
118
119   cio_MPI out_mpi;
120   out_mpi.NumberOfRank = nrank;
121   out_mpi.NumberOfGroup = 1;
122
123   cio_Domain out_domain;
124   cio_Process out_Process;
125
126   //出力する Process 情報の生成
127   cio_Create_dfiProcessInfo(comm, out_Process);
128
129   //orign の設定
130   /*
131   if( org!=NULL ) {
132     for(int i=0; i<3; i++) {
133       out_domain.GlobalOrigin[i] = org[i];
134     }
135   } else {
136   */
137     for(int i=0; i<3; i++) {
138       out_domain.GlobalOrigin[i] = DFI_Domain.GlobalOrigin[i];
139     }
140   //}
141
142   //Domain の設定
143   for(int i=0; i<3; i++) {
144     out_domain.GlobalVoxel[i]    = DFI_Domain.GlobalVoxel[i];
145     out_domain.GlobalDivision[i] = DFI_Domain.GlobalDivision[i];
146     out_domain.GlobalRegion[i]   = DFI_Domain.GlobalRegion[i];
147   }
148
149   //ホスト名出力指示ありの時、各ランクのホスト名を集める
150   if( out_host ) {
151     const int LEN=256;
152     char *recbuf = new char[out_Process.RankList.size()*LEN];
153     char  sedbuf[LEN];
154     //sprintf(sedbuf,"%s",hostname.c_str());
155     sprintf(sedbuf,"%s",DFI_Process.RankList[RankID].HostName.c_str());
156     MPI_Gather(sedbuf,LEN,MPI_CHAR,recbuf,LEN,MPI_CHAR,0,MPI_COMM_WORLD);
157
158     for( int i=0; i<out_Process.RankList.size(); i++ ) {
159      char* hn =&(recbuf[i*LEN]);
160      out_Process.RankList[i].HostName=(std::string(hn));
161     }
162
163     if( recbuf ) delete [] recbuf;
164   }
165
166   //proc.df の出力
```

```
167   if( RankID == 0 ) {
168
169     FILE* fp = NULL;
170     if( !(fp = fopen(procFileName.c_str(), "w")) )
171     {
172       fprintf(stderr, "Can't open file.(%s)\n", procFileName.c_str());
173       return CIO::E_CIO_ERROR_WRITE_PROCFILE_OPENERROR;
174     }
175
176     //Domain {} の出力
177     if( out_domain.Write(fp, 0) != CIO::E_CIO_SUCCESS )
178     {
179       if (fp) fclose(fp);
180       return CIO::E_CIO_ERROR_WRITE_DOMAIN;
181     }
182
183     //MPI {} の出力
184     if( out_mpi.Write(fp, 0) != CIO::E_CIO_SUCCESS )
185     {
186       fclose(fp);
187       return CIO::E_CIO_ERROR_WRITE_MPI;
188     }
189
190     //Process {} の出力
191     if( out_Process.Write(fp, 0) != CIO::E_CIO_SUCCESS )
192     {
193       fclose(fp);
194       return CIO::E_CIO_ERROR_WRITE_PROCESS;
195     }
196
197     fclose(fp);
198   }
199
200   return CIO::E_CIO_SUCCESS;
201
202 }
```

6.3.4 変数

6.3.4.1 cio_Domain cio_DFI::DFI_Domain [protected]

Domain class.

cio_DFI.h の 60 行で定義されています。

参照元 cio_DFI_AVS::cio_DFI_AVS(), cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_PLOT3D::cio_DFI_PLOT3D(), cio_DFI_SPH::cio_DFI_SPH(), cio_DFI_VTK::cio_DFI_VTK(), CreateReadStartEnd(), GetcioDomain(), GetDFIGlobalDivision(), GetDFIGlobalVoxel(), ReadData(), ReadInit(), SetcioDomain(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_ascii_header(), cio_DFI_PLOT3D::write_GridData(), cio_DFI_VTK::write_HeaderRecord(), cio_DFI_SPH::write_HeaderRecord(), と WriteProcDfiFile().

6.3.4.2 cio_FileInfo cio_DFI::DFI_Finfo [protected]

FileInfo class.

cio_DFI.h の 57 行で定義されています。

参照元 cio_DFI_AVS::cio_DFI_AVS(), cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_PLOT3D::cio_DFI_PLOT3D(), cio_DFI_SPH::cio_DFI_SPH(), cio_DFI_VTK::cio_DFI_VTK(), Generate_Directory_Path(), Generate_FieldFileName(), GetArrayShape(), GetArrayShapeString(), GetcioFileInfo(), getComponentVariable(), GetDataType(), GetDataTypeString(), GetNumComponent(), cio_DFI_SPH::read_averaged(), cio_DFI_SPH::read_HeaderRecord(), ReadData(), ReadFieldData(), setComponentVariable(), SetTimeSliceFlag(), cio_DFI_BOV::write_ascii_header(), cio_DFI_SPH::write_averaged(), cio_DFI_AVS::write_avs_cord(), cio_DFI_AVS::write_avs_header(), cio_DFI_BOV::write_DataRecord(), cio_DFI_AVS::write_DataRecord(), cio_DFI_SPH::write_DataRecord(), cio_DFI_PLOT3D::write_GridData(), cio_DFI_VTK::write_HeaderRecord(), cio_DFI_SPH::write_HeaderRecord(), WriteData(), WriteFieldData(), と WriteIndexDfiFile().

6.3.4.3 cio_FilePath cio_DFI::DFI_Fpath [protected]

FilePath class.

cio_DFI.h の 58 行で定義されています。

参照元 cio_DFI_AVS::cio_DFI_AVS(), cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_PLOT3D::cio_DFI_PLOT3D(), cio_DFI_SPH::cio_DFI_SPH(), cio_DFI_VTK::cio_DFI_VTK(), GetcioFilePath(), SetcioFilePath(), WriteIndexDfiFile(), と WriteProcDfiFile().

6.3.4.4 cio_MPI cio_DFI::DFI_MPI [protected]

MPI class.

cio_DFI.h の 61 行で定義されています。

参照元 cio_DFI_AVS::cio_DFI_AVS(), cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_PLOT3D::cio_DFI_PLOT3D(), cio_DFI_SPH::cio_DFI_SPH(), cio_DFI_VTK::cio_DFI_VTK(), GetcioMPI(), SetcioMPI(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_avs_cord(), cio_DFI_AVS::write_avs_header(), cio_DFI_PLOT3D::write_GridData(), と WriteData().

6.3.4.5 cio_Process cio_DFI::DFI_Process [protected]

Process class.

cio_DFI.h の 63 行で定義されています。

参照元 CheckReadRank(), cio_Create_dfiProcessInfo(), cio_DFI_AVS::cio_DFI_AVS(), cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_PLOT3D::cio_DFI_PLOT3D(), cio_DFI_SPH::cio_DFI_SPH(), cio_DFI_VTK::cio_DFI_VTK(), GetcioProcess(), ReadData(), SetcioProcess(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_ascii_header(), cio_DFI_AVS::write_avs_header(), cio_DFI_BOV::write_DataRecord(), cio_DFI_SPH::write_DataRecord(), cio_DFI_PLOT3D::write_GridData(), cio_DFI_VTK::write_HeaderRecord(), cio_DFI_SPH::write_HeaderRecord(), WriteData(), と WriteProcDfiFile().

6.3.4.6 cio_TimeSlice cio_DFI::DFI_TimeSlice [protected]

TimeSlice class.

cio_DFI.h の 62 行で定義されています。

参照元 cio_DFI_AVS::cio_DFI_AVS(), cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_PLOT3D::cio_DFI_PLOT3D(), cio_DFI_SPH::cio_DFI_SPH(), cio_DFI_VTK::cio_DFI_VTK(), GetcioTimeSlice(), getMinMax(), getVectorMinMax(), cio_DFI_BOV::read_averaged(), cio_DFI_BOV::read_HeaderRecord(), SetcioTimeSlice(), cio_DFI_AVS::write_avs_header(), WriteData(), と WriteIndexDfiFile().

6.3.4.7 cio_Unit cio_DFI::DFI_Unit [protected]

Unit class.

cio_DFI.h の 59 行で定義されています。

参照元 AddUnit(), cio_DFI_AVS::cio_DFI_AVS(), cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_PLOT3D::cio_DFI_PLOT3D(), cio_DFI_SPH::cio_DFI_SPH(), cio_DFI_VTK::cio_DFI_VTK(), GetcioUnit(), GetUnit(), GetUnitElem(), SetcioUnit(), と WriteIndexDfiFile().

6.3.4.8 bool cio_DFI::m_bgrid_interp_flag [protected]

節点への補間フラグ

cio_DFI.h の 68 行で定義されています。

参照元 cio_DFI_AVS::cio_DFI_AVS(), cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_PLOT3D::cio_DFI_PLOT3D(), cio_DFI_SPH::cio_DFI_SPH(), cio_DFI_VTK::cio_DFI_VTK(), と WriteFieldData().

6.3.4.9 MPI_Comm cio_DFI::m_comm [protected]

MPI コミュニケータ

cio_DFI.h の 50 行で定義されています。

参照元 ReadInit(), と WriteInit().

6.3.4.10 std::string cio_DFI::m_directoryPath [protected]

index dfi ファイルのディレクトリパス

cio_DFI.h の 51 行で定義されています。

参照元 Generate_Directory_Path(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_avs_cord(), cio_DFI_AVS::write_avs_header(), cio_DFI_PLOT3D::write_GridData(), WriteData(), と WriteInit().

6.3.4.11 std::string cio_DFI::m_indexDfiName [protected]

index dfi ファイル名

cio_DFI.h の 52 行で定義されています。

参照元 Generate_Directory_Path(), get_dfi_fname(), ReadData(), ReadInit(), WriteData(), WriteInit(), と WriteProcDfiFile().

6.3.4.12 CIO::E_CIO_OUTPUT_FNAME cio_DFI::m_output_fname [protected]

出力ファイル命名規約 (step_rank,rank_step)

cio_DFI.h の 70 行で定義されています。

参照元 cio_DFI(), set_output_fname(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_avs_cord(), cio_DFI_AVS::write_avs_header(), cio_DFI_PLOT3D::write_GridData(), と WriteData().

6.3.4.13 CIO::E_CIO_OUTPUT_TYPE cio_DFI::m_output_type [protected]

出力形式 (ascii,binary,FortarnBinary)

cio_DFI.h の 69 行で定義されています。

参照元 cio_DFI(), set_output_type(), cio_DFI_VTK::write_DataRecord(), cio_DFI_PLOT3D::write_DataRecord(), cio_DFI_PLOT3D::write_Func(), cio_DFI_VTK::write_HeaderRecord(), と cio_DFI_PLOT3D::write_XYZ().

6.3.4.14 int cio_DFI::m_RankID [protected]

ランク番号

cio_DFI.h の 55 行で定義されています。

参照元 cio_DFI(), ReadData(), ReadInit(), set_RankID(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_ascii_header(), cio_DFI_AVS::write_avs_cord(), cio_DFI_AVS::write_avs_header(), cio_DFI_PLOT3D::write_GridData(), WriteData(), WriteFieldData(), と WriteInit().

6.3.4.15 CIO::E_CIO_READTYPE cio_DFI::m_read_type [protected]

読込みタイプ

cio_DFI.h の 53 行で定義されています。

参照元 cio_DFI(), と ReadInit().

6.3.4.16 vector<int> cio_DFI::m_readRankList [protected]

読込みランクリスト

cio_DFI.h の 65 行で定義されています。

参照元 ReadData().

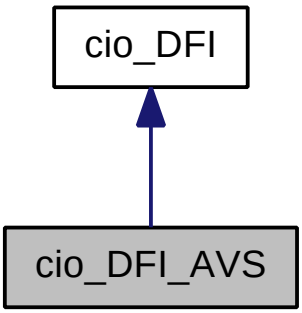
このクラスの説明は次のファイルから生成されました:

- cio_DFI.h
- cio_DFI.C
- cio_DFI_Read.C
- cio_DFI_Write.C
- cio_DFI_inline.h

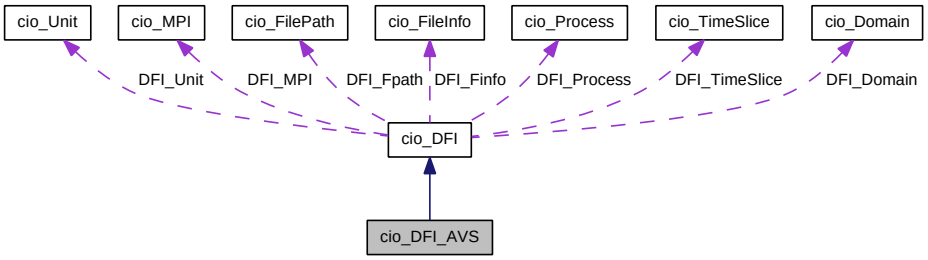
6.4 クラス cio_DFI_AVS

`#include <cio_DFI_AVS.h>`

cio_DFI_AVS に対する継承グラフ



cio_DFI_AVS のコラボレーション図



Public メソッド

- cio_DFI_AVS ()
- cio_DFI_AVS (const cio_FileInfo F_Info, const cio_FilePath F_Path, const cio_Unit unit, const cio_Domain domain, const cio_MPI mpi, const cio_TimeSlice TSlice, const cio_Process process)

  コンストラクタ
- ~cio_DFI_AVS ()

Protected メソッド

- CIO::E_CIO_ERRORCODE read_HeaderRecord (FILE *fp, bool matchEndian, unsigned step, const int head[3], const int tail[3], int gc, int voxsize[3], double &time)

  sph **ファイルのヘッダーレコード読込み**
- CIO::E_CIO_ERRORCODE read_Datarecord (FILE *fp, bool matchEndian, cio_Array *buf, int head[3], int nz, cio_Array *&src)

  フィールドデータファイルのデータレコード読込み
- CIO::E_CIO_ERRORCODE read_averaged (FILE *fp, bool matchEndian, unsigned step, unsigned &avr_step, double &avr_time)

  sph **ファイルの***Average* **データレコードの読込み**
- CIO::E_CIO_ERRORCODE write_HeaderRecord (FILE *fp, const unsigned step, const double time, const int RankID)

  SPH **ヘッダファイルの出力**
- CIO::E_CIO_ERRORCODE write_DataRecord (FILE *fp, cio_Array *val, const int gc, const int RankID)

  SPH **データレコードの出力**
- CIO::E_CIO_ERRORCODE write_averaged (FILE *fp, const unsigned step_avr, const double time_avr)

  Average **レコードの出力**
- bool write_ascii_header (const unsigned step, const double time)

  avs **の座標値データ、ヘッダーの出力コントロール**
- bool write_avs_cord (double min_ext[3], double max_ext[3])

  座標値データファイル出力
- bool write_avs_header ()

  ヘッダーデータファイルの出力

Additional Inherited Members

6.4.1 説明

cio_DFI_AVS.h の 20 行で定義されています。

6.4.2 コンストラクタとデストラクタ

6.4.2.1 cio_DFI_AVS::cio_DFI_AVS ( )

コンストラクタ

cio_DFI_AVS.C の 20 行で定義されています。

```
21 {
22
23 }
```

6.4.2.2 cio_DFI_AVS::cio_DFI_AVS (const cio_FileInfo *F_Info,* const cio_FilePath *F_Path,* const cio_Unit *unit,* const cio_Domain *domain,* const cio_MPI *mpi,* const cio_TimeSlice *TSlice,* const cio_Process *process*) [inline]

コンストラクタ

引数

in	*F_Info*	FileInfo
in	*F_Path*	FilePath
in	*unit*	Unit
in	*domain*	Domain
in	*mpi*	MPI
in	*TSlice*	TimeSlice
in	*process*	Process

cio_DFI_AVS.h の 39 行で定義されています。

参照先 cio_DFI::DFI_Domain, cio_DFI::DFI_Finfo, cio_DFI::DFI_Fpath, cio_DFI::DFI_MPI, cio_DFI::DFI_Process, cio_DFI::DFI_TimeSlice, cio_DFI::DFI_Unit, と cio_DFI::m_bgrid_interp_flag.

```
46   {
47     DFI_Finfo      = F_Info;
48     DFI_Fpath      = F_Path;
49     DFI_Unit       = unit;
50     DFI_Domain     = domain;
51     DFI_MPI        = mpi;
52     DFI_TimeSlice  = TSlice;
53     DFI_Process    = process;
54     m_bgrid_interp_flag = true;
55   };
```

6.4.2.3 cio_DFI_AVS::~cio_DFI_AVS ( )

デストラクタ

cio_DFI_AVS.C の 28 行で定義されています。

```
29 {
30
31 }
```

6.4.3 関数

6.4.3.1 CIO::E_CIO_ERRORCODE cio_DFI_AVS::read_averaged (FILE * *fp*, bool *matchEndian*, unsigned *step*, unsigned & *avr_step*, double & *avr_time*) [inline], [protected], [virtual]

sph ファイルのAverage データレコードの読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*step*	読込み step 番号
out	*avr_step*	平均ステップ
out	*avr_time*	平均タイム

戻り値

error code

cio_DFIを実装しています。

cio_DFI_AVS.h の 116 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
121   { return CIO::E_CIO_SUCCESS; };
```

6.4.3.2 CIO::E_CIO_ERRORCODE cio_DFI_AVS::read_Datarecord (FILE * *fp*, bool *matchEndian*, cio_Array * *buf*, int *head[3]*, int *nz*, cio_Array *& *src*) [inline], [protected], [virtual]

フィールドデータファイルのデータレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*buf*	読込み用バッファ
in	*head*	読込みバッファHeadIndex
in	*nz*	z 方向のボクセルサイズ（実セル＋ガイドセル＊２）
out	*src*	読み込んだデータを格納した配列のポインタ

戻り値

error code

cio_DFIを実装しています。

cio_DFI_AVS.h の 98 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
104   { return CIO::E_CIO_SUCCESS; };
```

6.4.3.3 CIO::E_CIO_ERRORCODE cio_DFI_AVS::read_HeaderRecord (FILE * *fp*, bool *matchEndian*, unsigned *step*, const int *head[3]*, const int *tail[3]*, int *gc*, int *voxsize[3]*, double & *time*) [inline], [protected], [virtual]

sph ファイルのヘッダーレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	エンディアンチェックフラグ true:合致
in	*step*	ステップ番号
in	*head*	dfi のHeadIndex
in	*tail*	dfi のTailIndex
in	*gc*	dfi のガイドセル数
out	*voxsize*	voxsize
out	*time*	時刻

戻り値

error code

cio_DFIを実装しています。

cio_DFI_AVS.h の 77 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
85   { return CIO::E_CIO_SUCCESS; };
```

6.4.3.4 bool cio_DFI_AVS::write_ascii_header (const unsigned *step*, const double *time*) [protected], [virtual]

avs の座標値データ、ヘッダーの出力コントロール

引数

in	*step*	step 番号
in	*time*	time

cio_DFIを再定義しています。

cio_DFI_AVS.C の 68 行で定義されています。

参照先 cio_DFI::DFI_Domain, cio_DFI::DFI_Process, cio_Domain::GlobalOrigin, cio_Domain::GlobalRegion, cio_Domain::GlobalVoxel, cio_DFI::m_RankID, cio_Process::RankList, write_avs_cord(), と write_avs_header().

```
70 {
71
72   int ndim,nspace;
73   int dims[3];
74   double min_ext[3],max_ext[3];
75   double pit[3];
76
77   //ピッチを計算
78   for(int i=0; i<3; i++) {
79     pit[i]=(DFI_Domain.GlobalRegion[i]/DFI_Domain.GlobalVoxel[i]);
80   }
81
82   //座標値の最小値、最大値をセット
83   for(int i=0; i<3; i++) {
84     min_ext[i]=DFI_Domain.GlobalOrigin[i]-pit[i]*0.5;
85     max_ext[i]=min_ext[i]+((double)DFI_Process.RankList[m_RankID].VoxelSize[i])*pit[i];
86   }
87
88   //座標値データファイルの出力
89   if( !write_avs_cord(min_ext,max_ext) ) return false;
90
91   //ヘッダーデータファイルの出力
92   if( !write_avs_header() ) return false;
93
94   return true;
95
96 }
```

6.4.3.5 CIO::E_CIO_ERRORCODE cio_DFI_AVS::write_averaged (FILE * *fp*, const unsigned *step_avr*, const double *time_avr*) [inline], [protected], [virtual]

Average レコードの出力

引数

in	*fp*	ファイルポインタ
in	*step_avr*	平均ステップ番号
in	*time_avr*	平均時刻

戻り値

error code

cio_DFIを実装しています。

cio_DFI_AVS.h の 159 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
162   { return CIO::E_CIO_SUCCESS; };
```

6.4.3.6 bool cio_DFI_AVS::write_avs_cord (double *min_ext[3]*, double *max_ext[3]*) [protected]

座標値データファイル出力

引数

in	*min_ext*	計算領域の最小値
in	*max_ext*	計算領域の最大値

cio_DFI_AVS.C の 100 行で定義されています。

参照先 CIO::cioPath_isAbsolute(), cio_DFI::DFI_Finfo, cio_DFI::DFI_MPI, cio_FileInfo::DirectoryPath, cio_DFI::Generate_FileName(), cio_DFI::m_directoryPath, cio_DFI::m_output_fname, cio_DFI::m_RankID, cio_MPI::NumberOfRank, と cio_FileInfo::TimeSliceDirFlag.

参照元 write_ascii_header().

```
102 {
103
104   FILE* fp=NULL;
105
106   //ファイル名の作成
107   bool mio = false;
108   if( DFI_MPI.NumberOfRank > 1 ) mio = true;
109
110   std::string fname,tmp;
111   tmp = Generate_FileName("cord",m_RankID,-1,"cod",m_output_fname,mio,
112                           DFI_Finfo.TimeSliceDirFlag);
113   if( CIO::cioPath_isAbsolute(DFI_Finfo.DirectoryPath) ){
114     fname = DFI_Finfo.DirectoryPath + "/" + tmp;
115   } else {
116     fname = m_directoryPath + "/" + DFI_Finfo.DirectoryPath +"/"+ tmp;
117   }
118
119   printf("cord file name : %s\n",fname.c_str());
120
121   //座標値データファイルオープン
122   if( (fp = fopen(fname.c_str(),"w"))  == NULL ) {
123     printf("\tCan't open file.(%s)\n",fname.c_str());
124     return false;
125   }
126
127   //座標値データ（min,max）の出力
128   fprintf(fp,"#### X #####\n");
129   fprintf(fp,"%.6f\n",min_ext[0]);
130   fprintf(fp,"%.6f\n",max_ext[0]);
131   fprintf(fp,"#### Y #####\n");
132   fprintf(fp,"%.6f\n",min_ext[1]);
133   fprintf(fp,"%.6f\n",max_ext[1]);
134   fprintf(fp,"#### Z #####\n");
135   fprintf(fp,"%.6f\n",min_ext[2]);
136   fprintf(fp,"%.6f\n",max_ext[2]);
137
138   //座標値データファイルクローズ
139   fclose(fp);
140
141   return true;
142
143 }
```

6.4.3.7 bool cio_DFI_AVS::write_avs_header ( ) [protected]

ヘッダーデータファイルの出力

cio_DFI_AVS.C の 147 行で定義されています。

参照先 CIO::cioPath_isAbsolute(), cio_FileInfo::Component, cio_DFI::DFI_Finfo, cio_DFI::DFI_MPI, cio_DFI::DFI_Process, cio_DFI::DFI_TimeSlice, cio_FileInfo::DirectoryPath, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, cio_DFI::Generate_FileName(), cio_DFI::getComponentVariable(), cio_DFI::GetDataType(), cio_DFI::GetDataTypeString(), cio_DFI::m_directoryPath, cio_DFI::m_output_fname, cio_DFI::m_RankID, cio_MPI::NumberOfRank, cio_FileInfo::Prefix, cio_Process::RankList, cio_TimeSlice::SliceList, と cio_FileInfo::TimeSliceDirFlag.

参照元 write_ascii_header().

```
148 {
149   FILE* fp=NULL;
150   std::string dType;
151   std::string out_fname;
152
153   bool mio=false;
```

```
154
155   //データタイプのセット
156   if( GetDataType() == CIO::E_CIO_FLOAT32 ) {
157     dType = "float";
158   } else if( GetDataType() == CIO::E_CIO_FLOAT64 ) {
159     dType = "double";
160   } else {
161     dType = GetDataTypeString();
162     printf("\tillergal data type.(%s)\n",dType.c_str());
163     return false;
164   }
165
166   //ファイル名生成
167
168   if( DFI_MPI.NumberOfRank > 1 ) mio = true;
169   std::string fname,tmp;
170   tmp = Generate_FileName(DFI_Finfo.Prefix,m_RankID,-1,"fld",m_output_fname,mio,
171                           DFI_Finfo.TimeSliceDirFlag);
172   if( CIO::cioPath_isAbsolute(DFI_Finfo.DirectoryPath) ){
173     fname = DFI_Finfo.DirectoryPath +"/"+ tmp;
174   } else {
175     fname = m_directoryPath + "/" + DFI_Finfo.DirectoryPath +"/"+ tmp;
176   }
177
178   printf("fld file name : %s\n",fname.c_str());
179
180   //出力ヘッダーファイルオープン
181   if( (fp = fopen(fname.c_str(),"w"))  == NULL ) {
182     printf("\tCan't open file.(%s)\n",fname.c_str());
183     return false;
184   }
185
186   int ndim = 3;
187   int nspace = 3;
188   //dims = DFI_Process.RankList[m_RankID].VoxelSize[0]
189
190   //先頭レコードの出力
191   fprintf(fp,"# AVS field file\n");
192
193   //計算空間の次元数を出力
194   fprintf(fp,"ndim=%d\n",ndim);
195
196   //計算空間サイズを出力
197   fprintf(fp,"dim1=%d\n",DFI_Process.RankList[m_RankID].VoxelSize[0]+1);
198   fprintf(fp,"dim2=%d\n",DFI_Process.RankList[m_RankID].VoxelSize[1]+1);
199   fprintf(fp,"dim3=%d\n",DFI_Process.RankList[m_RankID].VoxelSize[2]+1);
200
201   //物理空間の次元数を出力
202   fprintf(fp,"nspace=%d\n",nspace);
203
204   //成分数の出力
205   fprintf(fp,"veclen=%d\n",DFI_Finfo.Component);
206
207   //データのタイプ出力
208   fprintf(fp,"data=%s\n",dType.c_str());
209
210   //座標定義情報の出力
211   fprintf(fp,"field=uniform\n");
212
213   //label の出力
214   for(int i=0; i<DFI_Finfo.Component; i++) {
215     std::string label=getComponentVariable(i);
216     if( label == "" ) continue;
217     fprintf(fp,"label=%s\n",label.c_str());
218   }
219
220   //step 毎の出力
221   if( DFI_TimeSlice.SliceList.size()>1 ) {
222     fprintf(fp,"nstep=%d\n",(int)DFI_TimeSlice.SliceList.size());
223   }
224   for(int i=0; i<DFI_TimeSlice.SliceList.size(); i++) {
225     fprintf(fp,"time value=%.6f\n",DFI_TimeSlice.SliceList[i].time);
226
227     //field data file name 出力
228     for(int j=1; j<=DFI_Finfo.Component; j++) {
229       int skip;
230       if( dType == "float" ) {
231         skip=96+(j-1)*4;
232       } else {
233         skip=140+(j-1)*8;
234       }
235       out_fname=Generate_FileName(DFI_Finfo.Prefix,
236                                   m_RankID,
237                                   DFI_TimeSlice.SliceList[i].step,
238                                   "sph",
239                                   m_output_fname,
240                                   mio,
```

```
241                                 DFI_Finfo.TimeSliceDirFlag);
242       //std::string xxx = CIO::cioPath_FileName(out_fname,"sph");
243       fprintf(fp,"variable %d file=%s filetype=binary skip=%d stride=%d\n",
244               j,out_fname.c_str(),skip,DFI_Finfo.Component);
245     }
246
247     //coord data file name 出力
248     tmp = Generate_FileName("cord",m_RankID,-1,"cod",m_output_fname,mio,
249                             DFI_Finfo.TimeSliceDirFlag);
250     fprintf(fp,"coord 1 file=%s filetype=ascii skip=1\n",tmp.c_str());
251     fprintf(fp,"coord 2 file=%s filetype=ascii skip=4\n",tmp.c_str());
252     fprintf(fp,"coord 3 file=%s filetype=ascii skip=7\n",tmp.c_str());
253     fprintf(fp,"EOT\n");
254
255   }
256
257   //出力ヘッダーファイルクローズ
258   fclose(fp);
259
260   //if( tmp ) delete tmp;
261
262   return true;
263 }
```

6.4.3.8 CIO::E_CIO_ERRORCODE cio_DFI_AVS::write_DataRecord (FILE * *fp*, cio_Array * *val*, const int *gc*, const int *RankID*) [protected], [virtual]

SPH データレコードの出力

引数

in	*fp*	ファイルポインタ
in	*val*	データポインタ
in	*gc*	ガイドセル
in	*RankID*	ランク番号

戻り値

error code

cio_DFIを実装しています。

cio_DFI_AVS.C の 48 行で定義されています。

参照先 cio_FileInfo::Component, cio_FileInfo::DataType, cio_DFI::DFI_Finfo, CIO::E_CIO_ERROR_WRITE_FIELD_DATA_RECORD, CIO::E_CIO_SUCCESS, cio_DFI::get_cio_Datasize(), cio_Array::getArraySizeInt(), と cio_Array::writeBinary().

```
52 {
53
54   CIO::E_CIO_DTYPE Dtype = (CIO::E_CIO_DTYPE)DFI_Finfo.DataType;
55   int Real_size = get_cio_Datasize(Dtype);
56
57   const int *size = val->getArraySizeInt();
58   size_t dLen = (size_t)(size[0] * size[1] * size[2]);
59   if( DFI_Finfo.Component > 1 ) dLen *= 3;
60
61   if( val->writeBinary(fp) != dLen ) return
      CIO::E_CIO_ERROR_WRITE_FIELD_DATA_RECORD;
62
63   return CIO::E_CIO_SUCCESS;
64 }
```

6.4.3.9 CIO::E_CIO_ERRORCODE cio_DFI_AVS::write_HeaderRecord (FILE * *fp*, const unsigned *step*, const double *time*, const int *RankID*) [protected], [virtual]

SPH ヘッダファイルの出力

引数

in	*fp*	ファイルポインタ
in	*step*	ステップ番号
in	*time*	時刻
in	*RankID*	ランク番号

戻り値

error code

cio_DFIを実装しています。

cio_DFI_AVS.C の 36 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
40 {
41
42   return CIO::E_CIO_SUCCESS;
43 }
```

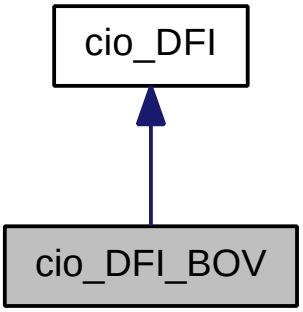
このクラスの説明は次のファイルから生成されました:

- cio_DFI_AVS.h
- cio_DFI_AVS.C

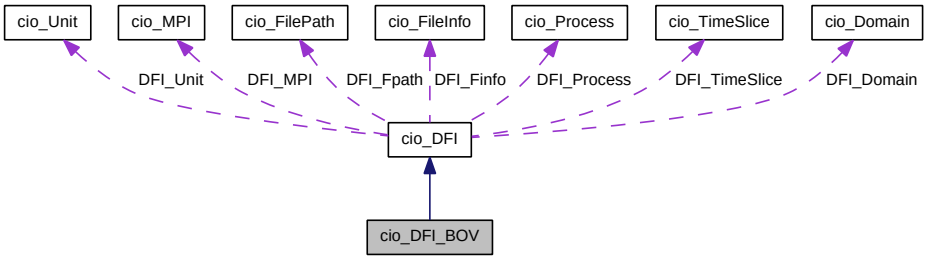
6.5 クラス cio_DFI_BOV

`#include <cio_DFI_BOV.h>`

cio_DFI_BOV に対する継承グラフ



cio_DFI_BOV のコラボレーション図



Public メソッド

- cio_DFI_BOV ()
- cio_DFI_BOV (const cio_FileInfo F_Info, const cio_FilePath F_Path, const cio_Unit unit, const cio_Domain domain, const cio_MPI mpi, const cio_TimeSlice TSlice, const cio_Process process)

  コンストラクタ
- ~cio_DFI_BOV ()

Protected メソッド

- CIO::E_CIO_ERRORCODE read_HeaderRecord (FILE *fp, bool matchEndian, unsigned step, const int head[3], const int tail[3], int gc, int voxsize[3], double &time)

  bov **ファイルのヘッダーレコード読込み**
- CIO::E_CIO_ERRORCODE read_Datarecord (FILE *fp, bool matchEndian, cio_Array *buf, int head[3], int nz, cio_Array *&src)

  フィールドデータファイルのデータレコード読込み
- CIO::E_CIO_ERRORCODE read_averaged (FILE *fp, bool matchEndian, unsigned step, unsigned &avr_step, double &avr_time)

  bov **ファイルの***Average* **データレコードの読込み**
- CIO::E_CIO_ERRORCODE write_HeaderRecord (FILE *fp, const unsigned step, const double time, const int RankID)

  bov **ヘッダファイルの出力**
- CIO::E_CIO_ERRORCODE write_DataRecord (FILE *fp, cio_Array *val, const int gc, const int RankID)

  bov **データ出力**
- CIO::E_CIO_ERRORCODE write_averaged (FILE *fp, const unsigned step_avr, const double time_avr)

  Average **レコードの出力**
- bool write_ascii_header (const unsigned step, const double time)

  ヘッダーデータファイルの出力

Additional Inherited Members

6.5.1 説明

cio_DFI_BOV.h の 20 行で定義されています。

6.5.2 コンストラクタとデストラクタ

6.5.2.1 cio_DFI_BOV::cio_DFI_BOV ( )

コンストラクタ

cio_DFI_BOV.C の 20 行で定義されています。

```
21 {
22
23 }
```

6.5.2.2 cio_DFI_BOV::cio_DFI_BOV (const cio_FileInfo *F_Info,* const cio_FilePath *F_Path,* const cio_Unit *unit,* const cio_Domain *domain,* const cio_MPI *mpi,* const cio_TimeSlice *TSlice,* const cio_Process *process*) [inline]

コンストラクタ

引数

in	*F_Info*	FileInfo
in	*F_Path*	FilePath
in	*unit*	Unit
in	*domain*	Domain
in	*mpi*	MPI
in	*TSlice*	TimeSlice
in	*process*	Process

cio_DFI_BOV.h の 36 行で定義されています。

参照先 cio_DFI::DFI_Domain, cio_DFI::DFI_Finfo, cio_DFI::DFI_Fpath, cio_DFI::DFI_MPI, cio_DFI::DFI_Process, cio_DFI::DFI_TimeSlice, cio_DFI::DFI_Unit, と cio_DFI::m_bgrid_interp_flag.

```
43   {
44     DFI_Finfo      = F_Info;
45     DFI_Fpath      = F_Path;
46     DFI_Unit       = unit;
47     DFI_Domain     = domain;
48     DFI_MPI        = mpi;
49     DFI_TimeSlice  = TSlice;
50     DFI_Process    = process;
51     m_bgrid_interp_flag = false;
52   };
```

6.5.2.3 cio_DFI_BOV::~cio_DFI_BOV ( )

デストラクタ

cio_DFI_BOV.C の 28 行で定義されています。

```
29 {
30
31 }
```

6.5.3 関数

6.5.3.1 CIO::E_CIO_ERRORCODE cio_DFI_BOV::read_averaged (FILE * *fp*, bool *matchEndian*, unsigned *step*, unsigned & *avr_step*, double & *avr_time*) [protected], [virtual]

bov ファイルのAverage データレコードの読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*step*	読込み step 番号
out	*avr_step*	平均ステップ
out	*avr_time*	平均タイム

戻り値

errorcode

cio_DFIを実装しています。

cio_DFI_BOV.C の 113 行で定義されています。

参照先 cio_DFI::DFI_TimeSlice, CIO::E_CIO_SUCCESS, と cio_TimeSlice::SliceList.

```
118 {
119
```

```
120   step_avr=0;
121   time_avr=0.0;
122
123   for(int i=0; i<DFI_TimeSlice.SliceList.size(); i++) {
124      if( DFI_TimeSlice.SliceList[i].step == step ) {
125        step_avr=(int)DFI_TimeSlice.SliceList[i].AveragedStep;
126        time_avr=(double)DFI_TimeSlice.SliceList[i].AveragedTime;
127      }
128   }
129
130   return CIO::E_CIO_SUCCESS;
131 }
```

6.5.3.2 CIO::E_CIO_ERRORCODE cio_DFI_BOV::read_Datarecord (FILE * *fp*, bool *matchEndian*, cio_Array * *buf*, int *head[3]*, int *nz*, cio_Array *& *src*) [protected], [virtual]

フィールドデータファイルのデータレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*buf*	読込み用バッファ
in	*head*	読込みバッファHeadIndex
in	*nz*	z 方向のボクセルサイズ（実セル＋ガイドセル＊２）
out	*src*	読み込んだデータを格納した配列のポインタ

戻り値

error code

cio_DFIを実装しています。

cio_DFI_BOV.C の 61 行で定義されています。

参照先 cio_Array::copyArray(), cio_Array::copyArrayNcomp(), CIO::E_CIO_ERROR_READ_FIELD_DATA_RECORD, CIO::E_CIO_IJKN, CIO::E_CIO_NIJK, CIO::E_CIO_SUCCESS, cio_Array::getArrayLength(), cio_Array::getArrayShape(), cio_Array::getNcomp(), cio_Array::readBinary(), と cio_Array::setHeadIndex().

```
67 {
68
69   //１層ずつ読み込み
70   int hzB = head[2];
71
72   CIO::E_CIO_ARRAYSHAPE shape = buf->getArrayShape();
73
74   //NIJK の読込み
75   if( shape == CIO::E_CIO_NIJK ) {
76     for( int k=0; k<nz; k++ ) {
77       //head インデクスをずらす
78       head[2]=hzB+k;
79       buf->setHeadIndex(head);
80
81       //１層読み込
82       size_t ndata = buf->getArrayLength();
83       if( buf->readBinary(fp,matchEndian) != ndata ) return
     CIO::E_CIO_ERROR_READ_FIELD_DATA_RECORD;
84
85       // コピー
86       buf->copyArray(src);
87     }
88   }
89   //IJKN の読込み
90   else if( shape == CIO::E_CIO_IJKN ) {
91     for(int n=0; n<src->getNcomp(); n++) {
92     for(int k=0; k<nz; k++) {
93       //head インデックスをずらす
94       head[2]=hzB+k;
95       buf->setHeadIndex(head);
96
97       //１層読み込
98       size_t ndata = buf->getArrayLength();
99       if( buf->readBinary(fp,matchEndian) != ndata ) return
```

```
      CIO::E_CIO_ERROR_READ_FIELD_DATA_RECORD;
100
101       //コピー
102       buf->copyArrayNcomp(src,n);
103     }}
104   }
105
106   return CIO::E_CIO_SUCCESS;
107
108 }
```

6.5.3.3 CIO::E_CIO_ERRORCODE cio_DFI_BOV::read_HeaderRecord (FILE * *fp*, bool *matchEndian*, unsigned *step*, const int *head[3]*, const int *tail[3]*, int *gc*, int *voxsize[3]*, double & *time*) [protected], [virtual]

bov ファイルのヘッダーレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	エンディアンチェックフラグ true:合致
in	*step*	ステップ番号
in	*head*	dfi のHeadIndex
in	*tail*	dfi のTailIndex
in	*gc*	dfi のガイドセル数
out	*voxsize*	voxsize
out	*time*	時刻

戻り値

error code

cio_DFIを実装しています。

cio_DFI_BOV.C の 36 行で定義されています。

参照先 cio_DFI::DFI_TimeSlice, CIO::E_CIO_SUCCESS, と cio_TimeSlice::SliceList.

```
44 {
45
46   time=0.0;
47   for(int i=0; i<DFI_TimeSlice.SliceList.size(); i++) {
48     if( DFI_TimeSlice.SliceList[i].step == step ) {
49       time=(double)DFI_TimeSlice.SliceList[i].time;
50     }
51   }
52
53   for(int i=0; i<3; i++) voxsize[i]=tail[i]-head[i]+1+(2*gc);
54
55   return CIO::E_CIO_SUCCESS;
56 }
```

6.5.3.4 bool cio_DFI_BOV::write_ascii_header (const unsigned *step*, const double *time*) [protected], [virtual]

ヘッダーデータファイルの出力

引数

in	*step*	step 番号
in	*time*	time

cio_DFIを再定義しています。

cio_DFI_BOV.C の 181 行で定義されています。

参照先 cio_FileInfo::ArrayShape, CIO::cioPath_isAbsolute(), cio_FileInfo::Component, cio_DFI::DFI_Domain, cio_DFI::DFI_Finfo, cio_DFI::DFI_MPI, cio_DFI::DFI_Process, cio_FileInfo::DirectoryPath, CIO::E_CIO_FLOAT32, C-

IO::E_CIO_FLOAT64, CIO::E_CIO_IJKN, CIO::E_CIO_INT32, CIO::E_CIO_INT8, CIO::E_CIO_LITTLE, cio_FileInfo::Endian, cio_DFI::Generate_FileName(), cio_DFI::GetDataType(), cio_Domain::GlobalOrigin, cio_Domain::GlobalRegion, cio_Domain::GlobalVoxel, cio_DFI::m_directoryPath, cio_DFI::m_output_fname, cio_DFI::m_RankID, cio_MPI::NumberOfRank, cio_FileInfo::Prefix, cio_Process::RankList, と cio_FileInfo::TimeSliceDirFlag.

```
183 {
184
185   FILE* fp=NULL;
186
187   //ファイル名生成
188   bool mio=false;
189   if( DFI_MPI.NumberOfRank > 1 ) mio = true;
190
191   std::string fname,tmp;
192   tmp = Generate_FileName(DFI_Finfo.Prefix,
193                           m_RankID,
194                           step,
195                           "bov",
196                           m_output_fname,
197                           mio,
198                           DFI_Finfo.TimeSliceDirFlag);
199
200   if( CIO::cioPath_isAbsolute(DFI_Finfo.DirectoryPath) ){
201     fname = DFI_Finfo.DirectoryPath + "/" + tmp;
202   } else {
203     fname = m_directoryPath + "/" + DFI_Finfo.DirectoryPath +"/"+ tmp;
204   }
205
206   //bov ヘッダーファイルオープン
207   if( (fp = fopen(fname.c_str(),"w")) == NULL ) {
208     printf("\tCan't open file.(%s)\n",fname.c_str());
209     return false;
210   }
211
212   //TIME:
213   fprintf(fp,"Time: %e\n",time);
214
215   //DATA_FILE:
216   std::string o_fname;
217   o_fname = Generate_FileName(DFI_Finfo.Prefix,
218                               m_RankID,
219                               step,
220                               "dat",
221                               m_output_fname,
222                               mio,
223                               DFI_Finfo.TimeSliceDirFlag);
224   fprintf(fp,"DATA_FILE: %s\n",o_fname.c_str());
225
226   //DATA_SIZE:
227   fprintf(fp,"DATA_SIZE: %d %d %d\n",DFI_Process.RankList[m_RankID].VoxelSize[0],
228                                      DFI_Process.RankList[m_RankID].VoxelSize[1],
229                                      DFI_Process.RankList[m_RankID].VoxelSize[2]);
230
231   //DATA_FORMAT
232   std::string dType;
233   if(        GetDataType() == CIO::E_CIO_INT8 ) {
234     dType="BYTE";
235   } else if( GetDataType() == CIO::E_CIO_INT32 ) {
236     dType="INT";
237   } else if( GetDataType() == CIO::E_CIO_FLOAT32 ) {
238     dType="FLOAT";
239   } else if( GetDataType() == CIO::E_CIO_FLOAT64 ) {
240     dType="DOUBLE";
241   }
242   fprintf(fp,"DATA_FORMAT: %s\n",dType.c_str());
243
244   //DATA_COMPONENT
245   fprintf(fp,"DATA_COMPONENT: %d\n",DFI_Finfo.Component);
246
247   //VARIABLE:
248   fprintf(fp,"VARIABLE: %s\n",DFI_Finfo.Prefix.c_str());
249
250   //DATA_ENDIAN
251   if( DFI_Finfo.Endian == CIO::E_CIO_LITTLE ) {
252     fprintf(fp,"DATA_ENDIAN: LITTLE\n");
253   } else {
254     fprintf(fp,"DATA_ENDIAN: BIG\n");
255   }
256
257   //CENTERING
258   fprintf(fp,"CENTERING: zonal\n");
259
260   //BRICK_ORIGN
261   fprintf(fp,"BRICK_ORIGN: %e %e %e\n",DFI_Domain.GlobalOrigin[0],
262                                        DFI_Domain.GlobalOrigin[1],
```

```
263                                     DFI_Domain.GlobalOrigin[2]);
264
265   //pit を計算
266   double pit[3];
267   for(int i=0; i<3; i++) {
268     pit[i]=(DFI_Domain.GlobalRegion[i]/DFI_Domain.GlobalVoxel[i]);
269   }
270
271   //BRICK_SIZE
272   fprintf(fp,"BRICK_SIZE: %e %e %e\n",
273           DFI_Process.RankList[m_RankID].VoxelSize[0]*pit[0],
274           DFI_Process.RankList[m_RankID].VoxelSize[1]*pit[1],
275           DFI_Process.RankList[m_RankID].VoxelSize[2]*pit[2]);
276
277   //#CIO_ARRAY_SHAPE
278   if( DFI_Finfo.ArrayShape == CIO::E_CIO_IJKN ) {
279     fprintf(fp,"#CIO_ARRAY_SHAPE: IJKN\n");
280   } else {
281     fprintf(fp,"#CIO_ARRAY_SHAPE: NIJK\n");
282   }
283
284   //file close
285   fclose(fp);
286
287   return true;
288 }
```

6.5.3.5 CIO::E_CIO_ERRORCODE cio_DFI_BOV::write_averaged (FILE * *fp*, const unsigned *step_avr*, const double *time_avr*) [protected], [virtual]

Average レコードの出力

引数

in	*fp*	ファイルポインタ
in	*step_avr*	平均ステップ番号
in	*time_avr*	平均時刻

戻り値

error code

cio_DFIを実装しています。

cio_DFI_BOV.C の 171 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
174 {
175   return CIO::E_CIO_SUCCESS;
176 }
```

6.5.3.6 CIO::E_CIO_ERRORCODE cio_DFI_BOV::write_DataRecord (FILE * *fp*, cio_Array * *val*, const int *gc*, const int *RankID*) [protected], [virtual]

bov データ出力

引数

in	*fp*	ファイルポインタ
in	*val*	データポインタ
in	*gc*	仮想セル数

in	*RankID*	ランク番号

戻り値

error code

cio_DFIを実装しています。

cio_DFI_BOV.C の 147 行で定義されています。

参照先 cio_FileInfo::Component, cio_FileInfo::DataType, cio_DFI::DFI_Finfo, cio_DFI::DFI_Process, CIO::E_CIO_ERROR_WRITE_FIELD_DATA_RECORD, CIO::E_CIO_SUCCESS, cio_DFI::get_cio_Datasize(), cio_Process::RankList, と cio_Array::writeBinary().

```
151 {
152
153   CIO::E_CIO_DTYPE Dtype = (CIO::E_CIO_DTYPE)DFI_Finfo.DataType;
154   int Real_size = get_cio_Datasize(Dtype);
155
156   int size[3];
157   for(int i=0; i<3; i++ ) size[i] = (int)DFI_Process.RankList[n].VoxelSize[i]+(int)(2*gc);
158
159   size_t dLen = (size_t)(size[0] * size[1] * size[2]);
160   if( DFI_Finfo.Component > 1 ) dLen *= 3;
161
162   unsigned int dmy = dLen * Real_size;
163
164   if( val->writeBinary(fp) != dLen ) return
      CIO::E_CIO_ERROR_WRITE_FIELD_DATA_RECORD;
165   return CIO::E_CIO_SUCCESS;
166 }
```

6.5.3.7 CIO::E_CIO_ERRORCODE cio_DFI_BOV::write_HeaderRecord (FILE * *fp*, const unsigned *step*, const double *time*, const int *RankID*) [protected], [virtual]

bov ヘッダファイルの出力

引数

in	*fp*	ファイルポインタ
in	*step*	ステップ番号
in	*time*	時刻
in	*RankID*	ランク番号

戻り値

error code

cio_DFIを実装しています。

cio_DFI_BOV.C の 136 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
140 {
141   return CIO::E_CIO_SUCCESS;
142 }
```

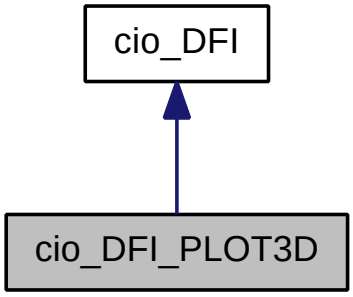
このクラスの説明は次のファイルから生成されました:

- cio_DFI_BOV.h
- cio_DFI_BOV.C

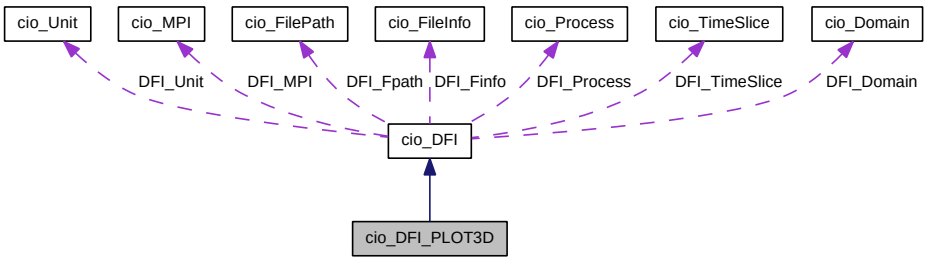
6.6 クラス cio_DFI_PLOT3D

`#include <cio_DFI_PLOT3D.h>`

cio_DFI_PLOT3D に対する継承グラフ



cio_DFI_PLOT3D のコラボレーション図



Public メソッド

- cio_DFI_PLOT3D ()
- cio_DFI_PLOT3D (const cio_FileInfo F_Info, const cio_FilePath F_Path, const cio_Unit unit, const cio_Domain domain, const cio_MPI mpi, const cio_TimeSlice TSlice, const cio_Process process)

  コンストラクタ
- ~cio_DFI_PLOT3D ()
- template<class T >
  CIO_INLINE void write_XYZ (FILE *fp, T *org, T *pit, int sz[3])
- template<class T >
  CIO_INLINE void write_Func (FILE *fp, cio_TypeArray< T > *data, const int sz[3], int ncomp)

Protected メソッド

- CIO::E_CIO_ERRORCODE read_HeaderRecord (FILE *fp, bool matchEndian, unsigned step, const int head[3], const int tail[3], int gc, int voxsize[3], double &time)

  sph **ファイルのヘッダーレコード読込み**
- CIO::E_CIO_ERRORCODE read_Datarecord (FILE *fp, bool matchEndian, cio_Array *buf, int head[3], int nz, cio_Array *&src)

  フィールドデータファイルのデータレコード読込み
- CIO::E_CIO_ERRORCODE read_averaged (FILE *fp, bool matchEndian, unsigned step, unsigned &avr_-step, double &avr_time)

  sph **ファイルの***Average* **データレコードの読込み**
- CIO::E_CIO_ERRORCODE write_HeaderRecord (FILE *fp, const unsigned step, const double time, const int RankID)

SPH **ヘッダファイルの出力**

- CIO::E_CIO_ERRORCODE write_DataRecord (FILE *fp, cio_Array *val, const int gc, const int RankID)

  SPH **データレコードの出力**
- CIO::E_CIO_ERRORCODE write_averaged (FILE *fp, const unsigned step_avr, const double time_avr)

  Average **レコードの出力**
- bool write_GridData ()

  Grid data file **出力 コントロール**
- template<class T >
  void write_XYZ (FILE *fp, T *org, T *pit, int sz[3])

  xyz **を計算して出力**
- template<class T >
  void write_Func (FILE *fp, cio_TypeArray< T > *data, const int sz[3], int ncomp)

  func data **出力**

Protected 変数

- bool m_OutputGrid

  plot3d grid file **出力指示**

Additional Inherited Members

6.6.1 説明

cio_DFI_PLOT3D.h の 20 行で定義されています。

6.6.2 コンストラクタとデストラクタ

6.6.2.1 cio_DFI_PLOT3D::cio_DFI_PLOT3D ( )

コンストラクタ

cio_DFI_PLOT3D.C の 20 行で定義されています。

```
21 {
22
23 }
```

6.6.2.2 cio_DFI_PLOT3D::cio_DFI_PLOT3D (const cio_FileInfo *F_Info*, const cio_FilePath *F_Path*, const cio_Unit *unit*, const cio_Domain *domain*, const cio_MPI *mpi*, const cio_TimeSlice *TSlice*, const cio_Process *process*) [inline]

コンストラクタ

引数

in	*F_Info*	FileInfo
in	*F_Path*	FilePath
in	*unit*	Unit
in	*domain*	Domain

in	*mpi*	MPI
in	*TSlice*	TimeSlice
in	*process*	Process

cio_DFI_PLOT3D.h の 41 行で定義されています。

参照先 cio_DFI::DFI_Domain, cio_DFI::DFI_Finfo, cio_DFI::DFI_Fpath, cio_DFI::DFI_MPI, cio_DFI::DFI_Process, cio_DFI::DFI_TimeSlice, cio_DFI::DFI_Unit, cio_DFI::m_bgrid_interp_flag, と m_OutputGrid.

```
48   {
49     DFI_Finfo      = F_Info;
50     DFI_Fpath      = F_Path;
51     DFI_Unit       = unit;
52     DFI_Domain     = domain;
53     DFI_MPI        = mpi;
54     DFI_TimeSlice  = TSlice;
55     DFI_Process    = process;
56     m_OutputGrid   = true;
57     m_bgrid_interp_flag = true;
58   };
```

6.6.2.3 cio_DFI_PLOT3D::~cio_DFI_PLOT3D ( )

デストラクタ

cio_DFI_PLOT3D.C の 28 行で定義されています。

```
29 {
30
31 }
```

6.6.3 関数

6.6.3.1 CIO::E_CIO_ERRORCODE cio_DFI_PLOT3D::read_averaged (FILE * *fp*, bool *matchEndian*, unsigned *step*, unsigned & *avr_step*, double & *avr_time*) [inline], [protected], [virtual]

sph ファイルのAverage データレコードの読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*step*	読込み step 番号
out	*avr_step*	平均ステップ
out	*avr_time*	平均タイム

戻り値

error code

cio_DFIを実装しています。

cio_DFI_PLOT3D.h の 119 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
124   { return CIO::E_CIO_SUCCESS; };
```

6.6.3.2 CIO::E_CIO_ERRORCODE cio_DFI_PLOT3D::read_Datarecord (FILE * *fp*, bool *matchEndian*, cio_Array * *buf*, int *head[3]*, int *nz*, cio_Array *& *src*) [inline], [protected], [virtual]

フィールドデータファイルのデータレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*buf*	読込み用バッファ
in	*head*	読込みバッファHeadIndex
in	*nz*	z 方向のボクセルサイズ（実セル＋ガイドセル＊２）
out	*src*	読み込んだデータを格納した配列のポインタ

戻り値

error code

cio_DFIを実装しています。

cio_DFI_PLOT3D.h の 101 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
107   { return CIO::E_CIO_SUCCESS; };
```

6.6.3.3 CIO::E_CIO_ERRORCODE cio_DFI_PLOT3D::read_HeaderRecord (FILE * *fp*, bool *matchEndian*, unsigned *step*, const int *head[3]*, const int *tail[3]*, int *gc*, int *voxsize[3]*, double & *time*) [inline], [protected], [virtual]

sph ファイルのヘッダーレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	エンディアンチェックフラグ true:合致
in	*step*	ステップ番号
in	*head*	dfi のHeadIndex
in	*tail*	dfi のTailIndex
in	*gc*	dfi のガイドセル数
out	*voxsize*	voxsize
out	*time*	時刻

戻り値

error code

cio_DFIを実装しています。

cio_DFI_PLOT3D.h の 80 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
88   { return CIO::E_CIO_SUCCESS; };
```

6.6.3.4 CIO::E_CIO_ERRORCODE cio_DFI_PLOT3D::write_averaged (FILE * *fp*, const unsigned *step_avr*, const double *time_avr*) [inline], [protected], [virtual]

Average レコードの出力

引数

in	*fp*	ファイルポインタ
in	*step_avr*	平均ステップ番号
in	*time_avr*	平均時刻

戻り値

error code

cio_DFIを実装しています。

cio_DFI_PLOT3D.h の 162 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
165   { return CIO::E_CIO_SUCCESS; };
```

6.6.3.5 CIO::E_CIO_ERRORCODE cio_DFI_PLOT3D::write_DataRecord (FILE * *fp*, cio_Array * *val*, const int *gc*, const int *RankID*) [protected], [virtual]

SPH データレコードの出力

引数

in	*fp*	ファイルポインタ
in	*val*	データポインタ
in	*gc*	ガイドセル
in	*RankID*	ランク番号

戻り値

error code

cio_DFIを実装しています。

cio_DFI_PLOT3D.C の 47 行で定義されています。

参照先 CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_OUTPUT_TYPE_ASCII, CIO::E_CIO_OUTPUT_TYPE_FBINARY, CIO::E_CIO_SUCCESS, cio_Array::getArraySizeInt(), cio_Array::getDataType(), cio_Array::getNcomp(), cio_DFI::m_output_type, m_OutputGrid, write_Func(), と write_GridData().

```
51 {
52
53   //GRID データファイル出力処理
54   if( m_OutputGrid == true ) {
55     write_GridData();
56     m_OutputGrid = false;
57   }
58
59   //フィールドデータの配列サイズ取得
60   const int *szVal = val->getArraySizeInt();
61
62   //配列成分の取得
63   int ncomp = val->getNcomp();
64
65   //printf("ID : %d prefix : %s size : %d %d %d ncomp : %d\n",n,
66   //        DFI_Finfo.Prefix.c_str(),szVal[0],szVal[1],szVal[2],
67   //        ncomp);
68
69   //ngrid,nblock 出力
70   int ngrid=1;
71   //ascii
72   if( m_output_type == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
73     fprintf(fp,"%5d\n",ngrid);
74     fprintf(fp,"%5d%5d%5d%5d\n",szVal[0],szVal[1],szVal[2],ncomp);
75   //Fortran Binary
76   } else if( m_output_type == CIO::E_CIO_OUTPUT_TYPE_FBINARY ) {
```

```
77     unsigned int dmy;
78     dmy = sizeof(int);
79     fwrite(&dmy, sizeof(int), 1, fp);
80     fwrite(&ngrid, sizeof(int), 1, fp);
81     fwrite(&dmy, sizeof(int), 1, fp);
82
83     dmy = sizeof(int)*4;
84     fwrite(&dmy, sizeof(int), 1, fp);
85     fwrite(&szVal[0], sizeof(int), 1, fp);
86     fwrite(&szVal[1], sizeof(int), 1, fp);
87     fwrite(&szVal[2], sizeof(int), 1, fp);
88     fwrite(&ncomp, sizeof(int), 1, fp);
89     fwrite(&dmy, sizeof(int), 1, fp);
90   //Bunary
91   } else {
92     fwrite(&ngrid, sizeof(int), 1, fp);
93     fwrite(&szVal[0], sizeof(int), 1, fp);
94     fwrite(&szVal[1], sizeof(int), 1, fp);
95     fwrite(&szVal[2], sizeof(int), 1, fp);
96     fwrite(&ncomp, sizeof(int), 1, fp);
97   }
98   //フィールドデータ出力
99
100   if( val->getDataType() == CIO::E_CIO_FLOAT32 ) {
101     cio_TypeArray<float> *data = dynamic_cast<cio_TypeArray<float>*>(val);
102     write_Func(fp, data, szVal, ncomp);
103   } else if( val->getDataType() == CIO::E_CIO_FLOAT64 ) {
104     cio_TypeArray<double> *data = dynamic_cast<cio_TypeArray<double>*>(val);
105     write_Func(fp, data, szVal, ncomp);
106   }
107
108   return CIO::E_CIO_SUCCESS;
109 }
```

6.6.3.6 template<class T > CIO_INLINE void cio_DFI_PLOT3D::write_Func (FILE * *fp*, cio_TypeArray< T > * *data*, const int *sz[3]*, int *ncomp*)

cio_Plot3d_inline.h の 149 行で定義されています。

参照先 CIO::E_CIO_IJKN, CIO::E_CIO_OUTPUT_TYPE_ASCII, CIO::E_CIO_OUTPUT_TYPE_FBINARY, cio_-Array::getArrayShape(), cio_DFI::m_output_type, と cio_TypeArray< T >::val().

```
151 {
152
153   //IJNK
154   if( data->getArrayShape() == CIO::E_CIO_IJKN ) {
155     //ascii
156     if( m_output_type == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
157       for(int n=0; n<ncomp; n++) {
158       for(int k=0; k<sz[2]; k++) {
159       for(int j=0; j<sz[1]; j++) {
160       for(int i=0; i<sz[0]; i++) {
161         fprintf(fp,"%15.6E\n",data->val(i,j,k,n));
162       }}}}
163
164     //Fortran Binary
165     } else if( m_output_type == CIO::E_CIO_OUTPUT_TYPE_FBINARY ) {
166       unsigned int dmy;
167       dmy = sizeof(T)*(sz[0]*sz[1]*sz[2]*ncomp);
168       fwrite(&dmy, sizeof(int), 1, fp);
169       for(int n=0; n<ncomp; n++) {
170       for(int k=0; k<sz[2]; k++) {
171       for(int j=0; j<sz[1]; j++) {
172       for(int i=0; i<sz[0]; i++) {
173         fwrite(&data->val(i,j,k,n), sizeof(T), 1, fp);
174       }}}}
175       fwrite(&dmy, sizeof(int), 1, fp);
176
177     //binary
178     } else {
179       for(int n=0; n<ncomp; n++) {
180       for(int k=0; k<sz[2]; k++) {
181       for(int j=0; j<sz[1]; j++) {
182       for(int i=0; i<sz[0]; i++) {
183         fwrite(&data->val(i,j,k,n), sizeof(T), 1, fp);
184       }}}}
185     }
186
187   //NIJK
188   } else {
189     //ascii
```

```
190     if( m_output_type == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
191       for(int n=0; n<ncomp; n++) {
192       for(int k=0; k<sz[2]; k++) {
193       for(int j=0; j<sz[1]; j++) {
194       for(int i=0; i<sz[0]; i++) {
195         fprintf(fp,"%15.6E\n",data->val(n,i,j,k));
196       }}}}
197
198     //Fortran Binary
199     } else if( m_output_type == CIO::E_CIO_OUTPUT_TYPE_FBINARY ) {
200       unsigned int dmy;
201       dmy = sizeof(T)*(sz[0]*sz[1]*sz[2]*ncomp);
202       fwrite(&dmy, sizeof(int), 1, fp);
203       for(int n=0; n<ncomp; n++) {
204       for(int k=0; k<sz[2]; k++) {
205       for(int j=0; j<sz[1]; j++) {
206       for(int i=0; i<sz[0]; i++) {
207         fwrite(&data->val(n,i,j,k), sizeof(T), 1, fp);
208       }}}}
209       fwrite(&dmy, sizeof(int), 1, fp);
210
211     //binary
212     } else {
213       for(int n=0; n<ncomp; n++) {
214       for(int k=0; k<sz[2]; k++) {
215       for(int j=0; j<sz[1]; j++) {
216       for(int i=0; i<sz[0]; i++) {
217         fwrite(&data->val(n,i,j,k), sizeof(T), 1, fp);
218       }}}}
219     }
220   }
221
222 }
```

6.6.3.7 template<class T > void cio_DFI_PLOT3D::write_Func (FILE * *fp*, cio_TypeArray< T > * *data*, const int *sz[3]*, int *ncomp*) [protected]

func data 出力

引数

in	*fp*	出力ファイルポインタ
in	*data*	出力データポインタ
in	*sz*	出力データのサイズ
in	*ncomp*	出力成分数

参照元 write_DataRecord().

6.6.3.8 bool cio_DFI_PLOT3D::write_GridData ( ) [protected]

Grid data file 出力 コントロール

cio_DFI_PLOT3D.C の 114 行で定義されています。

参照先 CIO::cioPath_isAbsolute(), cio_FileInfo::DataType, cio_DFI::DFI_Domain, cio_DFI::DFI_Finfo, cio_DFI::DFI_MPI, cio_DFI::DFI_Process, cio_FileInfo::DirectoryPath, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, cio_DFI::Generate_FileName(), cio_Domain::GlobalOrigin, cio_Domain::GlobalRegion, cio_Domain::GlobalVoxel, cio_DFI::m_directoryPath, cio_DFI::m_output_fname, cio_DFI::m_RankID, cio_MPI::NumberOfRank, cio_FileInfo::Prefix, cio_Process::RankList, cio_FileInfo::TimeSliceDirFlag, と write_XYZ().

参照元 write_DataRecord().

```
115 {
116
117   bool mio = false;
118   if( DFI_MPI.NumberOfRank > 1 ) mio = true;
119
120   //出力ファイル名の生成
121   std::string fname,tmp;
122   tmp = Generate_FileName(DFI_Finfo.Prefix,m_RankID,-1,"xyz",m_output_fname,mio,
123                           DFI_Finfo.TimeSliceDirFlag);
124   if( CIO::cioPath_isAbsolute(DFI_Finfo.DirectoryPath) ){
125     fname = DFI_Finfo.DirectoryPath + "/" + tmp;
```

```
126   } else {
127     fname = m_directoryPath + "/" + DFI_Finfo.DirectoryPath +"/"+ tmp;
128   }
129
130   printf("grid file name : %s\n",fname.c_str());
131
132   //GRID data file open
133   FILE* fp=NULL;
134   if( (fp = fopen(fname.c_str(),"w"))  == NULL ) {
135     printf("\tCan't open file.(%s)\n",fname.c_str());
136     return false;
137   }
138
139   //xyz を求めて出力
140   int sz[3];
141   for(int i=0; i<3; i++) sz[i] = DFI_Process.RankList[m_RankID].VoxelSize[i]+1;
142
143   printf("ID : %d sz : %d %d %d\n",m_RankID,sz[0],sz[1],sz[2]);
144
145   if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT32 ) {
146     float pit[3],org[3];
147     for(int i=0; i<3; i++) {
148       pit[i]=(float)DFI_Domain.GlobalRegion[i]/(float)DFI_Domain.GlobalVoxel[i];
149       org[i]=(float)DFI_Domain.GlobalOrigin[i]-pit[i]*0.5;
150     }
151     //xyz を計算して出力
152     write_XYZ(fp,org,pit,sz);
153   }else if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT64 ) {
154     double pit[3],org[3];
155     for(int i=0; i<3; i++) {
156       pit[i]=(double)DFI_Domain.GlobalRegion[i]/(double)DFI_Domain.GlobalVoxel[i];
157       org[i]=(double)DFI_Domain.GlobalOrigin[i]-pit[i]*0.5;
158     }
159     //xyz を計算して出力
160     write_XYZ(fp,org,pit,sz);
161   }
162
163   //file close
164   fclose(fp);
165
166   return true;
167
168 }
```

6.6.3.9 CIO::E_CIO_ERRORCODE cio_DFI_PLOT3D::write_HeaderRecord (FILE * *fp*, const unsigned *step*, const double *time*, const int *RankID*) [protected], [virtual]

SPH ヘッダファイルの出力

引数

in	*fp*	ファイルポインタ
in	*step*	ステップ番号
in	*time*	時刻
in	*RankID*	ランク番号

戻り値

error code

cio_DFIを実装しています。

cio_DFI_PLOT3D.C の 36 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
40 {
41   return CIO::E_CIO_SUCCESS;
42 }
```

6.6.3.10 template<class T > CIO_INLINE void cio_DFI_PLOT3D::write_XYZ (FILE * *fp*, T * *org*, T * *pit*, int *sz[3]*)

cio_Plot3d_inline.h の 33 行で定義されています。

参照先 CIO::E_CIO_OUTPUT_TYPE_ASCII, CIO::E_CIO_OUTPUT_TYPE_FBINARY, と cio_DFI::m_output_type.

```
34 {
35
36   int ngrid=1;
37   T xyz;
38
39   //ascii
40   if( m_output_type == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
41     fprintf(fp,"%5d\n",ngrid);
42     fprintf(fp,"%5d%5d%5d\n",sz[0],sz[1],sz[2]);
43
44     //x
45     for(int k=0; k<sz[2]; k++) {
46     for(int j=0; j<sz[1]; j++) {
47     for(int i=0; i<sz[0]; i++) {
48       xyz = org[0]+pit[0]*(T)i;
49       fprintf(fp,"%15.6E\n",xyz);
50     }}}
51
52     //y
53     for(int k=0; k<sz[2]; k++) {
54     for(int j=0; j<sz[1]; j++) {
55     for(int i=0; i<sz[0]; i++) {
56       xyz = org[1]+pit[1]*(T)j;
57       fprintf(fp,"%15.6E\n",xyz);
58     }}}
59
60     //z
61     for(int k=0; k<sz[2]; k++) {
62     for(int j=0; j<sz[1]; j++) {
63     for(int i=0; i<sz[0]; i++) {
64       xyz = org[2]+pit[2]*(T)k;
65       fprintf(fp,"%15.6E\n",xyz);
66     }}}
67
68   //Fortran Binary
69   } else if( m_output_type == CIO::E_CIO_OUTPUT_TYPE_FBINARY ) {
70     unsigned int dmy;
71     dmy = sizeof(int);
72     fwrite(&dmy, sizeof(int), 1, fp);
73     fwrite(&ngrid, sizeof(int), 1, fp);
74     fwrite(&dmy, sizeof(int), 1, fp);
75     dmy = sizeof(int)*3;
76     fwrite(&dmy, sizeof(int), 1, fp);
77     fwrite(&sz[0], sizeof(int), 1, fp);
78     fwrite(&sz[1], sizeof(int), 1, fp);
79     fwrite(&sz[2], sizeof(int), 1, fp);
80     fwrite(&dmy, sizeof(int), 1, fp);
81
82     dmy = sizeof(T)*(sz[0]*sz[1]*sz[2]*3);
83     fwrite(&dmy, sizeof(int), 1, fp);
84     //x
85     for(int k=0; k<sz[2]; k++) {
86     for(int j=0; j<sz[1]; j++) {
87     for(int i=0; i<sz[0]; i++) {
88       xyz = org[0]+pit[0]*(T)i;
89       fwrite(&xyz, sizeof(T), 1, fp);
90     }}}
91
92     //y
93     for(int k=0; k<sz[2]; k++) {
94     for(int j=0; j<sz[1]; j++) {
95     for(int i=0; i<sz[0]; i++) {
96       xyz = org[1]+pit[1]*(T)j;
97       fwrite(&xyz, sizeof(T), 1, fp);
98     }}}
99
100     //z
101     for(int k=0; k<sz[2]; k++) {
102     for(int j=0; j<sz[1]; j++) {
103     for(int i=0; i<sz[0]; i++) {
104       xyz = org[2]+pit[2]*(T)k;
105       fwrite(&xyz, sizeof(T), 1, fp);
106     }}}
107     fwrite(&dmy, sizeof(int), 1, fp);
108
109   //Binary
110   } else {
111     fwrite(&ngrid, sizeof(int), 1, fp);
112     fwrite(&sz[0], sizeof(int), 1, fp);
113     fwrite(&sz[1], sizeof(int), 1, fp);
114     fwrite(&sz[2], sizeof(int), 1, fp);
115
116     //x
```

```
117     for(int k=0; k<sz[2]; k++) {
118     for(int j=0; j<sz[1]; j++) {
119     for(int i=0; i<sz[0]; i++) {
120       xyz = org[0]+pit[0]*(T)i;
121       fwrite(&xyz, sizeof(T), 1, fp);
122     }}}
123
124     //y
125     for(int k=0; k<sz[2]; k++) {
126     for(int j=0; j<sz[1]; j++) {
127     for(int i=0; i<sz[0]; i++) {
128       xyz = org[1]+pit[1]*(T)j;
129       fwrite(&xyz, sizeof(T), 1, fp);
130     }}}
131
132     //z
133     for(int k=0; k<sz[2]; k++) {
134     for(int j=0; j<sz[1]; j++) {
135     for(int i=0; i<sz[0]; i++) {
136       xyz = org[2]+pit[2]*(T)k;
137       fwrite(&xyz, sizeof(T), 1, fp);
138     }}}
139
140   }
141
142 }
```

6.6.3.11 template<class T > void cio_DFI_PLOT3D::write_XYZ (FILE * *fp*, T * *org*, T * *pit*, int *sz[3]*) [protected]

xyz を計算して出力

引数

in	*fp*	出力ファイルポインタ
in	*org*	原点座標値
in	*pit*	ピッチ
in	*sz*	サイズ

参照元 write_GridData().

6.6.4 変数

6.6.4.1 bool cio_DFI_PLOT3D::m_OutputGrid [protected]

plot3d grid file 出力指示

cio_DFI_PLOT3D.h の 24 行で定義されています。

参照元 cio_DFI_PLOT3D(), と write_DataRecord().

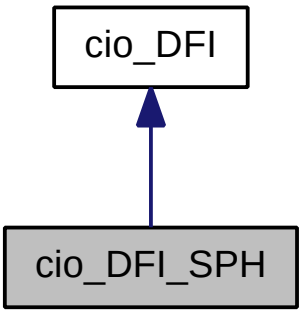
このクラスの説明は次のファイルから生成されました:

- cio_DFI_PLOT3D.h
- cio_DFI_PLOT3D.C
- cio_Plot3d_inline.h

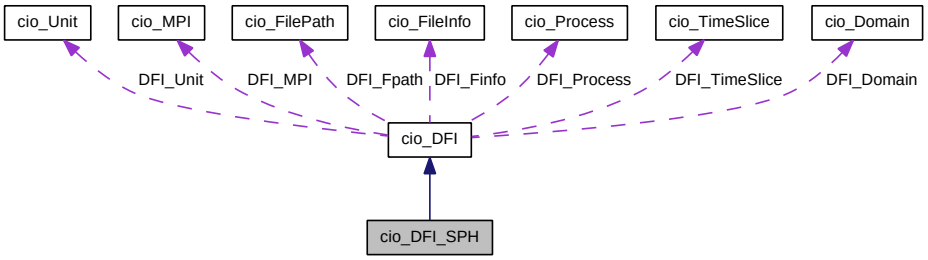
6.7 クラス cio_DFI_SPH

```
#include <cio_DFI_SPH.h>
```

cio_DFI_SPH に対する継承グラフ



cio_DFI_SPH のコラボレーション図



Public メソッド

- cio_DFI_SPH ()
- cio_DFI_SPH (const cio_FileInfo F_Info, const cio_FilePath F_Path, const cio_Unit unit, const cio_Domain domain, const cio_MPI mpi, const cio_TimeSlice TSlice, const cio_Process process)

  コンストラクタ
- ~cio_DFI_SPH ()

Protected 型

- enum DataDims { _DATA_UNKNOWN =0, _SCALAR, _VECTOR }
- enum RealType { _REAL_UNKNOWN =0, _FLOAT, _DOUBLE }

Protected メソッド

- CIO::E_CIO_ERRORCODE read_HeaderRecord (FILE *fp, bool matchEndian, unsigned step, const int head[3], const int tail[3], int gc, int voxsize[3], double &time)

  sph **ファイルのヘッダーレコード読込み**
- CIO::E_CIO_ERRORCODE read_Datarecord (FILE *fp, bool matchEndian, cio_Array *buf, int head[3], int nz, cio_Array *&src)

  フィールドデータファイルのデータレコード読込み
- CIO::E_CIO_ERRORCODE read_averaged (FILE *fp, bool matchEndian, unsigned step, unsigned &avr_step, double &avr_time)

  sph **ファイルの***Average* **データレコードの読込み**
- CIO::E_CIO_ERRORCODE write_HeaderRecord (FILE *fp, const unsigned step, const double time, const int RankID)

  SPH **ヘッダファイルの出力**

- CIO::E_CIO_ERRORCODE write_DataRecord (FILE *fp, cio_Array *val, const int gc, const int RankID)
  SPH **データレコードの出力**
- CIO::E_CIO_ERRORCODE write_averaged (FILE *fp, const unsigned step_avr, const double time_avr)
  Average **レコードの出力**

Additional Inherited Members

6.7.1 説明

cio_DFI_SPH.h の 20 行で定義されています。

6.7.2 列挙型

6.7.2.1 enum cio_DFI_SPH::DataDims [protected]

data dims(scalar or vector)

列挙型の値

_DATA_UNKNOWN
_SCALAR
_VECTOR

cio_DFI_SPH.h の 25 行で定義されています。

```
25 {_DATA_UNKNOWN=0, _SCALAR, _VECTOR} DataDims;
```

6.7.2.2 enum cio_DFI_SPH::RealType [protected]

data type(float or double)

列挙型の値

_REAL_UNKNOWN
_FLOAT
_DOUBLE

cio_DFI_SPH.h の 28 行で定義されています。

```
28 {_REAL_UNKNOWN=0, _FLOAT, _DOUBLE} RealType;
```

6.7.3 コンストラクタとデストラクタ

6.7.3.1 cio_DFI_SPH::cio_DFI_SPH ( )

コンストラクタ

cio_DFI_SPH.C の 20 行で定義されています。

```
21 {
22
23 }
```

6.7.3.2 cio_DFI_SPH::cio_DFI_SPH (const cio_FileInfo *F_Info,* const cio_FilePath *F_Path,* const cio_Unit *unit,* const cio_Domain *domain,* const cio_MPI *mpi,* const cio_TimeSlice *TSlice,* const cio_Process *process*) [inline]

コンストラクタ

引数

in	*F_Info*	FileInfo
in	*F_Path*	FilePath
in	*unit*	Unit
in	*domain*	Domain
in	*mpi*	MPI
in	*TSlice*	TimeSlice
in	*process*	Process

cio_DFI_SPH.h の 45 行で定義されています。

参照先 cio_DFI::DFI_Domain, cio_DFI::DFI_Finfo, cio_DFI::DFI_Fpath, cio_DFI::DFI_MPI, cio_DFI::DFI_Process, cio_DFI::DFI_TimeSlice, cio_DFI::DFI_Unit, と cio_DFI::m_bgrid_interp_flag.

```
52   {
53     DFI_Finfo      = F_Info;
54     DFI_Fpath      = F_Path;
55     DFI_Unit       = unit;
56     DFI_Domain     = domain;
57     DFI_MPI        = mpi;
58     DFI_TimeSlice  = TSlice;
59     DFI_Process    = process;
60     m_bgrid_interp_flag = false;
61   };
```

6.7.3.3 cio_DFI_SPH::~cio_DFI_SPH ( )

デストラクタ

cio_DFI_SPH.C の 28 行で定義されています。

```
29 {
30
31 }
```

6.7.4 関数

6.7.4.1 CIO::E_CIO_ERRORCODE cio_DFI_SPH::read_averaged (FILE * *fp,* bool *matchEndian,* unsigned *step,* unsigned & *avr_step,* double & *avr_time*) [protected],[virtual]

sph ファイルのAverage データレコードの読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*step*	読込み step 番号
out	*avr_step*	平均ステップ
out	*avr_time*	平均タイム

戻り値

error code

cio_DFIを実装しています。

cio_DFI_SPH.C の 249 行で定義されています。

参照先 BSWAP32, BSWAP64, cio_FileInfo::DataType, cio_DFI::DFI_Finfo, CIO::E_CIO_ERROR_READ_SPH_REC7, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, と CIO::E_CIO_SUCCESS.

```
254 {
255
256   unsigned int dmy,type_dmy;
257
258   if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT32 ) type_dmy = 8;
259   if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT64 ) type_dmy = 16;
260   if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC7; }
261   if( !matchEndian ) BSWAP32(dmy);
262   if( dmy != type_dmy ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_REC7; }
263   if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT32 ) {
264     int r_step;
265     if( fread(&r_step, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC7; }
266     if( !matchEndian ) BSWAP32(r_step);
267     step_avr=(unsigned)r_step;
268   } else if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT64 ) {
269     long long r_step;
270     if( fread(&r_step, sizeof(long long), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC7; }
271     if( !matchEndian ) BSWAP64(r_step);
272     step_avr=(unsigned)r_step;
273   }
274   if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT32 ) {
275     float r_time;
276     if( fread(&r_time, sizeof(float), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC7; }
277     if( !matchEndian ) BSWAP32(r_time);
278     time_avr = (double)r_time;
279   } else if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT64 ) {
280     double r_time;
281     if( fread(&r_time, sizeof(double), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC7; }
282     if( !matchEndian ) BSWAP64(r_time);
283     time_avr = r_time;
284   }
285   if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC7; }
286   if( !matchEndian ) BSWAP32(dmy);
287   if( dmy != type_dmy ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_REC7; }
288
289   return CIO::E_CIO_SUCCESS;
290 }
```

6.7.4.2 CIO::E_CIO_ERRORCODE cio_DFI_SPH::read_Datarecord (FILE * *fp*, bool *matchEndian*, cio_Array * *buf*, int *head[3]*, int *nz*, cio_Array *& *src*) [protected], [virtual]

フィールドデータファイルのデータレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*buf*	読込み用バッファ
in	*head*	読込みバッファHeadIndex
in	*nz*	z 方向のボクセルサイズ（実セル＋ガイドセル＊２）
out	*src*	読み込んだデータを格納した配列のポインタ

戻り値

error code

cio_DFIを実装しています。

cio_DFI_SPH.C の 208 行で定義されています。

参照先 BSWAP32, cio_Array::copyArray(), CIO::E_CIO_ERROR_READ_SPH_REC6, CIO::E_CIO_SUCCESS, cio_Array::getArrayLength(), cio_Array::readBinary(), と cio_Array::setHeadIndex().

```
214 {
215
216   // 1 層ずつ読み込み
217   int hzB = head[2];
218
```

```
219   // fortran record の読込み
220   int idmy;
221   if( fread(&idmy,sizeof(int),1,fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC6; }
222   if( !matchEndian ) BSWAP32(idmy);
223
224   for( int k=0; k<nz; k++ ) {
225     //head インデクスをずらす
226     head[2]=hzB+k;
227     buf->setHeadIndex(head);
228
229     //1層読み込み
230     size_t ndata = buf->getArrayLength();
231     if( buf->readBinary(fp,matchEndian) != ndata ) return
      CIO::E_CIO_ERROR_READ_SPH_REC6;
232
233     // コピー
234     buf->copyArray(src);
235   }
236
237   // fortran record の読込み
238   if( fread(&idmy,sizeof(int),1,fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC6; }
239   if( !matchEndian ) BSWAP32(idmy);
240
241   return CIO::E_CIO_SUCCESS;
242
243 }
```

6.7.4.3 CIO::E_CIO_ERRORCODE cio_DFI_SPH::read_HeaderRecord (FILE * *fp*, bool *matchEndian*, unsigned *step*, const int *head[3]*, const int *tail[3]*, int *gc*, int *voxsize[3]*, double & *time*) [protected], [virtual]

sph ファイルのヘッダーレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	エンディアンチェックフラグ true:合致
in	*step*	ステップ番号
in	*head*	dfi のHeadIndex
in	*tail*	dfi のTailIndex
in	*gc*	dfi のガイドセル数
out	*voxsize*	voxsize
out	*time*	時刻

戻り値

error code

cio_DFIを実装しています。

cio_DFI_SPH.C の 36 行で定義されています。

参照先 _DOUBLE, _FLOAT, _SCALAR, _VECTOR, BSWAP32, BSWAP64, cio_FileInfo::Component, cio_FileInfo::DataType, cio_DFI::DFI_Finfo, CIO::E_CIO_ERROR_NOMATCH_ENDIAN, CIO::E_CIO_ERROR_READ_SPH_FILE, CIO::E_CIO_ERROR_READ_SPH_REC1, CIO::E_CIO_ERROR_READ_SPH_REC2, CIO::E_CIO_ERROR_READ_SPH_REC3, CIO::E_CIO_ERROR_READ_SPH_REC4, CIO::E_CIO_ERROR_READ_SPH_REC5, CIO::E_CIO_ERROR_UNMATCH_VOXELSIZE, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, と CIO::E_CIO_SUCCESS.

```
44 {
45
46   unsigned int dmy,type_dmy;
47
48 //REC1
49   if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC1; }
50   if( !matchEndian ) BSWAP32(dmy);
51   if( dmy != 8 ) {
52     BSWAP32(dmy);
53     if( dmy != 8 ) {
```

```
54        fclose(fp);
55        return CIO::E_CIO_ERROR_READ_SPH_REC1;
56      } else {
57        fclose(fp);
58        return CIO::E_CIO_ERROR_NOMATCH_ENDIAN;
59      }
60    }
61
62    DataDims data_dims;
63    if( fread(&data_dims, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
       CIO::E_CIO_ERROR_READ_SPH_REC1; }
64    if( !matchEndian ) BSWAP32(data_dims);
65    if( data_dims == _SCALAR && DFI_Finfo.Component != 1 ) { fclose(fp); return
       CIO::E_CIO_ERROR_READ_SPH_REC1; }
66    if( data_dims == _VECTOR && DFI_Finfo.Component <= 1 ) { fclose(fp); return
       CIO::E_CIO_ERROR_READ_SPH_REC1; }
67
68    int real_type;
69
70    if( fread(&real_type, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
       CIO::E_CIO_ERROR_READ_SPH_REC1; }
71    if( !matchEndian ) BSWAP32(real_type);
72    if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
       CIO::E_CIO_ERROR_READ_SPH_REC1; }
73    if( !matchEndian ) BSWAP32(dmy);
74    if( dmy != 8 ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_REC1; }
75
76    if( real_type == _FLOAT ) {
77      if( DFI_Finfo.DataType != CIO::E_CIO_FLOAT32 ) return
       CIO::E_CIO_ERROR_READ_SPH_REC1;
78      type_dmy=12;
79    } else if( real_type == _DOUBLE) {
80      if( DFI_Finfo.DataType != CIO::E_CIO_FLOAT64 ) return
       CIO::E_CIO_ERROR_READ_SPH_REC1;
81      type_dmy=24;
82    }
83
84  //REC2
85  //ボクセルサイズ
86    if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
       CIO::E_CIO_ERROR_READ_SPH_REC2; }
87    if( !matchEndian ) BSWAP32(dmy);
88    if( dmy != type_dmy ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_REC2; }
89    if( real_type == _FLOAT ) {
90      if( fread(voxsize, sizeof(int), 3, fp) != 3 ){fclose(fp);return
       CIO::E_CIO_ERROR_READ_SPH_REC2;}
91      if( !matchEndian ) {
92        BSWAP32(voxsize[0]);
93        BSWAP32(voxsize[1]);
94        BSWAP32(voxsize[2]);
95      }
96    } else if( real_type == _DOUBLE ) {
97      long long tmp[3];
98      if( fread(tmp, sizeof(long long), 3, fp) != 3 ){fclose(fp);return
       CIO::E_CIO_ERROR_READ_SPH_REC2;}
99      if( !matchEndian ) {
100       BSWAP64(tmp[0]);
101       BSWAP64(tmp[1]);
102       BSWAP64(tmp[2]);
103     }
104     voxsize[0]=(int)tmp[0];
105     voxsize[1]=(int)tmp[1];
106     voxsize[2]=(int)tmp[2];
107   }
108   if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
       CIO::E_CIO_ERROR_READ_SPH_REC2; }
109   if( !matchEndian ) BSWAP32(dmy);
110   if( dmy != type_dmy ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_FILE; }
111
112 //REC3
113 //原点座標
114   if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
       CIO::E_CIO_ERROR_READ_SPH_REC3; }
115   if( !matchEndian ) BSWAP32(dmy);
116   if( dmy != type_dmy ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_REC3; }
117   if( real_type == _FLOAT ) {
118     float voxorg[3];
119     if( fread(voxorg, sizeof(float), 3, fp) != 3 ){fclose(fp);return
       CIO::E_CIO_ERROR_READ_SPH_REC3;}
120     if( !matchEndian ) {
121       BSWAP32(voxorg[0]);
122       BSWAP32(voxorg[1]);
123       BSWAP32(voxorg[2]);
124     }
125   } else if( real_type == _DOUBLE ) {
126     double voxorg[3];
127     if( fread(voxorg, sizeof(double), 3, fp) != 3 ){fclose(fp);return
```

```
      CIO::E_CIO_ERROR_READ_SPH_REC3;}
128     if( !matchEndian ) {
129       BSWAP64(voxorg[0]);
130       BSWAP64(voxorg[1]);
131       BSWAP64(voxorg[2]);
132     }
133   }
134   if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC3; }
135   if( !matchEndian ) BSWAP32(dmy);
136   if( dmy != type_dmy ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_REC3; }
137
138 //REC4
139 //pit
140   if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC4; }
141   if( !matchEndian ) BSWAP32(dmy);
142   if( dmy != type_dmy ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_REC4; }
143   if( real_type == _FLOAT ) {
144     float voxpit[3];
145     if( fread(voxpit, sizeof(float), 3, fp) != 3 ){fclose(fp);return
      CIO::E_CIO_ERROR_READ_SPH_REC4;}
146     if( !matchEndian ) {
147       BSWAP32(voxpit[0]);
148       BSWAP32(voxpit[1]);
149       BSWAP32(voxpit[2]);
150     }
151   } else if( real_type == _DOUBLE ) {
152     double voxpit[3];
153     if( fread(voxpit, sizeof(double), 3, fp) != 3 ){fclose(fp);return
      CIO::E_CIO_ERROR_READ_SPH_REC4;}
154     if( !matchEndian ) {
155       BSWAP64(voxpit[0]);
156       BSWAP64(voxpit[1]);
157       BSWAP64(voxpit[2]);
158     }
159   }
160   if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC4; }
161   if( !matchEndian ) BSWAP32(dmy);
162   if( dmy != type_dmy ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_FILE; }
163
164 //REC5
165 //step,time
166   if( real_type == _FLOAT ) type_dmy = 8;
167   if( real_type == _DOUBLE) type_dmy = 16;
168   if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC5; }
169   if( !matchEndian ) BSWAP32(dmy);
170   if( dmy != type_dmy ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_REC5; }
171   if( real_type == _FLOAT ) {
172     int r_step;
173     if( fread(&r_step, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC5; }
174     if( !matchEndian ) BSWAP32(r_step);
175     if( r_step != step ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_REC5; }
176   } else if( real_type == _DOUBLE ) {
177     long long r_step;
178     if( fread(&r_step, sizeof(long long), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC5; }
179     if( !matchEndian ) BSWAP64(r_step);
180     if( r_step != step ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_REC5; }
181   }
182   if( real_type == _FLOAT ) {
183     float r_time;
184     if( fread(&r_time, sizeof(float), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC5; }
185     if( !matchEndian ) BSWAP32(r_time);
186     time = r_time;
187   } else if( real_type == _DOUBLE ) {
188     double r_time;
189     if( fread(&r_time, sizeof(double), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC5; }
190     if( !matchEndian ) BSWAP64(r_time);
191     time = r_time;
192   }
193   if( fread(&dmy, sizeof(int), 1, fp) != 1 ) { fclose(fp); return
      CIO::E_CIO_ERROR_READ_SPH_REC5; }
194   if( !matchEndian ) BSWAP32(dmy);
195   if( dmy != type_dmy ) { fclose(fp); return CIO::E_CIO_ERROR_READ_SPH_REC5; }
196
197   for(int i=0; i<3; i++) {
198     if( voxsize[i] != (tail[i]-head[i]+1+2*gc) ) return
      CIO::E_CIO_ERROR_UNMATCH_VOXELSIZE;
199   }
200
201   return CIO::E_CIO_SUCCESS;
```

```
202
203 }
```

6.7.4.4 CIO::E_CIO_ERRORCODE cio_DFI_SPH::write_averaged (FILE * *fp*, const unsigned *step_avr*, const double *time_avr*) [protected], [virtual]

Average レコードの出力

引数

in	*fp*	ファイルポインタ
in	*step_avr*	平均ステップ番号
in	*time_avr*	平均時刻

戻り値

error code

cio_DFIを実装しています。

cio_DFI_SPH.C の 428 行で定義されています。

参照先 cio_FileInfo::DataType, cio_DFI::DFI_Finfo, CIO::E_CIO_ERROR_WRITE_SPH_REC7, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, と CIO::E_CIO_SUCCESS.

```
431 {
432   int dType = 0;
433   if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT32 ) dType = 1;
434   if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT64 ) dType = 2;
435
436   unsigned int dmy;
437   int Int_size,Real_size;
438   if ( dType == 1 ) {
439     dmy = 8;
440     Int_size  = sizeof(int);
441     Real_size = sizeof(float);
442   }else{
443     dmy = 16;
444     Int_size  = sizeof(long long);
445     Real_size = sizeof(double);
446   }
447   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) {
448     fclose(fp);
449     return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
450   }
451
452   //averaged step time の出力
453   if( dType == 1 ){
454     //float 型
455     int istep = (int)step_avr;
456     float ttime = (float)time_avr;
457     if( fwrite(&istep, Int_size, 1, fp) != 1 ) {
458       fclose(fp);
459       return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
460     }
461     if( fwrite(&ttime, Real_size, 1, fp) != 1 ) {
462       fclose(fp);
463       return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
464     }
465   } else {
466     //doublet 型
467     long long dstep = (long long)step_avr;
468     double ttime = (double)time_avr;
469     if( fwrite(&dstep, Int_size, 1, fp) != 1 ) {
470       fclose(fp);
471       return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
472     }
473     if( fwrite(&ttime, Real_size, 1, fp) != 1 ) {
474       fclose(fp);
475       return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
476     }
477   }
478
479   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) {
480     fclose(fp);
```

```
481     return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
482   }
483
484   return CIO::E_CIO_SUCCESS;
485
486 }
```

6.7.4.5 CIO::E_CIO_ERRORCODE cio_DFI_SPH::write_DataRecord (FILE * *fp*, cio_Array * *val*, const int *gc*, const int *RankID*) [protected], [virtual]

SPH データレコードの出力

引数

in	*fp*	ファイルポインタ
in	*val*	データポインタ
in	*gc*	ガイドセル
in	*RankID*	ランク番号

戻り値

error code

cio_DFIを実装しています。

cio_DFI_SPH.C の 402 行で定義されています。

参照先 cio_FileInfo::Component, cio_FileInfo::DataType, cio_DFI::DFI_Finfo, cio_DFI::DFI_Process, CIO::E_CIO_ERROR_WRITE_SPH_REC6, CIO::E_CIO_SUCCESS, cio_DFI::get_cio_Datasize(), cio_Process::RankList, と cio_Array::writeBinary().

```
406 {
407
408   CIO::E_CIO_DTYPE Dtype = (CIO::E_CIO_DTYPE)DFI_Finfo.DataType;
409   int Real_size = get_cio_Datasize(Dtype);
410
411   int size[3];
412   for(int i=0; i<3; i++ ) size[i] = (int)DFI_Process.RankList[n].VoxelSize[i]+(int)(2*gc);
413
414   size_t dLen = (size_t)(size[0] * size[1] * size[2]);
415   if( DFI_Finfo.Component > 1 ) dLen *= 3;
416
417   unsigned int dmy = dLen * Real_size;
418
419   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC6;
420   if( val->writeBinary(fp) != dLen ) return CIO::E_CIO_ERROR_WRITE_SPH_REC6;
421   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC6;
422   return CIO::E_CIO_SUCCESS;
423 }
```

6.7.4.6 CIO::E_CIO_ERRORCODE cio_DFI_SPH::write_HeaderRecord (FILE * *fp*, const unsigned *step*, const double *time*, const int *RankID*) [protected], [virtual]

SPH ヘッダファイルの出力

引数

in	*fp*	ファイルポインタ
in	*step*	ステップ番号

in	*time*	時刻
in	*RankID*	ランク番号

戻り値

error code

cio_DFIを実装しています。

cio_DFI_SPH.C の 295 行で定義されています。

参照先 cio_FileInfo::Component, cio_FileInfo::DataType, cio_DFI::DFI_Domain, cio_DFI::DFI_Finfo, cio_DFI::DFI_Process, CIO::E_CIO_ERROR_WRITE_SPH_REC1, CIO::E_CIO_ERROR_WRITE_SPH_REC2, CIO::E_CIO_ERROR_WRITE_SPH_REC3, CIO::E_CIO_ERROR_WRITE_SPH_REC4, CIO::E_CIO_ERROR_WRITE_SPH_REC5, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_SUCCESS, cio_Domain::GlobalOrigin, cio_Domain::GlobalRegion, cio_Domain::GlobalVoxel, cio_FileInfo::GuideCell, と cio_Process::RankList.

```
299 {
300
301   //REC1
302   int svType = 0;
303   if( DFI_Finfo.Component == 1 ) svType = 1;
304   if( DFI_Finfo.Component > 1  ) svType = 2;
305   if( svType == 0 ) return CIO::E_CIO_ERROR_WRITE_SPH_REC1;
306
307   int dType = 0;
308   if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT32 ) dType = 1;
309   if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT64 ) dType = 2;
310   if( dType == 0 ) return CIO::E_CIO_ERROR_WRITE_SPH_REC1;
311
312   unsigned int dmy;
313   dmy = 8;
314   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC1;
315   if( fwrite(&svType, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC1;
316   if( fwrite(&dType, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC1;
317   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC1;
318
319
320   if( dType == 1 ) dmy = 12; //float
321   else             dmy = 24; //double
322
323   //REC2
324   //voxel size
325   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC2;
326   if( dType == 1 ) {
327     int size[3];
328     for(int i=0; i<3; i++ ) size[i] = (int)DFI_Process.RankList[n].VoxelSize[i]+(int)(2*
      DFI_Finfo.GuideCell);
329     if( fwrite(size, sizeof(int), 3, fp) !=3 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC2;
330   } else {
331     long long  size[3];
332     for(int i=0; i<3; i++ ) size[i] = (long long)DFI_Process.RankList[n].VoxelSize[i]+(long long)(2*
      DFI_Finfo.GuideCell);
333     if( fwrite(size, sizeof(long long), 3, fp) !=3 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC2;
334   }
335
336   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC2;
337
338   //REC3
339   //origin
340   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC3;
341   if( dType == 1 ) {
342     float pch[3];
343     for(int i=0; i<3; i++ ) pch[i]=(float)DFI_Domain.GlobalRegion[i]/DFI_Domain.
      GlobalVoxel[i];
344     float org[3];
345     //for(int i=0; i<3; i++ ) org[i]=(float)DFI_Domain.GlobalOrigin[i];
346     for(int i=0; i<3; i++ ) org[i]=(float)DFI_Domain.GlobalOrigin[i]+0.5*pch[i];
347     if( fwrite(org, sizeof(float), 3, fp) !=3 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC3;
348   } else {
```

```
349     double pch[3];
350     for(int i=0; i<3; i++ ) pch[i]=(double)DFI_Domain.GlobalRegion[i]/DFI_Domain.
      GlobalVoxel[i];
351     double org[3];
352     for(int i=0; i<3; i++ ) org[i]=(double)DFI_Domain.GlobalOrigin[i]+0.5*pch[i];
353     if( fwrite(org, sizeof(double), 3, fp) !=3 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC3;
354   }
355   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC3;
356
357   //REC4
358   //pitch
359   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC4;
360   if( dType == 1 ) {
361     float pch[3];
362     for(int i=0; i<3; i++ ) pch[i]=(float)DFI_Domain.GlobalRegion[i]/DFI_Domain.
      GlobalVoxel[i];
363     if( fwrite(pch, sizeof(float), 3, fp) !=3 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC4;
364   } else {
365     double pch[3];
366     for(int i=0; i<3; i++ ) pch[i]=(double)DFI_Domain.GlobalRegion[i]/DFI_Domain.
      GlobalVoxel[i];
367     if( fwrite(pch, sizeof(double), 3, fp) !=3 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC4;
368   }
369   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC4;
370
371   //REC5
372   //step&time
373   int Int_size,Real_size;
374   if ( dType == 1 ) {
375     dmy = 8;
376     Int_size  = sizeof(int);
377     Real_size = sizeof(float);
378   }else{
379     dmy = 16;
380     Int_size  = sizeof(long long);
381     Real_size = sizeof(double);
382   }
383   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC5;
384   if( dType == 1 ){
385     float ttime = (float)time;
386     if( fwrite(&step, Int_size, 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC5;
387     if( fwrite(&ttime, Real_size, 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC5;
388   } else {
389     long long dstep = (long long)step;
390     double ttime = (double)time;
391     if( fwrite(&dstep, Int_size, 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC5;
392     if( fwrite(&ttime, Real_size, 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC5;
393   }
394   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC5;
395
396   return CIO::E_CIO_SUCCESS;
397 }
```

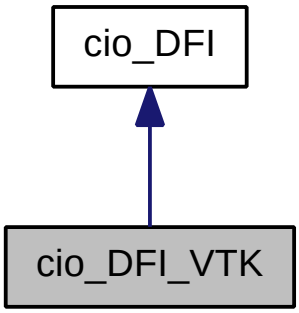
このクラスの説明は次のファイルから生成されました:

- cio_DFI_SPH.h
- cio_DFI_SPH.C

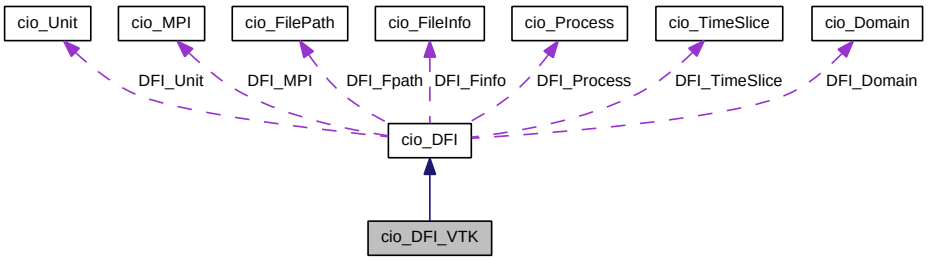
6.8 クラス cio_DFI_VTK

#include <cio_DFI_VTK.h>

cio_DFI_VTK に対する継承グラフ



cio_DFI_VTK のコラボレーション図



Public メソッド

- cio_DFI_VTK ()
- cio_DFI_VTK (const cio_FileInfo F_Info, const cio_FilePath F_Path, const cio_Unit unit, const cio_Domain domain, const cio_MPI mpi, const cio_TimeSlice TSlice, const cio_Process process)

  コンストラクタ
- ~cio_DFI_VTK ()

Protected メソッド

- CIO::E_CIO_ERRORCODE read_HeaderRecord (FILE *fp, bool matchEndian, unsigned step, const int head[3], const int tail[3], int gc, int voxsize[3], double &time)

  sph **ファイルのヘッダーレコード読込み**
- CIO::E_CIO_ERRORCODE read_Datarecord (FILE *fp, bool matchEndian, cio_Array *buf, int head[3], int nz, cio_Array *&src)

  フィールドデータファイルのデータレコード読込み
- CIO::E_CIO_ERRORCODE read_averaged (FILE *fp, bool matchEndian, unsigned step, unsigned &avr_step, double &avr_time)

  sph **ファイルの***Average* **データレコードの読込み**
- CIO::E_CIO_ERRORCODE write_HeaderRecord (FILE *fp, const unsigned step, const double time, const int RankID)

  VTK **ヘッダファイルの出力**
- CIO::E_CIO_ERRORCODE write_DataRecord (FILE *fp, cio_Array *val, const int gc, const int RankID)

  VTK **データレコードの出力**
- CIO::E_CIO_ERRORCODE write_averaged (FILE *fp, const unsigned step_avr, const double time_avr)

  Average **レコードの出力**

Additional Inherited Members

6.8.1 説明

cio_DFI_VTK.h の 20 行で定義されています。

6.8.2 コンストラクタとデストラクタ

6.8.2.1 cio_DFI_VTK::cio_DFI_VTK ( )

コンストラクタ

cio_DFI_VTK.C の 20 行で定義されています。

```
21 {
22
23 }
```

6.8.2.2 cio_DFI_VTK::cio_DFI_VTK (const cio_FileInfo *F_Info,* const cio_FilePath *F_Path,* const cio_Unit *unit,* const cio_Domain *domain,* const cio_MPI *mpi,* const cio_TimeSlice *TSlice,* const cio_Process *process*) [inline]

コンストラクタ

引数

in	*F_Info*	FileInfo
in	*F_Path*	FilePath
in	*unit*	Unit
in	*domain*	Domain
in	*mpi*	MPI
in	*TSlice*	TimeSlice
in	*process*	Process

cio_DFI_VTK.h の 39 行で定義されています。

参照先 cio_DFI::DFI_Domain, cio_DFI::DFI_Finfo, cio_DFI::DFI_Fpath, cio_DFI::DFI_MPI, cio_DFI::DFI_Process, cio_DFI::DFI_TimeSlice, cio_DFI::DFI_Unit, と cio_DFI::m_bgrid_interp_flag.

```
46   {
47     DFI_Finfo      = F_Info;
48     DFI_Fpath      = F_Path;
49     DFI_Unit       = unit;
50     DFI_Domain     = domain;
51     DFI_MPI        = mpi;
52     DFI_TimeSlice  = TSlice;
53     DFI_Process    = process;
54     m_bgrid_interp_flag = true;
55   };
```

6.8.2.3 cio_DFI_VTK::~cio_DFI_VTK ( )

デストラクタ

cio_DFI_VTK.C の 28 行で定義されています。

```
29 {
30
31 }
```

6.8.3 関数

6.8.3.1 CIO::E_CIO_ERRORCODE cio_DFI_VTK::read_averaged (FILE * *fp,* bool *matchEndian,* unsigned *step,* unsigned & *avr_step,* double & *avr_time*) [inline],[protected],[virtual]

sph ファイルのAverage データレコードの読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*step*	読込み step 番号
out	*avr_step*	平均ステップ
out	*avr_time*	平均タイム

戻り値

error code

cio_DFIを実装しています。

cio_DFI_VTK.h の 116 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
121   { return CIO::E_CIO_SUCCESS; };
```

6.8.3.2 CIO::E_CIO_ERRORCODE cio_DFI_VTK::read_Datarecord (FILE * *fp*, bool *matchEndian*, cio_Array * *buf*, int *head[3]*, int *nz*, cio_Array *& *src*) [inline], [protected], [virtual]

フィールドデータファイルのデータレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	true:Endian 一致
in	*buf*	読込み用バッファ
in	*head*	読込みバッファHeadIndex
in	*nz*	z 方向のボクセルサイズ（実セル＋ガイドセル＊２）
out	*src*	読み込んだデータを格納した配列のポインタ

戻り値

error code

cio_DFIを実装しています。

cio_DFI_VTK.h の 98 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
104   { return CIO::E_CIO_SUCCESS; };
```

6.8.3.3 CIO::E_CIO_ERRORCODE cio_DFI_VTK::read_HeaderRecord (FILE * *fp*, bool *matchEndian*, unsigned *step*, const int *head[3]*, const int *tail[3]*, int *gc*, int *voxsize[3]*, double & *time*) [inline], [protected], [virtual]

sph ファイルのヘッダーレコード読込み

引数

in	*fp*	ファイルポインタ
in	*matchEndian*	エンディアンチェックフラグ true:合致
in	*step*	ステップ番号
in	*head*	dfi のHeadIndex
in	*tail*	dfi のTailIndex
in	*gc*	dfi のガイドセル数
out	*voxsize*	voxsize
out	*time*	時刻

戻り値

error code

cio_DFIを実装しています。

cio_DFI_VTK.h の 77 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
85   { return CIO::E_CIO_SUCCESS; };
```

6.8.3.4 CIO::E_CIO_ERRORCODE cio_DFI_VTK::write_averaged (FILE * *fp*, const unsigned *step_avr*, const double *time_avr*) [inline], [protected], [virtual]

Average レコードの出力

引数

in	*fp*	ファイルポインタ
in	*step_avr*	平均ステップ番号
in	*time_avr*	平均時刻

戻り値

error code

cio_DFIを実装しています。

cio_DFI_VTK.h の 159 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
162   { return CIO::E_CIO_SUCCESS; };
```

6.8.3.5 CIO::E_CIO_ERRORCODE cio_DFI_VTK::write_DataRecord (FILE * *fp*, cio_Array * *val*, const int *gc*, const int *RankID*) [protected], [virtual]

VTK データレコードの出力

引数

in	*fp*	ファイルポインタ
in	*val*	データポインタ

in	*gc*	ガイドセル
in	*RankID*	ランク番号

戻り値

error code

cio_DFIを実装しています。

cio_DFI_VTK.C の 110 行で定義されています。

参照先 BSWAPVEC, DBSWAPVEC, CIO::E_CIO_ERROR, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_OUTPUT_TYPE_ASCII, CIO::E_CIO_OUTPUT_TYPE_BINARY, CIO::E_CIO_SUCCESS, cio_Array::getArraySizeInt(), cio_Array::getData(), cio_Array::getDataType(), cio_Array::getNcomp(), cio_DFI::m_output_type, と cio_Array::writeAscii().

```
114 {
115
116   const int* sz = val->getArraySizeInt();
117   size_t dLen = (size_t)sz[0]*(size_t)sz[1]*(size_t)sz[2]*val->getNcomp();
118
119   if( m_output_type == CIO::E_CIO_OUTPUT_TYPE_BINARY ) {
120
121     //出力実数タイプが float のとき
122     if( val->getDataType() == CIO::E_CIO_FLOAT32 ) {
123       float *data = (float*)val->getData();
124       BSWAPVEC(data,dLen);
125       fwrite( data, sizeof(float), dLen, fp );
126
127     //出力実数タイプが double のとき
128     }else if( val->getDataType() == CIO::E_CIO_FLOAT64 ) {
129       double *data = (double*)val->getData();
130       DBSWAPVEC(data,dLen);
131       fwrite( data, sizeof(double), dLen, fp );
132     }
133
134     fprintf( fp, "\n" );
135   } else if( m_output_type == CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
136
137
138     if( val->writeAscii(fp) != dLen ) {
139       return CIO::E_CIO_ERROR;
140     }
141     fprintf( fp, "\n" );
142
143   }
144   return CIO::E_CIO_SUCCESS;
145 }
```

6.8.3.6 CIO::E_CIO_ERRORCODE cio_DFI_VTK::write_HeaderRecord (FILE * *fp*, const unsigned *step*, const double *time*, const int *RankID*) [protected], [virtual]

VTK ヘッダファイルの出力

引数

in	*fp*	ファイルポインタ
in	*step*	ステップ番号
in	*time*	時刻
in	*RankID*	ランク番号

戻り値

error code

cio_DFIを実装しています。

cio_DFI_VTK.C の 36 行で定義されています。

参照先 cio_FileInfo::Component, cio_FileInfo::DataType, cio_DFI::DFI_Domain, cio_DFI::DFI_Finfo, cio_DFI::DFI_Process, CIO::E_CIO_ERROR, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64, CIO::E_CIO_OUTPUT_TYPE_ASCII, CIO::E_CIO_OUTPUT_TYPE_BINARY, CIO::E_CIO_SUCCESS, cio_Domain::GlobalOrigin, cio_Domain::GlobalRegion, cio_Domain::GlobalVoxel, cio_FileInfo::GuideCell, cio_DFI::m_output_type, cio_FileInfo::Prefix, と cio_Process::RankList.

```
40 {
41
42   if( !fp ) return CIO::E_CIO_ERROR;
43
44   fprintf( fp, "# vtk DataFile Version 2.0\n" );
45   fprintf( fp, "step=%d,time=%g\n", step, time );
46
47   if( m_output_type == CIO::E_CIO_OUTPUT_TYPE_BINARY ) {
48     fprintf( fp, "BINARY\n" );
49   } else if( m_output_type ==  CIO::E_CIO_OUTPUT_TYPE_ASCII ) {
50     fprintf( fp, "ASCII\n" );
51   }
52
53   fprintf( fp, "DATASET STRUCTURED_POINTS\n" );
54
55   int imax,jmax,kmax;
56   imax = (int)DFI_Process.RankList[n].VoxelSize[0]+(2*(int)DFI_Finfo.GuideCell);
57   jmax = (int)DFI_Process.RankList[n].VoxelSize[1]+(2*(int)DFI_Finfo.GuideCell);
58   kmax = (int)DFI_Process.RankList[n].VoxelSize[2]+(2*(int)DFI_Finfo.GuideCell);
59   fprintf( fp, "DIMENSIONS %d %d %d\n", imax+1, jmax+1, kmax+1 );
60
61   //double t_org[3];
62   double t_pit[3];
63   for(int i=0; i<3; i++ ) t_pit[i]=DFI_Domain.GlobalRegion[i]/
64                                    (double)DFI_Domain.GlobalVoxel[i];
65   //for(int i=0; i<3; i++ ) t_org[i]=DFI_Domain.GlobalOrigin[i]-(t_pit[0]*0.5);
66   fprintf( fp, "ORIGIN %e %e %e\n",DFI_Domain.GlobalOrigin[0],
67                                    DFI_Domain.GlobalOrigin[1],
68                                    DFI_Domain.GlobalOrigin[2]);
69
70   fprintf( fp, "ASPECT_RATIO %e %e %e\n", t_pit[0], t_pit[1], t_pit[2] );
71
72   //int nw = imax*jmax*kmax;
73   //fprintf( fp, "CELL_DATA %d\n", nw );
74   int nw = (imax+1)*(jmax+1)*(kmax+1);
75   fprintf( fp, "POINT_DATA %d\n", nw );
76
77   if( DFI_Finfo.Component == 1 )
78   {
79     if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT32 ) {
80       fprintf( fp, "SCALARS %s float\n", DFI_Finfo.Prefix.c_str() );
81     } else if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT64 ) {
82       fprintf( fp, "SCALARS %s double\n", DFI_Finfo.Prefix.c_str() );
83     }
84     fprintf( fp, "LOOKUP_TABLE default\n" );
85   }
86   else if( DFI_Finfo.Component == 3 )
87   {
88     if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT32 ) {
89       fprintf( fp, "VECTORS %s float\n", DFI_Finfo.Prefix.c_str() );
90     } else if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT64 ) {
91       fprintf( fp, "VECTORS %s double\n", DFI_Finfo.Prefix.c_str() );
92     }
93   }
94   else
95   {
96     fprintf( fp, "FIELD %s 1\n", DFI_Finfo.Prefix.c_str() );
97     if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT32 ) {
98       fprintf( fp, "%s %d %d float\n", DFI_Finfo.Prefix.c_str(), DFI_Finfo.
     Component, nw );
99     } else if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT64 ) {
100       fprintf( fp, "%s %d %d doubel\n", DFI_Finfo.Prefix.c_str(), DFI_Finfo.
     Component, nw );
101     }
102   }
103
104   return CIO::E_CIO_SUCCESS;
105 }
```

このクラスの説明は次のファイルから生成されました:

- cio_DFI_VTK.h
- cio_DFI_VTK.C

6.9 クラス cio_Domain

`#include <cio_Domain.h>`

Public メソッド

- cio_Domain ()
- cio_Domain (const double *_GlobalOrigin, const double *_GlobalRegion, const int *_GlobalVoxel, const int *_GlobalDivision)
  コンストラクタ
- ~cio_Domain ()
- CIO::E_CIO_ERRORCODE Read (cio_TextParser tpCntl)
  read Domain(proc.dfi)
- CIO::E_CIO_ERRORCODE Write (FILE *fp, const unsigned tab)
  DFI **ファイル***:Domain* **を出力する**

Public 変数

- double GlobalOrigin [3]
  計算空間の起点座標
- double GlobalRegion [3]
  計算空間の各軸方向の長さ
- int GlobalVoxel [3]
  計算領域全体のボクセル数
- int GlobalDivision [3]
  計算領域の分割数
- std::string ActiveSubdomainFile
  ActiveSubdomain **ファイル名**

6.9.1 説明

proc.dfi ファイルの Domain

cio_Domain.h の 19 行で定義されています。

6.9.2 コンストラクタとデストラクタ

6.9.2.1 cio_Domain::cio_Domain ( )

コンストラクタ

cio_Domain.C の 21 行で定義されています。

参照先 ActiveSubdomainFile, GlobalDivision, GlobalOrigin, GlobalRegion, と GlobalVoxel.

```
22 {
23
24   for(int i=0; i<3; i++) GlobalOrigin[i]=0.0;
25   for(int i=0; i<3; i++) GlobalRegion[i]=0.0;
26   for(int i=0; i<3; i++) GlobalVoxel[i]=0;
27   for(int i=0; i<3; i++) GlobalDivision[i]=0;
28   ActiveSubdomainFile="";
29
30 }
```

6.9.2.2 cio_Domain::cio_Domain (const double * *_GlobalOrigin,* const double * *_GlobalRegion,* const int * *_GlobalVoxel,* const int * *_GlobalDivision*)

コンストラクタ

引数

in	*_GlobalOrigin*	起点座標
in	*_GlobalRegion*	各軸方向の長さ
in	*_GlobalVoxel*	ボクセル数
in	*_GlobalDivision*	分割数

cio_Domain.C の 34 行で定義されています。

参照先 GlobalDivision, GlobalOrigin, GlobalRegion, と GlobalVoxel.

```
38 {
39   GlobalOrigin[0]=_GlobalOrigin[0];
40   GlobalOrigin[1]=_GlobalOrigin[1];
41   GlobalOrigin[2]=_GlobalOrigin[2];
42
43   GlobalRegion[0]=_GlobalRegion[0];
44   GlobalRegion[1]=_GlobalRegion[1];
45   GlobalRegion[2]=_GlobalRegion[2];
46
47   GlobalVoxel[0]=_GlobalVoxel[0];
48   GlobalVoxel[1]=_GlobalVoxel[1];
49   GlobalVoxel[2]=_GlobalVoxel[2];
50
51   GlobalDivision[0]=_GlobalDivision[0];
52   GlobalDivision[1]=_GlobalDivision[1];
53   GlobalDivision[2]=_GlobalDivision[2];
54 }
```

6.9.2.3 cio_Domain::∼cio_Domain ( )

デストラクタ

cio_Domain.C の 58 行で定義されています。

```
59 {
60
61 }
```

6.9.3 関数

6.9.3.1 CIO::E_CIO_ERRORCODE cio_Domain::Read (cio_TextParser *tpCntl*)

read Domain(proc.dfi)

引数

in	*tpCntl*	cio_TextParser クラス

戻り値

error code

cio_Domain.C の 66 行で定義されています。

参照先 ActiveSubdomainFile, CIO::E_CIO_ERROR_READ_DFI_GLOBALDIVISION, CIO::E_CIO_ERROR_READ_DFI_GLOBALORIGIN, CIO::E_CIO_ERROR_READ_DFI_GLOBALREGION, CIO::E_CIO_ERROR_READ_DFI_GLOBALVOXEL, CIO::E_CIO_SUCCESS, cio_TextParser::GetValue(), cio_TextParser::GetVector(), GlobalDivision, GlobalOrigin, GlobalRegion, と GlobalVoxel.

参照元 cio_DFI::ReadInit().

```
67 {
68
69   std::string str;
70   std::string label;
```

```
71   double v[3];
72   int iv[3];
73
74   //GlobalOrign
75   label = "/Domain/GlobalOrigin";
76   for (int n=0; n<3; n++) v[n]=0.0;
77   if ( !(tpCntl.GetVector(label, v, 3 )) )
78   {
79     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
80     return CIO::E_CIO_ERROR_READ_DFI_GLOBALORIGIN;
81   }
82   GlobalOrigin[0]=v[0];
83   GlobalOrigin[1]=v[1];
84   GlobalOrigin[2]=v[2];
85
86   //GlobalRegion
87   label = "/Domain/GlobalRegion";
88   for (int n=0; n<3; n++) v[n]=0.0;
89   if ( !(tpCntl.GetVector(label, v, 3 )) )
90   {
91     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
92     return CIO::E_CIO_ERROR_READ_DFI_GLOBALREGION;
93   }
94   GlobalRegion[0]=v[0];
95   GlobalRegion[1]=v[1];
96   GlobalRegion[2]=v[2];
97
98   //Global_Voxel
99   label = "/Domain/GlobalVoxel";
100   for (int n=0; n<3; n++) iv[n]=0;
101   if ( !(tpCntl.GetVector(label, iv, 3 )) )
102   {
103     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
104     return CIO::E_CIO_ERROR_READ_DFI_GLOBALVOXEL;
105   }
106   GlobalVoxel[0]=iv[0];
107   GlobalVoxel[1]=iv[1];
108   GlobalVoxel[2]=iv[2];
109
110   //Global_Division
111   label = "/Domain/GlobalDivision";
112   for (int n=0; n<3; n++) iv[n]=0;
113   if ( !(tpCntl.GetVector(label, iv, 3 )) )
114   {
115     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
116     return CIO::E_CIO_ERROR_READ_DFI_GLOBALDIVISION;
117   }
118   GlobalDivision[0]=iv[0];
119   GlobalDivision[1]=iv[1];
120   GlobalDivision[2]=iv[2];
121
122   //ActiveSubdomain
123   label = "/Domain/ActiveSubdomainFile";
124   if ( !(tpCntl.GetValue(label, &str )) )
125   {
126     str="";
127   }
128   ActiveSubdomainFile=str;
129
130   return CIO::E_CIO_SUCCESS;
131
132 }
```

6.9.3.2 CIO::E_CIO_ERRORCODE cio_Domain::Write (FILE * *fp*, const unsigned *tab*)

DFI ファイル:Domain を出力する

引数

in	*fp*	ファイルポインタ
in	*tab*	インデント

戻り値

true:出力成功 false:出力失敗

cio_Domain.C の 137 行で定義されています。

参照先 _CIO_WRITE_TAB, ActiveSubdomainFile, CIO::E_CIO_SUCCESS, GlobalDivision, GlobalOrigin, GlobalRegion, と GlobalVoxel.

参照元 cio_DFI::WriteProcDfiFile().

```
139 {
140
141   fprintf(fp, "Domain {\n");
142   fprintf(fp, "\n");
143
144   _CIO_WRITE_TAB(fp, tab+1);
145   fprintf(fp, "GlobalOrigin        = (%e, %e, %e)\n",
146           GlobalOrigin[0],
147           GlobalOrigin[1],
148           GlobalOrigin[2]);
149
150   _CIO_WRITE_TAB(fp, tab+1);
151   fprintf(fp, "GlobalRegion        = (%e, %e, %e)\n",
152           GlobalRegion[0],
153           GlobalRegion[1],
154           GlobalRegion[2]);
155
156   _CIO_WRITE_TAB(fp, tab+1);
157   fprintf(fp, "GlobalVoxel         = (%d, %d, %d)\n",
158           GlobalVoxel[0],
159           GlobalVoxel[1],
160           GlobalVoxel[2]);
161
162   _CIO_WRITE_TAB(fp, tab+1);
163   fprintf(fp, "GlobalDivision      = (%d, %d, %d)\n",
164           GlobalDivision[0],
165           GlobalDivision[1],
166           GlobalDivision[2]);
167
168   _CIO_WRITE_TAB(fp, tab+1);
169   fprintf(fp,"ActiveSubdomainFile = \"%s\"\n",ActiveSubdomainFile.c_str());
170
171   fprintf(fp, "\n");
172   fprintf(fp, "}\n");
173   fprintf(fp, "\n");
174
175   return CIO::E_CIO_SUCCESS;
176
177 }
```

6.9.4 変数

6.9.4.1 std::string cio_Domain::ActiveSubdomainFile

ActiveSubdomain ファイル名

cio_Domain.h の 27 行で定義されています。

参照元 cio_Domain(), cio_Process::CreateSubDomainInfo(), Read(), と Write().

6.9.4.2 int cio_Domain::GlobalDivision[3]

計算領域の分割数

cio_Domain.h の 26 行で定義されています。

参照元 cio_Process::CheckReadRank(), cio_Process::CheckStartEnd(), cio_Domain(), cio_Process::CreateRankList(), cio_Process::CreateSubDomainInfo(), cio_DFI::GetDFIGlobalDivision(), cio_MPI::Read(), Read(), cio_DFI::ReadData(), cio_DFI::ReadInit(), Write(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

6.9.4.3 double cio_Domain::GlobalOrigin[3]

計算空間の起点座標

cio_Domain.h の 23 行で定義されています。

参照元 cio_Domain(), Read(), Write(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_ascii_header(), cio_DFI_PLOT3D::write_GridData(), cio_DFI_VTK::write_HeaderRecord(), cio_DFI_SPH::write_HeaderRecord(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

6.9.4.4 double cio_Domain::GlobalRegion[3]

計算空間の各軸方向の長さ

cio_Domain.h の 24 行で定義されています。

参照元 cio_Domain(), Read(), Write(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_ascii_header(), cio_DFI_PLOT3D::write_GridData(), cio_DFI_VTK::write_HeaderRecord(), cio_DFI_SPH::write_HeaderRecord(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

6.9.4.5 int cio_Domain::GlobalVoxel[3]

計算領域全体のボクセル数

cio_Domain.h の 25 行で定義されています。

参照元 cio_Domain(), cio_Process::CreateRankList(), cio_DFI::CreateReadStartEnd(), cio_DFI::GetDFIGlobalVoxel(), Read(), cio_DFI::ReadData(), cio_DFI::ReadInit(), Write(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_ascii_header(), cio_DFI_PLOT3D::write_GridData(), cio_DFI_VTK::write_HeaderRecord(), cio_DFI_SPH::write_HeaderRecord(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

このクラスの説明は次のファイルから生成されました:

- cio_Domain.h
- cio_Domain.C

6.10 クラス cio_FileInfo

`#include <cio_FileInfo.h>`

Public メソッド

- cio_FileInfo ()
- cio_FileInfo (const std::string _DirectoryPath, const CIO::E_CIO_ONOFF _TimeSliceDirFlag, const std::string _Prefix, const CIO::E_CIO_FORMAT _FileFormat, const int _GuideCell, const CIO::E_CIO_DTYPE _DataType, const CIO::E_CIO_ENDIANTYPE _Endian, const CIO::E_CIO_ARRAYSHAPE _ArrayShape, const int _Component)

  コンストラクタ
- ~cio_FileInfo ()
- CIO::E_CIO_ERRORCODE Read (cio_TextParser tpCntl)

  read FileInfo(inde.dfi)
- CIO::E_CIO_ERRORCODE Write (FILE *fp, const unsigned tab)

  DFI **ファイル***:FileInfo* **要素を出力する**
- void setComponentVariable (int pcomp, std::string compName)

  成分名をセットする
- std::string getComponentVariable (int pcomp)

  成分名を取得する

Public 変数

- std::string DirectoryPath
- CIO::E_CIO_ONOFF TimeSliceDirFlag

  TimeSlice on or off.
- std::string Prefix

  ファイル接頭文字
- CIO::E_CIO_FORMAT FileFormat

  ファイルフォーマット *"bov","sph",,,*
- int GuideCell

  仮想セルの数
- CIO::E_CIO_DTYPE DataType

  配列のデータタイプ *"float",,,,*
- CIO::E_CIO_ENDIANTYPE Endian

  エンディアンタイプ *"big","little"*
- CIO::E_CIO_ARRAYSHAPE ArrayShape

  配列形状
- int Component

  成分数
- vector< std::string > ComponentVariable

  成分名

6.10.1 説明

index.dfi ファイルの FileInfo

cio_FileInfo.h の 20 行で定義されています。

6.10.2 コンストラクタとデストラクタ

6.10.2.1 cio_FileInfo::cio_FileInfo ( )

コンストラクタ

cio_FileInfo.C の 20 行で定義されています。

参照先 ArrayShape, Component, DataType, DirectoryPath, CIO::E_CIO_ARRAYSHAPE_UNKNOWN, CIO::E_CIO_DTYPE_UNKNOWN, CIO::E_CIO_ENDIANTYPE_UNKNOWN, CIO::E_CIO_FMT_UNKNOWN, CIO::E_CIO_OFF, Endian, FileFormat, GuideCell, Prefix, と TimeSliceDirFlag.

```
21 {
22   DirectoryPath    ="";
23   TimeSliceDirFlag =CIO::E_CIO_OFF;
24   Prefix           ="";
25   FileFormat       =CIO::E_CIO_FMT_UNKNOWN;
26   GuideCell        =0;
27   DataType         =CIO::E_CIO_DTYPE_UNKNOWN;
28   Endian           =CIO::E_CIO_ENDIANTYPE_UNKNOWN;
29   ArrayShape       =CIO::E_CIO_ARRAYSHAPE_UNKNOWN;
30   Component        =0;
31 }
```

6.10.2.2 cio_FileInfo::cio_FileInfo (const std::string *_DirectoryPath,* const CIO::E_CIO_ONOFF *_TimeSliceDirFlag,* const std::string *_Prefix,* const CIO::E_CIO_FORMAT *_FileFormat,* const int *_GuideCell,* const CIO::E_CIO_DTYPE *_DataType,* const CIO::E_CIO_ENDIANTYPE *_Endian,* const CIO::E_CIO_ARRAYSHAPE *_ArrayShape,* const int *_Component*)

コンストラクタ

引数

in	*_DirectoryPath*	ディレクトリパス
in	*_TimeSliceDir-Flag*	TimeSlice on or off
in	*_Prefix*	ファイル接頭文字
in	*_FileFormat*	ファイルフォーマット
in	*_GuideCell*	仮想セルの数
in	*_DataType*	配列のデータタイプ
in	*_Endian*	エンディアンタイプ
in	*_ArrayShape*	配列形状
in	*_Component*	成分数

cio_FileInfo.C の 35 行で定義されています。

参照先 ArrayShape, Component, DataType, DirectoryPath, Endian, FileFormat, GuideCell, Prefix, と TimeSliceDir-Flag.

```
44 {
45   DirectoryPath    =_DirectoryPath;
46   Prefix           =_Prefix;
47   TimeSliceDirFlag =_TimeSliceDirFlag;
48   FileFormat       =_FileFormat;
49   GuideCell        =_GuideCell;
50   DataType         =_DataType;
51   Endian           =_Endian;
52   ArrayShape       =_ArrayShape;
53   Component        =_Component;
54 }
```

6.10.2.3 cio_FileInfo::~cio_FileInfo ( )

デストラクタ

cio_FileInfo.C の 57 行で定義されています。

```
58 {
59
60 }
```

6.10.3 関数

6.10.3.1 std::string cio_FileInfo::getComponentVariable (int *pcomp*)

成分名を取得する

引数

in	*pcomp*	成分位置 0:u, 1:v, 2:w

戻り値

成分名　成分名が無い場合は空白が返される

cio_FileInfo.C の 79 行で定義されています。

参照先 ComponentVariable.

参照元 cio_DFI::getComponentVariable().

```
80 {
81   std::string CompName="";
82   if(ComponentVariable.size()<pcomp+1) return CompName;
83   return ComponentVariable[pcomp];
84 }
```

6.10.3.2 CIO::E_CIO_ERRORCODE cio_FileInfo::Read (cio_TextParser *tpCntl*)

read FileInfo(inde.dfi)

引数

in	*tpCntl*	cio_TextParser クラス

戻り値

error code

cio_FileInfo.C の 90 行で定義されています。

参照先 ArrayShape, cio_TextParser::chkNode(), Component, ComponentVariable, cio_DFI::ConvDatatypeS2E(), cio_TextParser::countLabels(), DataType, DirectoryPath, CIO::E_CIO_BIG, CIO::E_CIO_ERROR_READ_DFI_ARRAYSHAPE, CIO::E_CIO_ERROR_READ_DFI_COMPONENT, CIO::E_CIO_ERROR_READ_DFI_DATATYPE, CIO::E_CIO_ERROR_READ_DFI_DIRECTORYPATH, CIO::E_CIO_ERROR_READ_DFI_ENDIAN, CIO::E_CIO_ERROR_READ_DFI_FILEFORMAT, CIO::E_CIO_ERROR_READ_DFI_GUIDECELL, CIO::E_CIO_ERROR_READ_DFI_MIN, CIO::E_CIO_ERROR_READ_DFI_NO_MINMAX, CIO::E_CIO_ERROR_READ_DFI_PREFIX, CIO::E_CIO_ERROR_READ_DFI_TIMESLICEDIRECTORY, CIO::E_CIO_FMT_BOV, CIO::E_CIO_FMT_SPH, CIO::E_CIO_FMT_UNKNOWN, CIO::E_CIO_IJKN, CIO::E_CIO_LITTLE, CIO::E_CIO_NIJK, CIO::E_CIO_OFF, CIO::E_CIO_ON, CIO::E_CIO_SUCCESS, Endian, FileFormat, cio_TextParser::GetNodeStr(), cio_TextParser::GetValue(), GuideCell, Prefix, と TimeSliceDirFlag.

参照元 cio_DFI::ReadInit().

```
91 {
92
93   std::string str;
94   std::string label,label_base,label_leaf,label_leaf_leaf;
95   int ct;
96
97   int ncnt=0;
98
99   //Directorypath
100   label = "/FileInfo/DirectoryPath";
101   if ( !(tpCntl.GetValue(label, &str )) )
102   {
103     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
104     return CIO::E_CIO_ERROR_READ_DFI_DIRECTORYPATH;
105   }
106   DirectoryPath=str;
107
108   ncnt++;
109
110   //TimeSilceDirectory
111   label = "/FileInfo/TimeSliceDirectory";
112   if ( !(tpCntl.GetValue(label, &str )) )
113   {
114     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
115     return CIO::E_CIO_ERROR_READ_DFI_TIMESLICEDIRECTORY;
116   }
117
118   if( !strcasecmp(str.c_str(),"on" ) ) {
119     TimeSliceDirFlag=CIO::E_CIO_ON;
120   } else if( !strcasecmp(str.c_str(),"off" ) ) {
121     TimeSliceDirFlag=CIO::E_CIO_OFF;
122   } else {
123     printf("\tCIO Parsing error : fail to get '%s'\n",str.c_str());
124     return CIO::E_CIO_ERROR_READ_DFI_TIMESLICEDIRECTORY;
125   }
126
127   ncnt++;
128
129   //Prefix
130   label = "/FileInfo/Prefix";
131   if ( !(tpCntl.GetValue(label, &str )) )
132   {
133     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
134     return CIO::E_CIO_ERROR_READ_DFI_PREFIX;
135   }
136   Prefix=str;
137
138   ncnt++;
139
140   //FileFormat
141   label = "/FileInfo/FileFormat";
142   if ( !(tpCntl.GetValue(label, &str )) )
143   {
144     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
145     return CIO::E_CIO_ERROR_READ_DFI_FILEFORMAT;
```

```
146   }
147   if( !strcasecmp(str.c_str(),"sph" ) ) {
148     FileFormat=CIO::E_CIO_FMT_SPH;
149   }
150   else if( !strcasecmp(str.c_str(),"bov" ) ) {
151     FileFormat=CIO::E_CIO_FMT_BOV;
152   }
153   else FileFormat=CIO::E_CIO_FMT_UNKNOWN;
154
155   ncnt++;
156
157   //GuidCell
158   label = "/FileInfo/GuideCell";
159   if ( !(tpCntl.GetValue(label, &ct )) )
160   {
161     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
162     return CIO::E_CIO_ERROR_READ_DFI_GUIDECELL;
163   }
164   GuideCell=ct;
165
166   ncnt++;
167
168   //DataType
169   label = "/FileInfo/DataType";
170   if ( !(tpCntl.GetValue(label, &str )) )
171   {
172     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
173     return CIO::E_CIO_ERROR_READ_DFI_DATATYPE;
174   }
175   DataType=cio_DFI::ConvDatatypeS2E(str);
176
177   ncnt++;
178
179   //Endian
180   label = "/FileInfo/Endian";
181   if ( !(tpCntl.GetValue(label, &str )) )
182   {
183     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
184     return CIO::E_CIO_ERROR_READ_DFI_ENDIAN;
185   }
186   if( !strcasecmp(str.c_str(),"little" ) ) {
187     Endian=CIO::E_CIO_LITTLE;
188   }else if( !strcasecmp(str.c_str(),"big" ) ) {
189     Endian=CIO::E_CIO_BIG;
190   }else {
191     printf("\tCIO Parsing error : fail to get '%s'\n",str.c_str());
192     return CIO::E_CIO_ERROR_READ_DFI_ENDIAN;
193   }
194
195   ncnt++;
196
197   //ArrayShape
198   label = "/FileInfo/ArrayShape";
199   if ( !(tpCntl.GetValue(label, &str )) )
200   {
201     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
202     return CIO::E_CIO_ERROR_READ_DFI_ARRAYSHAPE;
203   }
204   if( !strcasecmp(str.c_str(),"ijkn" ) ) {
205     ArrayShape=CIO::E_CIO_IJKN;
206   } else if( !strcasecmp(str.c_str(),"nijk" ) ) {
207     ArrayShape=CIO::E_CIO_NIJK;
208   }else {
209     printf("\tCIO Parsing error : fail to get '%s'\n",str.c_str());
210     return CIO::E_CIO_ERROR_READ_DFI_ARRAYSHAPE;
211   }
212
213   ncnt++;
214
215   //Componet
216   label = "/FileInfo/Component";
217   if ( !(tpCntl.GetValue(label, &ct )) )
218   {
219     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
220     return CIO::E_CIO_ERROR_READ_DFI_COMPONENT;
221   }
222   Component=ct;
223
224   ncnt++;
225
226   //Component Variable
227   int ncomp=0;
228   label_leaf_leaf = "/FileInfo/Variable";
229   if ( tpCntl.chkNode(label_leaf_leaf) )  //があれば
230   {
231     ncomp = tpCntl.countLabels(label_leaf_leaf);
232   }
```

```
233
234   ncnt++;
235
236   label_leaf = "/FileInfo";
237
238   if( ncomp>0 ) {
239     for(int i=0; i<ncomp; i++) {
240       if(!tpCntl.GetNodeStr(label_leaf,ncnt+i,&str))
241       {
242         printf("\tCIO Parsing error : No Elem name\n");
243         return CIO::E_CIO_ERROR_READ_DFI_NO_MINMAX;
244       }
245       if( !strcasecmp(str.substr(0,8).c_str(), "variable") ) {
246         label_leaf_leaf = label_leaf+"/"+str;
247
248         label = label_leaf_leaf + "/name";
249         if ( !(tpCntl.GetValue(label, &str )) ) {
250           printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
251           return CIO::E_CIO_ERROR_READ_DFI_MIN;
252         }
253         else {
254           ComponentVariable.push_back(str);
255         }
256       }
257     }
258   }
259
260   return CIO::E_CIO_SUCCESS;
261 }
```

6.10.3.3 void cio_FileInfo::setComponentVariable (int *pcomp*, std::string *compName*)

成分名をセットする

引数

in	*pcomp*	成分位置 0:u, 1:v, 2:w
in	*compName*	成分名 "u","v","w",,,

cio_FileInfo.C の 64 行で定義されています。

参照先 ComponentVariable.

参照元 cio_DFI::setComponentVariable().

```
66 {
67
68   if( ComponentVariable.size()>pcomp+1 ) {
69     ComponentVariable[pcomp]=compName;
70   } else {
71     for(int i=ComponentVariable.size(); i<pcomp+1; i++) {
72       ComponentVariable.push_back(compName);
73     }
74   }
75 }
```

6.10.3.4 CIO::E_CIO_ERRORCODE cio_FileInfo::Write (FILE * *fp*, const unsigned *tab*)

DFI ファイル:FileInfo 要素を出力する

引数

in	*fp*	ファイルポインタ
in	*tab*	インデント

戻り値

error code

cio_FileInfo.C の 266 行で定義されています。

参照先 _CIO_WRITE_TAB, ArrayShape, Component, ComponentVariable, cio_DFI::ConvDatatypeE2S(), DataType, DirectoryPath, CIO::E_CIO_FMT_BOV, CIO::E_CIO_FMT_SPH, CIO::E_CIO_IJKN, CIO::E_CIO_LITTLE, CIO::E_CIO_OFF, CIO::E_CIO_ON, CIO::E_CIO_SUCCESS, Endian, FileFormat, GuideCell, Prefix, と TimeSliceDirFlag.

参照元 cio_DFI::WriteIndexDfiFile().

```
268 {
269
270   fprintf(fp, "FileInfo {\n");
271   fprintf(fp, "\n");
272
273   _CIO_WRITE_TAB(fp, tab+1);
274   fprintf(fp, "DirectoryPath      = \"%s\"\n", DirectoryPath.c_str());
275
276   _CIO_WRITE_TAB(fp, tab+1);
277   if(        TimeSliceDirFlag == CIO::E_CIO_OFF ) {
278     fprintf(fp, "TimeSliceDirectory = \"off\"\n");
279   } else if( TimeSliceDirFlag == CIO::E_CIO_ON ) {
280     fprintf(fp, "TimeSliceDirectory = \"on\"\n");
281   }
282
283   _CIO_WRITE_TAB(fp, tab+1);
284   fprintf(fp, "Prefix             = \"%s\"\n", Prefix.c_str());
285
286   _CIO_WRITE_TAB(fp, tab+1);
287   if(        FileFormat == CIO::E_CIO_FMT_SPH ) {
288     fprintf(fp, "FileFormat         = \"sph\"\n");
289   } else if( FileFormat == CIO::E_CIO_FMT_BOV ) {
290     fprintf(fp, "FileFormat         = \"bov\"\n");
291   }
292
293   _CIO_WRITE_TAB(fp, tab+1);
294   fprintf(fp, "GuideCell          = %d\n", GuideCell);
295
296   _CIO_WRITE_TAB(fp, tab+1);
297   std::string Dtype = cio_DFI::ConvDatatypeE2S((CIO::E_CIO_DTYPE)DataType);
298   fprintf(fp, "DataType           = \"%s\"\n",Dtype.c_str());
299
300   _CIO_WRITE_TAB(fp, tab+1);
301   if( Endian == CIO::E_CIO_LITTLE) {
302     fprintf(fp, "Endian             = \"little\"\n");
303   } else {
304     fprintf(fp, "Endian             = \"big\"\n");
305   }
306
307   _CIO_WRITE_TAB(fp, tab+1);
308   if( ArrayShape == CIO::E_CIO_IJKN ) {
309     fprintf(fp, "ArrayShape         = \"ijkn\"\n");
310   } else {
311     fprintf(fp, "ArrayShape         = \"nijk\"\n");
312   }
313
314   _CIO_WRITE_TAB(fp, tab+1);
315   fprintf(fp, "Component          = %d\n",Component);
316
317   if( ComponentVariable.size()>0 ) {
318     _CIO_WRITE_TAB(fp, tab+1);
319     fprintf(fp, "Variable[@]{ name  = \"%s\" }\n",ComponentVariable[0].c_str());
320     _CIO_WRITE_TAB(fp, tab+1);
321     fprintf(fp, "Variable[@]{ name  = \"%s\" }\n",ComponentVariable[1].c_str());
322     _CIO_WRITE_TAB(fp, tab+1);
323     fprintf(fp, "Variable[@]{ name  = \"%s\" }\n",ComponentVariable[2].c_str());
324   }
325
326   fprintf(fp, "\n");
327   fprintf(fp, "}\n");
328   fprintf(fp, "\n");
329
330   return CIO::E_CIO_SUCCESS;
331
332 }
```

6.10.4 変数

6.10.4.1 CIO::E_CIO_ARRAYSHAPE cio_FileInfo::ArrayShape

配列形状

cio_FileInfo.h の 32 行で定義されています。

参照元 cio_FileInfo(), cio_DFI::GetArrayShape(), cio_DFI::GetArrayShapeString(), Read(), cio_DFI::ReadData(), cio_DFI::ReadFieldData(), Write(), cio_DFI_BOV::write_ascii_header(), cio_DFI::WriteData(), と cio_DFI::WriteInit().

6.10.4.2 int cio_FileInfo::Component

成分数

cio_FileInfo.h の 33 行で定義されています。

参照元 cio_FileInfo(), cio_DFI::GetNumComponent(), Read(), cio_DFI_SPH::read_HeaderRecord(), cio_DFI::ReadData(), cio_DFI::ReadFieldData(), Write(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_avs_header(), cio_DFI_BOV::write_DataRecord(), cio_DFI_AVS::write_DataRecord(), cio_DFI_SPH::write_DataRecord(), cio_DFI_VTK::write_HeaderRecord(), cio_DFI_SPH::write_HeaderRecord(), cio_DFI::WriteData(), と cio_DFI::WriteInit().

6.10.4.3 vector<std::string> cio_FileInfo::ComponentVariable

成分名

cio_FileInfo.h の 34 行で定義されています。

参照元 getComponentVariable(), Read(), setComponentVariable(), と Write().

6.10.4.4 CIO::E_CIO_DTYPE cio_FileInfo::DataType

配列のデータタイプ "float",,,,

cio_FileInfo.h の 30 行で定義されています。

参照元 cio_FileInfo(), cio_DFI::GetDataType(), cio_DFI::GetDataTypeString(), Read(), cio_DFI_SPH::read_averaged(), cio_DFI_SPH::read_HeaderRecord(), cio_DFI::ReadData(), cio_DFI::ReadFieldData(), Write(), cio_DFI_SPH::write_averaged(), cio_DFI_BOV::write_DataRecord(), cio_DFI_AVS::write_DataRecord(), cio_DFI_SPH::write_DataRecord(), cio_DFI_PLOT3D::write_GridData(), cio_DFI_VTK::write_HeaderRecord(), cio_DFI_SPH::write_HeaderRecord(), cio_DFI::WriteData(), と cio_DFI::WriteInit().

6.10.4.5 std::string cio_FileInfo::DirectoryPath

フィールデータの存在するディレクトリパス index.dfi からの相対パスまたは絶対パス

cio_FileInfo.h の 24 行で定義されています。

参照元 cio_FileInfo(), cio_DFI::Generate_FieldFileName(), Read(), cio_DFI::ReadData(), Write(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_avs_cord(), cio_DFI_AVS::write_avs_header(), cio_DFI_PLOT3D::write_GridData(), cio_DFI::WriteData(), と cio_DFI::WriteInit().

6.10.4.6 CIO::E_CIO_ENDIANTYPE cio_FileInfo::Endian

エンディアンタイプ "big","little"

cio_FileInfo.h の 31 行で定義されています。

参照元 cio_FileInfo(), Read(), cio_DFI::ReadFieldData(), Write(), cio_DFI_BOV::write_ascii_header(), と cio_DFI::WriteInit().

6.10.4.7 CIO::E_CIO_FORMAT cio_FileInfo::FileFormat

ファイルフォーマット "bov","sph",,,

cio_FileInfo.h の 28 行で定義されています。

参照元 cio_FileInfo(), cio_DFI::Generate_FieldFileName(), Read(), cio_DFI::ReadInit(), Write(), cio_DFI::WriteData(), と cio_DFI::WriteInit().

6.10.4.8 int cio_FileInfo::GuideCell

仮想セルの数

cio_FileInfo.h の 29 行で定義されています。

参照元 cio_FileInfo(), Read(), cio_DFI::ReadData(), cio_DFI::ReadFieldData(), Write(), cio_DFI_VTK::write_HeaderRecord(), cio_DFI_SPH::write_HeaderRecord(), cio_DFI::WriteData(), cio_DFI::WriteFieldData(), と cio_DFI::WriteInit().

6.10.4.9 std::string cio_FileInfo::Prefix

ファイル接頭文字

cio_FileInfo.h の 27 行で定義されています。

参照元 cio_FileInfo(), cio_DFI::Generate_FieldFileName(), Read(), Write(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_avs_header(), cio_DFI_PLOT3D::write_GridData(), cio_DFI_VTK::write_HeaderRecord(), cio_DFI::WriteData(), cio_DFI::WriteIndexDfiFile(), と cio_DFI::WriteInit().

6.10.4.10 CIO::E_CIO_ONOFF cio_FileInfo::TimeSliceDirFlag

TimeSlice on or off.

cio_FileInfo.h の 26 行で定義されています。

参照元 cio_FileInfo(), cio_DFI::Generate_Directory_Path(), cio_DFI::Generate_FieldFileName(), Read(), cio_DFI::SetTimeSliceFlag(), Write(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_avs_cord(), cio_DFI_AVS::write_avs_header(), cio_DFI_PLOT3D::write_GridData(), cio_DFI::WriteData(), と cio_DFI::WriteInit().

このクラスの説明は次のファイルから生成されました:

- cio_FileInfo.h
- cio_FileInfo.C

6.11 クラス cio_FilePath

`#include <cio_FilePath.h>`

Public メソッド

- cio_FilePath ()
- cio_FilePath (const std::string _ProcDFIFile)

  コンストラクタ
- ~cio_FilePath ()
- CIO::E_CIO_ERRORCODE Read (cio_TextParser tpCntl)

  read FilePath(inde.dfi)
- CIO::E_CIO_ERRORCODE Write (FILE *fp, const unsigned tab)

  DFI **ファイル***:Process* **を出力する**

Public 変数

- std::string ProcDFIFile
  proc.dfi ファイル名

6.11.1 説明

index.dfi ファイルの FilePath

cio_FilePath.h の 19 行で定義されています。

6.11.2 コンストラクタとデストラクタ

6.11.2.1 cio_FilePath::cio_FilePath ( )

コンストラクタ

cio_FilePath.C の 21 行で定義されています。

参照先 ProcDFIFile.

```
22 {
23   ProcDFIFile="";
24 }
```

6.11.2.2 cio_FilePath::cio_FilePath (const std::string *_ProcDFIFile*)

コンストラクタ

引数

in	*_ProcDFIFile*	proc.dfi ファイル名

cio_FilePath.C の 28 行で定義されています。

参照先 ProcDFIFile.

```
29 {
30   ProcDFIFile=_ProcDFIFile;
31 }
```

6.11.2.3 cio_FilePath::~cio_FilePath ( )

デストラクタ

cio_FilePath.C の 35 行で定義されています。

```
36 {
37
38 }
```

6.11.3 関数

6.11.3.1 CIO::E_CIO_ERRORCODE cio_FilePath::Read (cio_TextParser *tpCntl*)

read FilePath(inde.dfi)

proc.dfi ファイル名の読込み

引数

in	*tpCntl*	cio_TextParser クラス

戻り値

error code

cio_FilePath.C の 43 行で定義されています。

参照先 CIO::E_CIO_ERROR_READ_DFI_FILEPATH_PROCESS, CIO::E_CIO_SUCCESS, cio_TextParser::GetValue(), と ProcDFIFile.

参照元 cio_DFI::ReadInit().

```
44 {
45
46   std::string str;
47   std::string label;
48
49   //Process
50   label = "/FilePath/Process";
51   if ( !(tpCntl.GetValue(label, &str )) )
52   {
53     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
54     return CIO::E_CIO_ERROR_READ_DFI_FILEPATH_PROCESS;
55   }
56   ProcDFIFile=str;
57
58   return CIO::E_CIO_SUCCESS;
59
60 }
```

6.11.3.2 CIO::E_CIO_ERRORCODE cio_FilePath::Write (FILE * *fp*, const unsigned *tab*)

DFI ファイル:Process を出力する

proc.dfi ファイル名の出力

引数

in	*fp*	ファイルポインタ
in	*tab*	インデント

戻り値

error code

cio_FilePath.C の 65 行で定義されています。

参照先 _CIO_WRITE_TAB, CIO::E_CIO_SUCCESS, と ProcDFIFile.

参照元 cio_DFI::WriteIndexDfiFile().

```
67 {
68
69   fprintf(fp, "FilePath {\n");
70   fprintf(fp, "\n");
71
72   _CIO_WRITE_TAB(fp, tab);
73   fprintf(fp, "Process = \"%s\"\n",ProcDFIFile.c_str());
74
75   fprintf(fp, "\n");
76   fprintf(fp, "}\n");
77   fprintf(fp, "\n");
78
79   return CIO::E_CIO_SUCCESS;
80
81 }
```

6.11.4 変数

6.11.4.1 std::string cio_FilePath::ProcDFIFile

proc.dfi ファイル名

cio_FilePath.h の 23 行で定義されています。

参照元 cio_FilePath(), Read(), cio_DFI::ReadInit(), Write(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

このクラスの説明は次のファイルから生成されました:

- cio_FilePath.h
- cio_FilePath.C

6.12 クラス cio_MPI

```
#include <cio_MPI.h>
```

Public メソッド

- cio_MPI ()
- cio_MPI (const int _NumberOfRank, int _NumberOfGroup=0)
  コンストラクタ
- ~cio_MPI ()
- CIO::E_CIO_ERRORCODE Read (cio_TextParser tpCntl, const cio_Domain domain)
  read MPI(proc.dfi)
- CIO::E_CIO_ERRORCODE Write (FILE *fp, const unsigned tab)
  DFI **ファイル***:MPI* **を出力する**

Public 変数

- int NumberOfRank
  プロセス数
- int NumberOfGroup
  グループ数

6.12.1 説明

proc.dfi ファイルの MPI

cio_MPI.h の 19 行で定義されています。

6.12.2 コンストラクタとデストラクタ

6.12.2.1 cio_MPI::cio_MPI ( )

コンストラクタ

cio_MPI.C の 21 行で定義されています。

参照先 NumberOfGroup, と NumberOfRank.

```
22 {
23   NumberOfRank=0;
24   NumberOfGroup=1;
25 }
```

6.12.2.2 cio_MPI::cio_MPI (const int *_NumberOfRank,* int *_NumberOfGroup* = 0)

コンストラクタ

引数

in	*_NumberOfRank*	プロセス数
in	*_NumberOf-Group*	グループ数

cio_MPI.C の 29 行で定義されています。

参照先 NumberOfGroup, と NumberOfRank.

```
30 {
31   NumberOfRank=_NumberOfRank;
32   NumberOfGroup=_NumberOfGroup;
33 }
```

6.12.2.3 cio_MPI::~cio_MPI ( )

デストラクタ

cio_MPI.C の 37 行で定義されています。

```
38 {
39
40 }
```

6.12.3 関数

6.12.3.1 CIO::E_CIO_ERRORCODE cio_MPI::Read (cio_TextParser *tpCntl,* const cio_Domain *domain*)

read MPI(proc.dfi)

引数

in	*tpCntl*	cio_TextParser クラス
in	*domain*	Domain

戻り値

error code

cio_MPI.C の 45 行で定義されています。

参照先 CIO::E_CIO_SUCCESS, cio_TextParser::GetValue(), cio_Domain::GlobalDivision, NumberOfGroup, と NumberOfRank.

参照元 cio_DFI::ReadInit().

```
47 {
48
49   std::string str;
50   std::string label;
51   int ct;
52
53   //NumberOfRank
54   label = "/MPI/NumberOfRank";
55   if ( !(tpCntl.GetValue(label, &ct )) ) {
56     ct = domain.GlobalDivision[0]*domain.GlobalDivision[1]*domain.GlobalDivision[2];
57   }
58   else {
59     NumberOfRank = ct;
60   }
61
```

```
62   //NumberOfGroup
63   label = "/MPI/NumberOfGroup";
64   if ( !(tpCntl.GetValue(label, &ct )) ) {
65     ct = 1;
66   }
67   else {
68     NumberOfGroup = ct;
69   }
70
71   return CIO::E_CIO_SUCCESS;
72
73 }
```

6.12.3.2 CIO::E_CIO_ERRORCODE cio_MPI::Write (FILE * *fp*, const unsigned *tab*)

DFI ファイル:MPI を出力する

引数

in	*fp*	ファイルポインタ
in	*tab*	インデント

戻り値

error code

cio_MPI.C の 78 行で定義されています。

参照先 _CIO_WRITE_TAB, CIO::E_CIO_SUCCESS, と NumberOfRank.

参照元 cio_DFI::WriteProcDfiFile().

```
79 {
80
81   fprintf(fp, "MPI {\n");
82   fprintf(fp, "\n");
83
84   _CIO_WRITE_TAB(fp, tab+1);
85   fprintf(fp, "NumberOfRank    = %d\n", NumberOfRank);
86
87   _CIO_WRITE_TAB(fp, tab+1);
88   fprintf(fp, "NumberOfGroup   = %d\n", 1);
89
90   fprintf(fp, "\n");
91   fprintf(fp, "}\n");
92   fprintf(fp, "\n");
93
94   return CIO::E_CIO_SUCCESS;
95
96 }
```

6.12.4 変数

6.12.4.1 int cio_MPI::NumberOfGroup

グループ数

cio_MPI.h の 24 行で定義されています。

参照元 cio_MPI(), Read(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

6.12.4.2 int cio_MPI::NumberOfRank

プロセス数

cio_MPI.h の 23 行で定義されています。

参照元 cio_MPI(), Read(), Write(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_avs_cord(), cio_DFI_-AVS::write_avs_header(), cio_DFI_PLOT3D::write_GridData(), cio_DFI::WriteData(), cio_DFI::WriteInit(), と cio_-DFI::WriteProcDfiFile().

このクラスの説明は次のファイルから生成されました:

- cio_MPI.h
- cio_MPI.C

6.13 クラス cio_Process

`#include <cio_Process.h>`

Public 型

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

Public メソッド

- cio_Process ()
- ~cio_Process ()
- CIO::E_CIO_ERRORCODE Read (cio_TextParser tpCntl)

  read Rank(proc.dfi)
- CIO::E_CIO_ERRORCODE CheckReadRank (cio_Domain dfi_domain, const int head[3], const int tail[3], CIO::E_CIO_READTYPE readflag, vector< int > &readRankList)

  読込みランクリストの作成
- CIO::E_CIO_ERRORCODE CreateRankList (cio_Domain dfi_domain, map< int, int > &mapHeadX, map< int, int > &mapHeadY, map< int, int > &mapHeadZ)

  DFI **の** *Process* **に** *HeadIndex,TailIndex* **指定が無い場合**
- CIO::E_CIO_ERRORCODE CreateRankList (int div[3], int gvox[3], map< int, int > &mapHeadX, map< int, int > &mapHeadY, map< int, int > &mapHeadZ)

  DFI **の** *Process* **に** *HeadIndex,TailIndex* **指定が無い場合 渡された、** *subDomain* **をもとに** *CPM* **同様の分割方法で** *RankList* **を生成する**
- CIO::E_CIO_ERRORCODE CreateSubDomainInfo (cio_Domain dfi_domain, vector< cio_ActiveSubDomain > &subDomainInfo)

  ActiveSubDomain **情報を作成**
- int * CreateRankMap (int div[3], std::vector< cio_ActiveSubDomain > &subDomainInfo)

  subdomain **情報からランクマップを生成（非活性を含む）**
- int * CreateRankMap (int ndiv[3], headT &mapHeadX, headT &mapHeadY, headT &mapHeadZ)

  生成済の *RankList* **からランクマップを生成**
- void CreateHeadMap (std::set< int >head, headT &map)

  head map **の生成**
- void CreateHeadMap (int *head, int ndiv, headT &map)

  head map **の生成**
- CIO::E_CIO_ERRORCODE CheckStartEnd (cio_Domain dfi_domain, const int head[3], const int tail[3], CIO::E_CIO_READTYPE readflag, headT mapHeadX, headT mapHeadY, headT mapHeadZ, vector< int > &readRankList)

  読込みランクファイルリストの作成
- CIO::E_CIO_ERRORCODE Write (FILE *fp, const unsigned tab)

  DFI **ファイル**:*Process* **を出力する**

Static Public メソッド

- static int isMatchEndianSbdmMagick (int ident)

  ActiveSubdomain **ファイルのエンディアンをチェック**
- static CIO::E_CIO_ERRORCODE ReadActiveSubdomainFile (std::string subDomainFile, std::vector< cio_ActiveSubDomain > &subDomainInfo, int div[3])

  ActiveSubdomain **ファイルの読み込み** *(static* **関数***)*

Public 変数

- vector< cio_Rank > RankList
- int * m_rankMap

6.13.1 説明

proc.dfi ファイルのProcess

cio_Process.h の 59 行で定義されています。

6.13.2 型定義

6.13.2.1 typedef std::map<int,int> cio_Process::headT

cio_Process.h の 63 行で定義されています。

6.13.3 コンストラクタとデストラクタ

6.13.3.1 cio_Process::cio_Process ( )

コンストラクタ

cio_Process.C の 145 行で定義されています。

参照先 m_rankMap.

```
146 {
147
148   m_rankMap=NULL;
149
150 }
```

6.13.3.2 cio_Process::~cio_Process ( )

デストラクタ

cio_Process.C の 154 行で定義されています。

参照先 m_rankMap.

```
155 {
156   if( m_rankMap ) delete m_rankMap;
157 }
```

6.13.4 関数

6.13.4.1 CIO::E_CIO_ERRORCODE cio_Process::CheckReadRank (cio_Domain *dfi_domain*, const int *head[3]*, const int *tail[3]*, CIO::E_CIO_READTYPE *readflag*, vector< int > & *readRankList*)

読込みランクリストの作成

RankList があるかないか判定しないときは新規にRankList を生成し それをもとにランクマップの生成、読込みランクリスト readRankList を生成する

引数

in	*dfi_domain*	DFI の domain 情報
in	*head*	ソルバーのHeadIndex
in	*tail*	ソルバーのTailIndex
in	*readflag*	読込み方法
out	*readRankList*	読込みランクリスト

戻り値

error code

cio_Process.C の 205 行で定義されています。

参照先 CheckStartEnd(), CreateHeadMap(), CreateRankList(), CreateRankMap(), CIO::E_CIO_SUCCESS, cio_-Domain::GlobalDivision, m_rankMap, と RankList.

参照元 cio_DFI::CheckReadRank(), と cio_DFI::ReadData().

```
210 {
211
212   headT mapHeadX,mapHeadY,mapHeadZ;
213
214   //DFI に Process/Rank[@] がない処理
215   if( RankList.empty() ) {
216     CIO::E_CIO_ERRORCODE ret = CreateRankList(dfi_domain, mapHeadX, mapHeadY, mapHeadZ);
217     if( ret != CIO::E_CIO_SUCCESS ) return ret;
218   }
219
220   //rankMap が未定義（DFI に Process/Rank[@] がある場合）
221   if( m_rankMap == NULL ) {
222     m_rankMap = CreateRankMap(dfi_domain.GlobalDivision, mapHeadX, mapHeadY, mapHeadZ);
223   }
224
225   //mapHeadX,mapHeadY,mapHeadZ が未定義
226   if( mapHeadX.empty() || mapHeadY.empty() || mapHeadZ.empty() ) {
227     std::set<int>headx,heady,headz;
228     for(int i=0; i<RankList.size(); i++ ) {
229       headx.insert(RankList[i].HeadIndex[0]);
230       heady.insert(RankList[i].HeadIndex[1]);
231       headz.insert(RankList[i].HeadIndex[2]);
232     }
233     CreateHeadMap(headx,mapHeadX);
234     CreateHeadMap(heady,mapHeadY);
235     CreateHeadMap(headz,mapHeadZ);
236   }
237
238   return CheckStartEnd(dfi_domain, head, tail, readflag, mapHeadX, mapHeadY, mapHeadZ, ReadRankList);
239
240 }
```

6.13.4.2 CIO::E_CIO_ERRORCODE cio_Process::CheckStartEnd (cio_Domain *dfi_domain*, const int *head[3]*, const int *tail[3]*, CIO::E_CIO_READTYPE *readflag*, headT *mapHeadX*, headT *mapHeadY*, headT *mapHeadZ*, vector< int > & *readRankList*)

読込みランクファイルリストの作成

引数

in	*dfi_domain*	DFI のDomain 情報
in	*head*	計算領域の開始位置
in	*tail*	計算領域の終了位置
in	*readflag*	粗密データ判定フラグ
in	*mapHeadX*	headX をキーにした位置情報マップ
in	*mapHeadY*	headY をキーにした位置情報マップ
in	*mapHeadZ*	headZ をキーにした位置情報マップ
out	*readRankList*	読込みに必要なランク番号リスト

cio_Process.C の 610 行で定義されています。

参照先 _CIO_IDX_IJK, CIO::E_CIO_DIFFDIV_SAMERES, CIO::E_CIO_SAMEDIV_SAMERES, CIO::E_CIO_SUCCESS, cio_Domain::GlobalDivision, と m_rankMap.

参照元 CheckReadRank().

```
618 {
619
620   int StartEnd[6];
621
622   int ndiv = dfi_domain.GlobalDivision[0]*
623              dfi_domain.GlobalDivision[1]*
624              dfi_domain.GlobalDivision[2];
625   int head2[3],tail2[3];
626   if( readflag == CIO::E_CIO_SAMEDIV_SAMERES || readflag ==
      CIO::E_CIO_DIFFDIV_SAMERES ) {
627     for(int i=0; i<3; i++) {
628       head2[i]=head[i];
629       tail2[i]=tail[i];
630     }
631   } else {
632     for(int i=0; i<3; i++) {
633       if( head[i] < 0 ) head2[i]=head[i]/2;
634       else              head2[i]=(head[i]+1)/2;
635       if( tail[i] < 0 ) tail2[i]=tail[i]/2;
636       else              tail2[i]=(tail[i]+1)/2;
637     }
638   }
639
640 //x 方向の絞り込み
641   for( headT::iterator it=mapHeadX.begin();it!=mapHeadX.end();it++ )
642   {
643     if( head2[0] >= (*it).first ) StartEnd[0] = (*it).second;
644     if( tail2[0] >= (*it).first ) StartEnd[3] = (*it).second;
645     else break;
646   }
647
648 //y 方向の絞り込み
649   for( headT::iterator it=mapHeadY.begin();it!=mapHeadY.end();it++ )
650   {
651     if( head2[1] >= (*it).first ) StartEnd[1] = (*it).second;
652     if( tail2[1] >= (*it).first ) StartEnd[4] = (*it).second;
653     else break;
654   }
655
656 //z 方向の絞り込み
657   for( headT::iterator it=mapHeadZ.begin();it!=mapHeadZ.end();it++ )
658   {
659     if( head2[2] >= (*it).first ) StartEnd[2] = (*it).second;
660     if( tail2[2] >= (*it).first ) StartEnd[5] = (*it).second;
661     else break;
662   }
663
664   readRankList.clear();
665
666   for(int k=StartEnd[2]; k<=StartEnd[5]; k++) {
667   for(int j=StartEnd[1]; j<=StartEnd[4]; j++) {
668   for(int i=StartEnd[0]; i<=StartEnd[3]; i++) {
669     int rank = m_rankMap[_CIO_IDX_IJK(i,j,k,dfi_domain.GlobalDivision[0],
670                                       dfi_domain.GlobalDivision[1],
671                                       dfi_domain.GlobalDivision[2],0)];
672     if( rank<0 ) continue;
673
674     readRankList.push_back(rank);
675
676   }}}
677
678   return CIO::E_CIO_SUCCESS;
679 }
```

6.13.4.3 void cio_Process::CreateHeadMap (std::set< int > *head,* headT & *map*)

head map の生成

引数

in	*head*	head インデックス
out	*map*	head map

cio_Process.C の 529 行で定義されています。

参照元 CheckReadRank(), CreateRankList(), と CreateRankMap().

```
531 {
532
533   map.clear();
534
535   int cnt=0;
536   for(std::set<int>::iterator it=head.begin();it!=head.end();it++)
537   {
538     int key=*it;
539     map.insert(headT::value_type(key,cnt));
540     cnt++;
541   }
542 }
```

6.13.4.4 void cio_Process::CreateHeadMap (int * *head,* int *ndiv,* headT & *map*)

head map の生成

引数

in	*head*	head インデックス
in	*ndiv*	分割数
out	*map*	head map

cio_Process.C の 546 行で定義されています。

```
549 {
550
551   map.clear();
552
553   for(int i=0; i<ndiv; i++)
554   {
555     map.insert(headT::value_type(head[i],i));
556   }
557 }
```

6.13.4.5 CIO::E_CIO_ERRORCODE cio_Process::CreateRankList (cio_Domain *dfi_domain,* map< int, int > & *mapHeadX,* map< int, int > & *mapHeadY,* map< int, int > & *mapHeadZ*)

DFI のProcess にHeadIndex,TailIndex 指定が無い場合

ActiveSubDomain があれば、読込み、なければ全て有効で subDomain を生成し、CreateRankList に渡す CPM と同じ分割で head&tail 情報を作成してRankList を作成する

引数

in	*dfi_domain*	DFI の domain 情報
out	*mapHeadX*	headX をキーにした位置情報マップ
out	*mapHeadY*	headX をキーにした位置情報マップ

out	*mapHeadZ*	headX をキーにした位置情報マップ

戻り値

error code

cio_Process.C の 245 行で定義されています。

参照先 CreateRankMap(), CreateSubDomainInfo(), CIO::E_CIO_SUCCESS, cio_Domain::GlobalDivision, cio_Domain::GlobalVoxel, と m_rankMap.

参照元 CheckReadRank().

```
249 {
250
251   vector<cio_ActiveSubDomain> subDomainInfo;
252
253   CIO::E_CIO_ERRORCODE ret;
254
255   ret = CreateSubDomainInfo(dfi_domain,subDomainInfo);
256   if( ret != CIO::E_CIO_SUCCESS ) return ret;
257
258   m_rankMap = CreateRankMap(dfi_domain.GlobalDivision,subDomainInfo);
259
260   ret = CreateRankList(dfi_domain.GlobalDivision, dfi_domain.GlobalVoxel,
261                        mapHeadX, mapHeadY, mapHeadZ);
262
263   return ret;
264
265 }
```

6.13.4.6 CIO::E_CIO_ERRORCODE cio_Process::CreateRankList (int *div[3]*, int *gvox[3]*, map< int, int > & *mapHeadX*, map< int, int > & *mapHeadY*, map< int, int > & *mapHeadZ*)

DFI のProcess にHeadIndex,TailIndex 指定が無い場合 渡された、subDomain をもとにCPM 同様の分割方法で RankList を生成する

引数

in	*div*	分割数
in	*gvox*	ボクセルサイズ
out	*mapHeadX*	headX をキーにした位置情報マップ
out	*mapHeadY*	headX をキーにした位置情報マップ
out	*mapHeadZ*	headX をキーにした位置情報マップ

戻り値

error code

cio_Process.C の 295 行で定義されています。

参照先 _CIO_IDX_IJK, CreateHeadMap(), CIO::E_CIO_ERROR, CIO::E_CIO_SUCCESS, cio_Rank::HeadIndex, m_rankMap, cio_Rank::RankID, RankList, cio_Rank::TailIndex, と cio_Rank::VoxelSize.

```
300 {
301   if( !m_rankMap ) return CIO::E_CIO_ERROR;
302   int ndiv = div[0]*div[1]*div[2];
303
304   cio_Rank rank;
305
306   //ローカルの VOXEL 数
307   int *nvX = new int[div[0]];
308   int *nvY = new int[div[1]];
309   int *nvZ = new int[div[2]];
310   int *nv[3] = {nvX,nvY,nvZ};
311   for( int n=0;n<3;n++ )
312   {
313     int *nvd = nv[n];
```

```
314     //基準のボクセル数
315     int nbase = gvox[n] / div[n];
316
317     //余り
318     int amari = gvox[n] % div[n];
319
320     //ボクセル数をセット
321     for( int i=0;i<div[n];i++ )
322     {
323       nvd[i] = nbase;
324       if( i<amari ) nvd[i]++;
325     }
326   }
327
328   //head
329   int *headX = new int[div[0]];
330   int *headY = new int[div[1]];
331   int *headZ = new int[div[2]];
332   int *head[3] = {headX,headY,headZ};
333   for( int n=0;n<3;n++ )
334   {
335     int *nvd = nv[n];
336     int *hd = head[n];
337     hd[0] = 1;
338
339     for( int i=1;i<div[n];i++ )
340     {
341       hd[i] = hd[i-1]+nvd[i-1];
342     }
343   }
344
345   CreateHeadMap(headX,div[0],mapHeadX);
346   //CreateHeadMap(headY,div[0],mapHeadY);
347   CreateHeadMap(headY,div[1],mapHeadY);
348   //CreateHeadMap(headZ,div[0],mapHeadZ);
349   CreateHeadMap(headZ,div[2],mapHeadZ);
350
351   for( int k=0;k<div[2];k++ ){
352   for( int j=0;j<div[1];j++ ){
353   for( int i=0;i<div[0];i++ ){
354     int rankNo = m_rankMap[_CIO_IDX_IJK(i,j,k,div[0],div[1],div[2],0)];
355     if( rankNo < 0 ) continue;
356     rank.RankID = rankNo;
357     rank.VoxelSize[0]=(headX[i]+nvX[i]-1)-headX[i]+1;
358     rank.VoxelSize[1]=(headY[j]+nvY[j]-1)-headY[j]+1;
359     rank.VoxelSize[2]=(headZ[k]+nvZ[k]-1)-headZ[k]+1;
360     rank.HeadIndex[0]=headX[i];
361     rank.HeadIndex[1]=headY[j];
362     rank.HeadIndex[2]=headZ[k];
363     rank.TailIndex[0]=headX[i]+nvX[i]-1;
364     rank.TailIndex[1]=headY[j]+nvY[j]-1;
365     rank.TailIndex[2]=headZ[k]+nvZ[k]-1;
366     RankList.push_back(rank);
367   }}}
368
369   delete [] nvX;
370   delete [] nvY;
371   delete [] nvZ;
372   delete [] headX;
373   delete [] headY;
374   delete [] headZ;
375
376   return CIO::E_CIO_SUCCESS;
377 }
```

6.13.4.7 int ∗ cio_Process::CreateRankMap (int *div[3]*, std::vector< cio_ActiveSubDomain > & *subDomainInfo*)

subdomain 情報からランクマップを生成（非活性を含む）

引数

in	*div*	領域分割数
in	*subDomainInfo*	活性ドメイン情報

戻り値

ランクマップ
NULL

cio_Process.C の 483 行で定義されています。

参照先 _CIO_IDX_IJK, と cio_ActiveSubDomain::GetPos().

参照元 CheckReadRank(), と CreateRankList().

```
486 {
487
488   size_t ndiv = size_t(div[0]) * size_t(div[1]) * size_t(div[2]);
489   int *rankMap = new int[ndiv];
490   if( !rankMap ) return NULL;
491
492   for( size_t i=0;i<ndiv;i++ ) rankMap[i] = -1;
493
494   // 活性サブドメイン情報配置位置に 0 をセット
495   for( int i=0; i<subDomainInfo.size(); i++ )
496   {
497     //サブドメイン情報
498     const cio_ActiveSubDomain dom = subDomainInfo[i];
499
500     //位置を取得
501     const int *pos = dom.GetPos();
502     if( !pos )
503     {
504       delete [] rankMap;
505       return NULL;
506     }
507
508     //0 をセット
509     rankMap[_CIO_IDX_IJK(pos[0],pos[1],pos[2],div[0],div[1],div[2],0)] = 0;
510   }
511
512  //i->j->k の優先順で活性サブドメインにランク番号をセット
513   int rankCount = 0;
514   for( int k=0;k<div[2];k++ ){
515   for( int j=0;j<div[1];j++ ){
516   for( int i=0;i<div[0];i++ ){
517     if( rankMap[_CIO_IDX_IJK(i,j,k,div[0],div[1],div[2],0)] == 0 )
518     {
519       rankMap[_CIO_IDX_IJK(i,j,k,div[0],div[1],div[2],0)] = rankCount;
520       rankCount++;
521     }
522   }}}
523
524   return rankMap;
525 }
```

6.13.4.8 int * cio_Process::CreateRankMap (int *ndiv[3],* headT & *mapHeadX,* headT & *mapHeadY,* headT & *mapHeadZ*)

生成済のRankList からランクマップを生成

引数

in	*ndiv*	領域分割数
in	*mapHeadX*	headX をキーにした位置情報マップ
in	*mapHeadY*	headY をキーにした位置情報マップ
in	*mapHeadZ*	headZ をキーにした位置情報マップ

戻り値

ランクマップ
NULL

cio_Process.C の 562 行で定義されています。

参照先 _CIO_IDX_IJK, CreateHeadMap(), と RankList.

```
566 {
567
568   int i,j,k;
569
570   std::set<int>headx,heady,headz;
571   for(int i=0; i<RankList.size(); i++ ) {
572     headx.insert(RankList[i].HeadIndex[0]);
```

```
573     heady.insert(RankList[i].HeadIndex[1]);
574     headz.insert(RankList[i].HeadIndex[2]);
575   }
576   CreateHeadMap(headx,mapHeadX);
577   CreateHeadMap(heady,mapHeadY);
578   CreateHeadMap(headz,mapHeadZ);
579
580   size_t ndiv = div[0]*div[1]*div[2];
581
582   int *rankMap = new int[ndiv];
583   for(int i=0; i<ndiv; i++) rankMap[i]=-1;
584
585   headT::iterator it;
586
587   for(int n=0; n<RankList.size(); n++)
588   {
589     it=mapHeadX.find(RankList[n].HeadIndex[0]);
590     i=it->second;
591
592     it=mapHeadY.find(RankList[n].HeadIndex[1]);
593     j=it->second;
594
595     it=mapHeadZ.find(RankList[n].HeadIndex[2]);
596     k=it->second;
597
598     int rnkPos=_CIO_IDX_IJK(i,j,k,div[0],div[1],div[2],0);
599
600     rankMap[_CIO_IDX_IJK(i,j,k,div[0],div[1],div[2],0)] = n;
601   }
602
603   return rankMap;
604
605 }
```

6.13.4.9 CIO::E_CIO_ERRORCODE cio_Process::CreateSubDomainInfo (cio_Domain *dfi_domain,* vector< cio_ActiveSubDomain > & *subDomainInfo*)

ActiveSubDomain 情報を作成

引数

in	*dfi_domain*	DFI の domain 情報
out	*subDomainInfo*	活性ドメイン情報

cio_Process.C の 270 行で定義されています。

参照先 cio_Domain::ActiveSubdomainFile, CIO::E_CIO_SUCCESS, cio_Domain::GlobalDivision, と ReadActiveSubdomainFile().

参照元 CreateRankList().

```
272 {
273   if( !domain.ActiveSubdomainFile.empty() ) {
274     int divSudomain[3] = {0,0,0};
275     CIO::E_CIO_ERRORCODE ret = ReadActiveSubdomainFile( domain.ActiveSubdomainFile,
276                                                         subDomainInfo, divSudomain);
277     if( ret != CIO::E_CIO_SUCCESS ) return ret;
278   } else {
279     //活性サブドメイン情報
280     for( int k=0;k<domain.GlobalDivision[2];k++ ){
281     for( int j=0;j<domain.GlobalDivision[1];j++ ){
282     for( int i=0;i<domain.GlobalDivision[0];i++ ){
283       int pos[3] = {i,j,k};
284       cio_ActiveSubDomain dom( pos );
285       subDomainInfo.push_back(dom);
286     }}}
287
288   }
289
290   return CIO::E_CIO_SUCCESS;
291 }
```

6.13.4.10 int cio_Process::isMatchEndianSbdmMagick (int *ident*) [static]

ActiveSubdomain ファイルのエンディアンをチェック

引数

in	*ident*	ActiveSubdomain ファイルのIdentifier

戻り値

1 一致
0 不一致
-1 フォーマットが異なる

cio_Process.C の 458 行で定義されています。

参照元 ReadActiveSubdomainFile().

```
459 {
460   char magick_c[] = "SBDM";
461   int  magick_i=0;
462
463   //cheak match
464   magick_i = (magick_c[3]<<24) + (magick_c[2]<<16) + (magick_c[1]<<8) + magick_c[0];
465   if( magick_i == ident )
466   {
467     return 1;
468   }
469
470   //chack unmatch
471   magick_i = (magick_c[0]<<24) + (magick_c[1]<<16) + (magick_c[2]<<8) + magick_c[3];
472   if( magick_i == ident )
473   {
474     return 0;
475   }
476
477   //unknown format
478   return -1;
479 }
```

6.13.4.11 CIO::E_CIO_ERRORCODE cio_Process::Read (cio_TextParser *tpCntl*)

read Rank(proc.dfi)

引数

in	*tpCntl*	cio_TextParser クラス

戻り値

error code

< リターンコード

Rank の読込み

cio_Process.C の 162 行で定義されています。

参照先 cio_TextParser::chkNode(), cio_TextParser::countLabels(), CIO::E_CIO_ERROR_READ_DFI_NO_RANK, CIO::E_CIO_SUCCESS, cio_TextParser::GetNodeStr(), RankList, と cio_Rank::Read().

参照元 cio_DFI::ReadInit().

```
163 {
164
165   std::string str;
166   std::string label_base,label_leaf;
167   int nnode=0;
168   CIO::E_CIO_ERRORCODE iret;
169
170   cio_Rank rank;
171
172   //Process
173   nnode=0;
```

```
174   label_base = "/Process";
175   if ( tpCntl.chkNode(label_base) )  //node があれば
176   {
177     nnode = tpCntl.countLabels(label_base);
178   }
179
180   for (int i=0; i<nnode; i++) {
181
182     if(!tpCntl.GetNodeStr(label_base,i+1,&str))
183     {
184       printf("\tCIO Parsing error : No Elem name\n");
185       return CIO::E_CIO_ERROR_READ_DFI_NO_RANK;
186     }
187     if( strcasecmp(str.substr(0,4).c_str(), "Rank") ) continue;
188     label_leaf=label_base+"/"+str;
189
191     iret = rank.Read(tpCntl, label_leaf);
192     if( iret == CIO::E_CIO_SUCCESS ) {
193       RankList.push_back(rank);
194     } else  return iret;
195
196   }
197
198   return CIO::E_CIO_SUCCESS;
199
200 }
```

6.13.4.12 CIO::E_CIO_ERRORCODE cio_Process::ReadActiveSubdomainFile (std::string *subDomainFile*, std::vector< cio_ActiveSubDomain > & *subDomainInfo*, int *div[3]*) [static]

ActiveSubdomain ファイルの読み込み (static 関数)

引数

in	*subDomainFile*	ActiveSubdomain ファイル名
out	*subDomainInfo*	活性ドメイン情報
out	*div*	ActiveSubdiomain ファイル中の領域分割数

戻り値

終了コード (CIO_SUCCESS=正常終了)

cio_Process.C の 382 行で定義されています。

参照先 BSWAPVEC, CIO::E_CIO_ERROR_OPEN_SBDM, CIO::E_CIO_ERROR_READ_SBDM_CONTENTS, CIO::E_CIO_ERROR_READ_SBDM_DIV, CIO::E_CIO_ERROR_READ_SBDM_FORMAT, CIO::E_CIO_ERROR_READ_SBDM_HEADER, CIO::E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO, CIO::E_CIO_SUCCESS, と isMatchEndianSbdmMagick().

参照元 CreateSubDomainInfo().

```
386 {
387   if( subDomainFile.empty() ) return CIO::E_CIO_ERROR_OPEN_SBDM;
388
389   // ファイルオープン
390   FILE*fp = fopen( subDomainFile.c_str(), "rb" );
391   if( !fp ) return CIO::E_CIO_ERROR_OPEN_SBDM;
392
393   //エンディアン識別子
394   int ident;
395     if( fread( &ident, sizeof(int), 1, fp ) != 1 )
396   {
397     fclose(fp);
398     return CIO::E_CIO_ERROR_READ_SBDM_HEADER;
399   }
400
401  //エンディアンチェック
402   if( isMatchEndianSbdmMagick( ident ) < 0 )
403   {
404     fclose(fp);
405     return CIO::E_CIO_ERROR_READ_SBDM_FORMAT;
406   }
407
408   // 領域分割数
```

```
409   if( fread( div, sizeof(int), 3, fp ) != 3 )
410   {
411     fclose(fp);
412     return CIO::E_CIO_ERROR_READ_SBDM_DIV;
413   }
414
415   if( isMatchEndianSbdmMagick( ident ) == 0 ) BSWAPVEC(div,3);
416
417   // contents
418   size_t nc = size_t(div[0]) * size_t(div[1]) * size_t(div[2]);
419   unsigned char *contents = new unsigned char[nc];
420   if( fread( contents, sizeof(unsigned char), nc, fp ) != nc )
421   {
422     delete [] contents;
423     fclose(fp);
424     return CIO::E_CIO_ERROR_READ_SBDM_CONTENTS;
425   }
426
427   // ファイルクローズ
428   fclose(fp);
429
430   size_t ptr = 0;
431   // 活性ドメイン情報の生成
432   for( int k=0;k<div[2];k++ ){
433   for( int j=0;j<div[1];j++ ){
434   for( int i=0;i<div[0];i++ ){
435     if( contents[ptr] == 0x01 )
436     {
437       int pos[3] = {i,j,k};
438       cio_ActiveSubDomain dom( pos );
439       subDomainInfo.push_back(dom);
440     }
441     ptr++;
442   }}}
443
444   // contents の delete
445   delete [] contents;
446
447   // 活性ドメインの数をチェック
448   if( subDomainInfo.size() == 0 )
449   {
450     return CIO::E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO;
451   }
452
453   return CIO::E_CIO_SUCCESS;
454 }
```

6.13.4.13 CIO::E_CIO_ERRORCODE cio_Process::Write (FILE * *fp*, const unsigned *tab*)

DFI ファイル:Process を出力する

引数

in	*fp*	ファイルポインタ
in	*tab*	インデント

戻り値

true:出力成功 false:出力失敗

cio_Process.C の 684 行で定義されています。

参照先 _CIO_WRITE_TAB, CIO::E_CIO_ERROR, CIO::E_CIO_SUCCESS, RankList, と cio_Rank::Write().

参照元 cio_DFI::WriteProcDfiFile().

```
686 {
687
688   fprintf(fp, "Process {\n");
689   fprintf(fp, "\n");
690
691   cio_Rank rank;
692
693   for(int i=0; i<RankList.size(); i++) {
694     _CIO_WRITE_TAB(fp, tab+1);
695     fprintf(fp, "Rank[@] {\n");
696     fprintf(fp, "\n");
```

```
697
698     rank = RankList[i];
699
700     //Rank 要素の出力
701     if( rank.Write(fp,tab+2) != CIO::E_CIO_SUCCESS ) return CIO::E_CIO_ERROR;
702
703     fprintf(fp, "\n");
704     _CIO_WRITE_TAB(fp, tab+1);
705     fprintf(fp, "}\n");
706   }
707
708   fprintf(fp, "\n");
709   fprintf(fp, "}\n");
710
711   return CIO::E_CIO_SUCCESS;
712 }
```

6.13.5 変数

6.13.5.1 int* cio_Process::m_rankMap

cio_Process.h の 67 行で定義されています。

参照元 CheckReadRank(), CheckStartEnd(), cio_Process(), CreateRankList(), と ~cio_Process().

6.13.5.2 vector<cio_Rank> cio_Process::RankList

cio_Process.h の 65 行で定義されています。

参照元 CheckReadRank(), cio_DFI::cio_Create_dfiProcessInfo(), CreateRankList(), CreateRankMap(), Read(), cio_DFI::ReadData(), Write(), cio_DFI_BOV::write_ascii_header(), cio_DFI_AVS::write_ascii_header(), cio_DFI_AVS::write_avs_header(), cio_DFI_BOV::write_DataRecord(), cio_DFI_SPH::write_DataRecord(), cio_DFI_PLOT3D::write_GridData(), cio_DFI_VTK::write_HeaderRecord(), cio_DFI_SPH::write_HeaderRecord(), cio_DFI::WriteData(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

このクラスの説明は次のファイルから生成されました:

- cio_Process.h
- cio_Process.C

6.14 クラス cio_Rank

`#include <cio_Process.h>`

Public メソッド

- cio_Rank ()
- ~cio_Rank ()
- CIO::E_CIO_ERRORCODE Read (cio_TextParser tpCntl, std::string label_leaf)

  read Rank(proc.dfi)
- CIO::E_CIO_ERRORCODE Write (FILE *fp, const unsigned tab)

  DFI **ファイル***:Rank* **出力する**

Public 変数

- int RankID

  ランク番号
- std::string HostName

ホスト名

- int VoxelSize [3]

  ボクセルサイズ

- int HeadIndex [3]

  始点インデックス

- int TailIndex [3]

  終点インデックス

6.14.1 説明

proc.dfi ファイルの Rank

cio_Process.h の 19 行で定義されています。

6.14.2 コンストラクタとデストラクタ

6.14.2.1 cio_Rank::cio_Rank ( )

コンストラクタ

cio_Process.C の 21 行で定義されています。

参照先 HeadIndex, HostName, RankID, TailIndex, と VoxelSize.

```
22 {
23
24  RankID = 0;
25  HostName = "";
26  for(int i=0; i<3; i++) {
27    VoxelSize[i]=0;
28    HeadIndex[i]=0;
29    TailIndex[i]=0;
30  }
31
32 }
```

6.14.2.2 cio_Rank::~cio_Rank ( )

デストラクタ

cio_Process.C の 37 行で定義されています。

```
38 {
39
40 }
```

6.14.3 関数

6.14.3.1 CIO::E_CIO_ERRORCODE cio_Rank::Read (cio_TextParser *tpCntl*, std::string *label_leaf*)

read Rank(proc.dfi)

引数

in	*tpCntl*	cio_TextParser クラス

in	*label_leaf*	ベースとなる名前（"/Process/Rank")

戻り値

error code

cio_Process.C の 45 行で定義されています。

参照先 CIO::E_CIO_ERROR_READ_DFI_HEADINDEX, CIO::E_CIO_ERROR_READ_DFI_HOSTNAME, CIO::E_CIO_ERROR_READ_DFI_ID, CIO::E_CIO_ERROR_READ_DFI_TAILINDEX, CIO::E_CIO_ERROR_READ_DFI_VOXELSIZE, CIO::E_CIO_SUCCESS, cio_TextParser::GetValue(), cio_TextParser::GetVector(), HeadIndex, HostName, RankID, TailIndex, と VoxelSize.

参照元 cio_Process::Read().

```
47 {
48
49   std::string str;
50   std::string label;
51   int ct;
52   int iv[3];
53
54   //ID
55   label = label_leaf + "/ID";
56   if ( !(tpCntl.GetValue(label, &ct )) ) {
57     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
58     return CIO::E_CIO_ERROR_READ_DFI_ID;
59   }
60   else {
61     RankID= ct;
62   }
63
64   //HostName
65   label = label_leaf + "/HostName";
66   if ( !(tpCntl.GetValue(label, &str )) ) {
67     printf("\tCIO Parsing error : fail to get '%s'\n", label.c_str());
68     return CIO::E_CIO_ERROR_READ_DFI_HOSTNAME;
69   }
70   HostName= str;
71
72   //VoxelSize
73   label = label_leaf + "/VoxelSize";
74   for (int n=0; n<3; n++) iv[n]=0.0;
75   if ( !(tpCntl.GetVector(label, iv, 3 )) )
76   {
77     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
78     return CIO::E_CIO_ERROR_READ_DFI_VOXELSIZE;
79   }
80   VoxelSize[0]=iv[0];
81   VoxelSize[1]=iv[1];
82   VoxelSize[2]=iv[2];
83
84   //HeadIndex
85   label = label_leaf + "/HeadIndex";
86   for (int n=0; n<3; n++) iv[n]=0.0;
87   if ( !(tpCntl.GetVector(label, iv, 3 )) )
88   {
89     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
90     return CIO::E_CIO_ERROR_READ_DFI_HEADINDEX;
91   }
92   HeadIndex[0]=iv[0];
93   HeadIndex[1]=iv[1];
94   HeadIndex[2]=iv[2];
95
96   //TailIndex
97   label = label_leaf + "/TailIndex";
98   for (int n=0; n<3; n++) iv[n]=0.0;
99   if ( !(tpCntl.GetVector(label, iv, 3 )) )
100   {
101     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
102     return CIO::E_CIO_ERROR_READ_DFI_TAILINDEX;
103   }
104   TailIndex[0]=iv[0];
105   TailIndex[1]=iv[1];
106   TailIndex[2]=iv[2];
107
108   return CIO::E_CIO_SUCCESS;
109 }
```

6.14.3.2 CIO::E_CIO_ERRORCODE cio_Rank::Write (FILE * *fp*, const unsigned *tab*)

DFI ファイル:Rank 出力する

引数

in	*fp*	ファイルポインタ
in	*tab*	インデント

戻り値

true:出力成功 false:出力失敗

cio_Process.C の 114 行で定義されています。

参照先 _CIO_WRITE_TAB, CIO::E_CIO_SUCCESS, HeadIndex, HostName, RankID, TailIndex, と VoxelSize.

参照元 cio_Process::Write().

```
116 {
117
118     _CIO_WRITE_TAB(fp, tab);
119     fprintf(fp, "ID        = %d\n", RankID);
120
121     _CIO_WRITE_TAB(fp, tab);
122     fprintf(fp, "HostName  = \"%s\"\n", HostName.c_str());
123
124     _CIO_WRITE_TAB(fp, tab);
125     fprintf(fp, "VoxelSize = (%d, %d, %d)\n", VoxelSize[0],
126                                               VoxelSize[1],
127                                               VoxelSize[2]);
128
129     _CIO_WRITE_TAB(fp, tab);
130     fprintf(fp, "HeadIndex = (%d, %d, %d)\n", HeadIndex[0],
131                                               HeadIndex[1],
132                                               HeadIndex[2]);
133
134     _CIO_WRITE_TAB(fp, tab);
135     fprintf(fp, "TailIndex = (%d, %d, %d)\n", TailIndex[0],
136                                               TailIndex[1],
137                                               TailIndex[2]);
138
139   return CIO::E_CIO_SUCCESS;
140
141 }
```

6.14.4 変数

6.14.4.1 int cio_Rank::HeadIndex[3]

始点インデックス

cio_Process.h の 27 行で定義されています。

参照元 cio_DFI::cio_Create_dfiProcessInfo(), cio_Rank(), cio_Process::CreateRankList(), Read(), と Write().

6.14.4.2 std::string cio_Rank::HostName

ホスト名

cio_Process.h の 25 行で定義されています。

参照元 cio_Rank(), Read(), と Write().

6.14.4.3 int cio_Rank::RankID

ランク番号

cio_Process.h の 24 行で定義されています。

参照元 cio_DFI::cio_Create_dfiProcessInfo(), cio_Rank(), cio_Process::CreateRankList(), Read(), と Write().

6.14.4.4 int cio_Rank::TailIndex[3]

終点インデックス

cio_Process.h の 28 行で定義されています。

参照元 cio_DFI::cio_Create_dfiProcessInfo(), cio_Rank(), cio_Process::CreateRankList(), Read(), と Write().

6.14.4.5 int cio_Rank::VoxelSize[3]

ボクセルサイズ

cio_Process.h の 26 行で定義されています。

参照元 cio_DFI::cio_Create_dfiProcessInfo(), cio_Rank(), cio_Process::CreateRankList(), Read(), と Write().

このクラスの説明は次のファイルから生成されました:

- cio_Process.h
- cio_Process.C

6.15 クラス cio_Slice

```
#include <cio_TimeSlice.h>
```

Public メソッド

- cio_Slice ()
- ~cio_Slice ()
- CIO::E_CIO_ERRORCODE Read (cio_TextParser tpCntl, std::string label_leaf)
  TimeSlice **要素を読込む** *(inde.dfi)*
- CIO::E_CIO_ERRORCODE Write (FILE *fp, const unsigned tab)
  DFI **ファイル***:TimeSlice* **要素を出力する**

Public 変数

- int step
  ステップ番号
- double time
  時刻
- bool avr_mode
  Average **出力フラグ** *true:***出力なし、** *false:***出力**
- int AveragedStep
  平均ステップ
- double AveragedTime
  平均タイム
- double VectorMin
  Vector **のとき、最小値の合成値**
- double VectorMax
  Vector **のとき、最大値の合成値**
- vector< double > Min
  最小値
- vector< double > Max
  最大値

6.15.1 説明

index.dfi ファイルの Slice

cio_TimeSlice.h の 19 行で定義されています。

6.15.2 コンストラクタとデストラクタ

6.15.2.1 cio_Slice::cio_Slice ( )

コンストラクタ

cio_TimeSlice.C の 21 行で定義されています。

参照先 AveragedStep, AveragedTime, Max, Min, step, time, VectorMax, と VectorMin.

```
22 {
23
24   step = 0;
25   time = 0.0;
26   AveragedStep = 0;
27   AveragedTime = 0.0;
28   VectorMin = 0.0;
29   VectorMax = 0.0;
30
31   Min.clear();
32   Max.clear();
33
34 }
```

6.15.2.2 cio_Slice::~cio_Slice ( )

デストラクタ

cio_TimeSlice.C の 39 行で定義されています。

```
40 {
41
42 }
```

6.15.3 関数

6.15.3.1 CIO::E_CIO_ERRORCODE cio_Slice::Read (cio_TextParser *tpCntl,* std::string *label_leaf*)

TimeSlice 要素を読込む (inde.dfi)

引数

in	*tpCntl*	cio_TextParser クラス
in	*label_leaf*	ベースとなる名前（"/TimeSlice/Slice")

戻り値

error code

cio_TimeSlice.C の 47 行で定義されています。

参照先 AveragedStep, AveragedTime, cio_TextParser::chkNode(), cio_TextParser::countLabels(), CIO::E_CIO_ERROR_READ_DFI_MAX, CIO::E_CIO_ERROR_READ_DFI_MIN, CIO::E_CIO_ERROR_READ_DFI_NO_MINMAX, CIO::E_CIO_ERROR_READ_DFI_STEP, CIO::E_CIO_ERROR_READ_DFI_TIME, CIO::E_CIO_SUCCESS, cio_TextParser::GetNodeStr(), cio_TextParser::GetValue(), Max, Min, step, time, VectorMax, と VectorMin.

参照元 cio_TimeSlice::Read().

```
49 {
50
51   std::string str;
52   std::string label,label_leaf_leaf;
53
54   int ct;
55   double dt;
56
57   int ncnt=0;
58
59   //Step
60   label = label_leaf + "/Step";
61   if ( !(tpCntl.GetValue(label, &ct )) ) {
62     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
63     return CIO::E_CIO_ERROR_READ_DFI_STEP;
64   }
65   else {
66     step=ct;
67   }
68
69   ncnt++;
70
71   //Time
72   label = label_leaf + "/Time";
73   if ( !(tpCntl.GetValue(label, &dt )) ) {
74     printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
75     return CIO::E_CIO_ERROR_READ_DFI_TIME;
76   }
77   else {
78     time= dt;
79   }
80
81   ncnt++;
82
83   //AveragedStep
84   label = label_leaf + "/AveragedStep";
85   if ( !(tpCntl.GetValue(label, &ct )) ) {
86     AveragedStep=-1;
87   }
88   else {
89     AveragedStep= ct;
90     ncnt++;
91   }
92
93   //AveragedTime
94   label = label_leaf + "/AveragedTime";
95   if ( !(tpCntl.GetValue(label, &dt )) ) {
96     AveragedTime=0.0;
97   }
98   else {
99     AveragedTime= dt;
100     ncnt++;
101   }
102
103   //VectorMinMax/Min
104   label = label_leaf + "/VectorMinMax/Min";
105   if ( (tpCntl.GetValue(label, &dt )) )
106   {
107     VectorMin=dt;
108     ncnt++;
109   }
110
111   //VectorMinMax/Max
112   label = label_leaf + "/VectorMinMax/Max";
113   if ( (tpCntl.GetValue(label, &dt )) )
114   {
115     VectorMax=dt;
116   }
117
118   //MinMax
119   int ncomp=0;
120   label_leaf_leaf = label_leaf + "/MinMax";
121   if ( tpCntl.chkNode(label_leaf_leaf) )  //があれば
122   {
123     ncomp = tpCntl.countLabels(label_leaf_leaf);
124   }
125
126   ncnt++;
127
128   Min.clear();
129   Max.clear();
130
131   for ( int j=0; j<ncomp; j++ ) {
132
133     if(!tpCntl.GetNodeStr(label_leaf,j+ncnt,&str))
134     {
135       printf("\tCIO Parsing error : No Elem name\n");
```

```
136       return CIO::E_CIO_ERROR_READ_DFI_NO_MINMAX;
137     }
138     if( strcasecmp(str.substr(0,6).c_str(), "minmax") ) continue;
139     label_leaf_leaf = label_leaf+"/"+str;
140
141     label = label_leaf_leaf + "/Min";
142     if ( !(tpCntl.GetValue(label, &dt )) ) {
143       printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
144       return CIO::E_CIO_ERROR_READ_DFI_MIN;
145     }
146     else {
147       Min.push_back(dt);
148     }
149
150     label = label_leaf_leaf + "/Max";
151     if ( !(tpCntl.GetValue(label, &dt )) ) {
152       printf("\tCIO Parsing error : fail to get '%s'\n",label.c_str());
153       return CIO::E_CIO_ERROR_READ_DFI_MAX;
154     }
155     else {
156       Max.push_back(dt);
157     }
158
159   }
160
161   return CIO::E_CIO_SUCCESS;
162 }
```

6.15.3.2 CIO::E_CIO_ERRORCODE cio_Slice::Write (FILE * *fp*, const unsigned *tab*)

DFI ファイル:TimeSlice 要素を出力する

引数

in	*fp*	ファイルポインタ
in	*tab*	インデント

戻り値

error code

cio_TimeSlice.C の 167 行で定義されています。

参照先 _CIO_WRITE_TAB, AveragedStep, AveragedTime, avr_mode, CIO::E_CIO_SUCCESS, Max, Min, step, time, VectorMax, と VectorMin.

```
169 {
170
171   _CIO_WRITE_TAB(fp, tab);
172   fprintf(fp, "Step = %u\n",step);
173
174   _CIO_WRITE_TAB(fp, tab);
175   fprintf(fp, "Time = %e\n",time);
176
177   if( !avr_mode ) {
178     _CIO_WRITE_TAB(fp, tab);
179     fprintf(fp, "AveragedStep = %u\n",AveragedStep);
180     _CIO_WRITE_TAB(fp, tab);
181     fprintf(fp, "AveragedTime = %e\n",AveragedTime);
182   }
183
184   if( Min.size()>1 ) {
185     _CIO_WRITE_TAB(fp, tab);
186     fprintf(fp, "VectorMinMax {\n");
187     _CIO_WRITE_TAB(fp, tab+1);
188     fprintf(fp, "Min = %e\n",VectorMin);
189     _CIO_WRITE_TAB(fp, tab+1);
190     fprintf(fp, "Max = %e\n",VectorMax);
191     _CIO_WRITE_TAB(fp, tab);
192     fprintf(fp, "}\n");
193   }
194
195   for(int j=0; j<Min.size(); j++){
196     _CIO_WRITE_TAB(fp, tab);
197     fprintf(fp, "MinMax[@] {\n");
198     _CIO_WRITE_TAB(fp, tab+1);
199     fprintf(fp, "Min = %e\n",Min[j]);
```

```
200     _CIO_WRITE_TAB(fp, tab+1);
201     fprintf(fp, "Max = %e\n",Max[j]);
202     _CIO_WRITE_TAB(fp, tab);
203     fprintf(fp, "}\n");
204   }
205
206   return CIO::E_CIO_SUCCESS;
207
208 }
```

6.15.4 変数

6.15.4.1 int cio_Slice::AveragedStep

平均ステップ

cio_TimeSlice.h の 26 行で定義されています。

参照元 cio_TimeSlice::AddSlice(), cio_Slice(), Read(), と Write().

6.15.4.2 double cio_Slice::AveragedTime

平均タイム

cio_TimeSlice.h の 27 行で定義されています。

参照元 cio_TimeSlice::AddSlice(), cio_Slice(), Read(), と Write().

6.15.4.3 bool cio_Slice::avr_mode

Average 出力フラグ true:出力なし、false:出力

cio_TimeSlice.h の 25 行で定義されています。

参照元 cio_TimeSlice::AddSlice(), と Write().

6.15.4.4 vector<double> cio_Slice::Max

最大値

cio_TimeSlice.h の 31 行で定義されています。

参照元 cio_TimeSlice::AddSlice(), cio_Slice(), Read(), と Write().

6.15.4.5 vector<double> cio_Slice::Min

最小値

cio_TimeSlice.h の 30 行で定義されています。

参照元 cio_TimeSlice::AddSlice(), cio_Slice(), Read(), と Write().

6.15.4.6 int cio_Slice::step

ステップ番号

cio_TimeSlice.h の 23 行で定義されています。

参照元 cio_TimeSlice::AddSlice(), cio_Slice(), Read(), と Write().

6.15.4.7 double cio_Slice::time

時刻

cio_TimeSlice.h の 24 行で定義されています。

参照元 cio_TimeSlice::AddSlice(), cio_Slice(), Read(), と Write().

6.15.4.8 double cio_Slice::VectorMax

Vector のとき、最大値の合成値

cio_TimeSlice.h の 29 行で定義されています。

参照元 cio_TimeSlice::AddSlice(), cio_Slice(), Read(), と Write().

6.15.4.9 double cio_Slice::VectorMin

Vector のとき、最小値の合成値

cio_TimeSlice.h の 28 行で定義されています。

参照元 cio_TimeSlice::AddSlice(), cio_Slice(), Read(), と Write().

このクラスの説明は次のファイルから生成されました:

- cio_TimeSlice.h
- cio_TimeSlice.C

6.16 クラス cio_TextParser

```
#include <cio_TextParser.h>
```

Public メソッド

- cio_TextParser ()
- ~cio_TextParser ()
- bool GetVector (const std::string label, int *vec, const int nvec)
  TextParser **入力ファイルからベクトル値を取得する（整数型）**
- bool GetVector (const std::string label, double *vec, const int nvec)
  TextParser **入力ファイルからベクトル値を取得する（実数型）**
- bool GetVector (const std::string label, std::string *vec, const int nvec)
  TextParser **入力ファイルからベクトル値を取得する（文字列型）**
- bool GetValue (const std::string label, int *ct)
  TextParser **入力ファイルから変数を取得する（整数型）**
- bool GetValue (const std::string label, double *ct)
  TextParser **入力ファイルから変数を取得する（実数型）**
- bool GetValue (const std::string label, std::string *ct)
  TextParser **入力ファイルから変数を取得する（文字列型）**
- bool chkLabel (const std::string label)
  ラベルの有無をチェック
- bool chkNode (const std::string label)
  ノードの有無をチェック
- bool GetNodeStr (const std::string label, const int nnode, std::string *ct)
  ノード以下の *nnode* **番目の文字列を取得する**

- int countLabels (const std::string label)
  ノード以下のラベルの数を数える
- void getTPinstance ()
  TextParserLibrary **のインスタンス生成**
- int readTPfile (const std::string filename)
  TextParser **オブジェクトに入力ファイルをセットする**
- int remove ()

Private 変数

- TextParser * tp
  テキストパーサ

6.16.1 説明

cio_TextParser.h の 30 行で定義されています。

6.16.2 コンストラクタとデストラクタ

6.16.2.1 cio_TextParser::cio_TextParser ( ) [inline]

コンストラクタ

cio_TextParser.h の 37 行で定義されています。

```
37 {};
```

6.16.2.2 cio_TextParser::~cio_TextParser ( ) [inline]

デストラクタ

cio_TextParser.h の 40 行で定義されています。

```
40 {};
```

6.16.3 関数

6.16.3.1 bool cio_TextParser::chkLabel (const std::string *label*)

ラベルの有無をチェック

引数

in	*label*	チェックするラベル（絶対パス）

cio_TextParser.C の 21 行で定義されています。

参照先 tp.

参照元 GetValue(), と GetVector().

```
22 {
23   int ierror;
24   std::string value;
25
26   if( !tp ) return false;
27
```

```
28   // ラベルがあるかチェック
29   vector<std::string> labels;
30
31   ierror=tp->getAllLabels(labels);
32
33   if (ierror != 0)
34   {
35     cout <<  "ERROR in TextParser::getAllLabels file: "
36     << " ERROR CODE "<< ierror << endl;
37     return false;
38   }
39
40   int flag=0;
41   for (int i = 0; i < labels.size(); i++)
42   {
43           if( !strcasecmp(label.c_str(), labels[i].c_str()) )
44     {
45                   flag=1;
46                   break;
47           }
48   }
49
50   if (flag==0)
51   {
52           return false;
53   }
54
55   return true;
56 }
```

6.16.3.2 bool cio_TextParser::chkNode (const std::string *label*)

ノードの有無をチェック

引数

in	*label*	チェックするノード（絶対パス）

cio_TextParser.C の 62 行で定義されています。

参照先 tp.

参照元 cio_Slice::Read(), cio_FileInfo::Read(), cio_TimeSlice::Read(), cio_Process::Read(), と cio_Unit::Read().

```
63 {
64   int ierror;
65   std::string node;
66   vector<std::string> labels;
67   int len=label.length();
68
69   if( !tp ) return false;
70
71   // Node があるかチェック
72   ierror = tp->getAllLabels(labels);
73
74   if (ierror != 0)
75   {
76     cout <<  "ERROR in TextParser::getAllLabels file: "
77     << " ERROR CODE "<< ierror << endl;
78     return false;
79   }
80
81   int flag=0;
82   for (int i = 0; i < labels.size(); i++) {
83           node = labels[i].substr(0,len);
84
85     if ( !strcasecmp(node.c_str(), label.c_str()) )
86     {
87                   flag=1;
88                   break;
89           }
90   }
91
92   if (flag==0)
93   {
94           return false;
95   }
96
97   return true;
98 }
```

6.16.3.3 int cio_TextParser::countLabels (const std::string *label*)

ノード以下のラベルの数を数える

引数

in	*label*	ラベルを数えるノードの絶対パス

戻り値

ラベルの数（エラー、もしくはない場合は-1 を返す）	

cio_TextParser.C の 104 行で定義されています。

参照先 tp.

参照元 cio_Slice::Read(), cio_FileInfo::Read(), cio_TimeSlice::Read(), cio_Process::Read(), と cio_Unit::Read().

```
105 {
106   int ierror;
107   std::string node,str,chkstr="";
108   vector<std::string> labels;
109   int len=label.length();
110   int flag=0;
111   int inode=0;
112   int next=0;
113
114   if( !tp ) return -1;
115
116   // Node があるかチェック
117   ierror=tp->getAllLabels(labels);
118
119   if (ierror != 0){
120     cout <<  "ERROR in TextParser::getAllLabels file: "
121     << " ERROR CODE "<< ierror << endl;
122     return -1;
123   }
124
125   for (int i = 0; i < labels.size(); i++) {
126           node=labels[i].substr(0,len);
127
128           if( !strcasecmp(node.c_str(), label.c_str()) ){
129                   str=labels[i].substr(len+1);
130                   next=str.find("/");
131
132                   if(next==0) inode++;
133                   else{
134                           if(chkstr!=str.substr(0,next)){
135                                   chkstr=str.substr(0,next);
136                                   inode++;
137                           }
138                   }
139
140           }
141   }
142
143   return inode;
144 }
```

6.16.3.4 bool cio_TextParser::GetNodeStr (const std::string *label*, const int *nnode*, std::string * *ct*)

ノード以下の nnode 番目の文字列を取得する

引数

in	*label*	ノードの絶対パス
in	*nnode*	取得する文字列が現れる順番
out	*ct*	取得した文字列

cio_TextParser.C の 159 行で定義されています。

参照先 tp.

参照元 cio_Slice::Read(), cio_FileInfo::Read(), cio_TimeSlice::Read(), cio_Process::Read(), と cio_Unit::Read().

```
160 {
161   if ( !tp ) return -1;
162
163   int ierror;
164   int len=label.length();
165   int flag=0;
166   int inode=0;
167   int next=0;
168
169   std::string node;
170   std::string str;
171   std::string chkstr="";
172   vector<std::string> labels;
173
174
175   // Node があるかチェック
176   ierror = tp->getAllLabels(labels);
177
178   if (ierror != 0)
179   {
180     cout <<  "ERROR in TextParser::getAllLabels file: " << " ERROR CODE "<< ierror << endl;
181     return false;
182   }
183
184   for (int i = 0; i < labels.size(); i++) {
185           node = labels[i].substr(0, len);
186
187           if ( !strcasecmp(node.c_str(), label.c_str()) )
188     {
189                   str = labels[i].substr(len+1);
190                   next = str.find("/");
191
192                   if ( next == 0 )
193       {
194                           inode++;
195                   }
196                   else
197       {
198                           if ( chkstr != str.substr(0, next) )
199         {
200                                   chkstr = str.substr(0, next);
201                                   inode++;
202                           }
203                   }
204
205                   if ( inode == nnode )
206       {
207                           *ct = chkstr;
208                           return true;
209                   }
210           }
211   }
212   return false;
213 }
```

6.16.3.5 void cio_TextParser::getTPinstance ( )

TextParserLibrary のインスタンス生成

戻り値

エラーコード

cio_TextParser.C の 150 行で定義されています。

参照先 tp.

参照元 cio_DFI::ReadInit().

```
151 {
152   tp = new TextParser;
153 }
```

6.16.3.6 bool cio_TextParser::GetValue (const std::string *label,* int * *ct*)

TextParser 入力ファイルから変数を取得する（整数型）

引数

in	*label*	取得する変数のラベル（絶対パス）
out	*ct*	変数格納ポインタ

cio_TextParser.C の 341 行で定義されています。

参照先 chkLabel(), と tp.

参照元 cio_Rank::Read(), cio_FilePath::Read(), cio_MPI::Read(), cio_Slice::Read(), cio_UnitElem::Read(), cio_Domain::Read(), と cio_FileInfo::Read().

```
342 {
343   int ierror;
344   std::string value;
345
346   if( !tp ) return false;
347
348   // ラベルがあるかチェック
349   if( !chkLabel(label)){
350           return false;
351   }
352
353   //値の取得
354   ierror=tp->getValue(label,value);//label は絶対パスを想定
355   if (ierror != TP_NO_ERROR){
356           cout << " label: " << label << endl;
357           cout <<  "ERROR no label " << label << ierror << endl;
358           return false;
359   }
360
361   //型の取得
362   TextParserValueType type = tp->getType(label, &ierror);
363   if (ierror != TP_NO_ERROR){
364           cout << " label: " << label << endl;
365           cout <<  "ERROR in TextParser::getType file: " << ierror << endl;
366           return false;
367   }
368   if( type != TP_NUMERIC_VALUE ){
369           cout << " label: " << label << endl;
370           cout <<  "ERROR in TextParser::Type error: " << ierror << endl;
371           return false;
372   }
373
374   // string to real
375   int val = tp->convertInt(value, &ierror);
376   if (ierror != TP_NO_ERROR){
377           cout << " label: " << label << endl;
378           cout <<  "ERROR convertInt " << ierror << endl;
379           return false;
380   }
381
382   *ct=val;
383
384   return true;
385 }
```

6.16.3.7 bool cio_TextParser::GetValue (const std::string *label*, double * *ct*)

TextParser 入力ファイルから変数を取得する（実数型）

引数

in	*label*	取得する変数のラベル（絶対パス）
out	*ct*	変数格納ポインタ

cio_TextParser.C の 390 行で定義されています。

参照先 chkLabel(), と tp.

```
391 {
392   int ierror;
393   std::string value;
394   std::string node;
395
396   if( !tp ) return false;
397
```

```
398   // ラベルがあるかチェック
399   if( !chkLabel(label)){
400           return false;
401   }
402
403   //値の取得
404   ierror=tp->getValue(label,value);//label は絶対パスを想定
405   if (ierror != TP_NO_ERROR){
406           cout << " label: " << label << endl;
407           cout <<  "ERROR no label " << ierror << endl;
408           return false;
409   }
410
411   //型の取得
412   TextParserValueType type = tp->getType(label, &ierror);
413   if (ierror != TP_NO_ERROR){
414           cout << " label: " << label << endl;
415           cout <<  "ERROR in TextParser::getType file: " << ierror << endl;
416           return false;
417   }
418   if( type != TP_NUMERIC_VALUE ){
419           cout << " label: " << label << endl;
420           cout <<  "ERROR in TextParser::Type error: " << ierror << endl;
421           return false;
422   }
423
424   // string to real
425   //REAL_TYPE val = tp->convertFloat(value, &ierror);
426   double val = tp->convertFloat(value, &ierror);
427   if (ierror != TP_NO_ERROR){
428           cout << " label: " << label << endl;
429           cout <<  "ERROR convertInt " << ierror << endl;
430           return false;
431   }
432
433   *ct=val;
434
435   return true;
436 }
```

6.16.3.8 bool cio_TextParser::GetValue (const std::string *label*, std::string * *ct*)

TextParser 入力ファイルから変数を取得する（文字列型）

引数

in	*label*	取得する変数のラベル（絶対パス）
out	*ct*	変数格納ポインタ

cio_TextParser.C の 441 行で定義されています。

参照先 chkLabel(), と tp.

```
442 {
443   int ierror;
444   std::string value;
445
446   if ( !tp ) return false;
447
448   // ラベルがあるかチェック
449   if ( !chkLabel(label) )
450   {
451           return false;
452   }
453
454   //値の取得
455   ierror = tp->getValue(label, value); //label は絶対パスを想定
456
457   if (ierror != TP_NO_ERROR)
458   {
459           cout << " label: " << label << endl;
460           cout <<  "ERROR no label " << label << endl;
461           return false;
462   }
463
464   //型の取得
465   TextParserValueType type = tp->getType(label, &ierror);
466   if (ierror != TP_NO_ERROR)
467   {
468           cout << " label: " << label << endl;
```

```
469           cout <<  "ERROR in TextParser::getType file: " << ierror << endl;
470           return false;
471   }
472
473   if( type != TP_STRING_VALUE )
474   {
475           cout << " label: " << label << endl;
476           cout <<  "ERROR in TextParser::Type error: " << ierror << endl;
477           return false;
478   }
479
480   *ct=value;
481
482   return true;
483 }
```

6.16.3.9 bool cio_TextParser::GetVector (const std::string *label*, int * *vec*, const int *nvec*)

TextParser 入力ファイルからベクトル値を取得する（整数型）

引数

in	*label*	取得するベクトルのラベル（絶対パス）
out	*vec*	ベクトル格納配列ポインタ
in	*nvec*	ベクトルサイズ

cio_TextParser.C の 219 行で定義されています。

参照先 chkLabel(), と tp.

参照元 cio_Rank::Read(), と cio_Domain::Read().

```
220 {
221   int ierr = TP_NO_ERROR;
222   std::string value;
223
224   if( !tp ) return false;
225
226   // ラベルがあるかチェック
227   if( !chkLabel(label)){
228           return false;
229   }
230
231   // get value
232   if( (ierr = tp->getValue(label, value)) != TP_NO_ERROR ) return false;
233
234   // get type
235   TextParserValueType type = tp->getType(label, &ierr);
236   if( ierr != TP_NO_ERROR ) return false;
237   if( type != TP_VECTOR_NUMERIC ) return false;
238
239   // split
240   vector<std::string> vec_value;
241   if( (ierr = tp->splitVector(value, vec_value)) != TP_NO_ERROR ) return false;
242
243   // check number of vector element
244   if( vec_value.size() != nvec ) return false;
245
246   // string to real
247   for(int i=0; i<vec_value.size(); i++ )
248   {
249     vec[i] = tp->convertInt(vec_value[i], &ierr);
250     if( ierr != TP_NO_ERROR ) return false;
251   }
252
253   return true;
254 }
```

6.16.3.10 bool cio_TextParser::GetVector (const std::string *label*, double * *vec*, const int *nvec*)

TextParser 入力ファイルからベクトル値を取得する（実数型）

引数

in	*label*	取得するベクトルのラベル（絶対パス）
out	*vec*	ベクトル格納配列ポインタ
in	*nvec*	ベクトルサイズ

cio_TextParser.C の 259 行で定義されています。

参照先 chkLabel(), と tp.

```
260 {
261   int ierr = TP_NO_ERROR;
262   std::string value;
263
264   if( !tp ) return false;
265
266   // ラベルがあるかチェック
267   if( !chkLabel(label)){
268           return false;
269   }
270
271   // get value
272   if( (ierr = tp->getValue(label, value)) != TP_NO_ERROR ){
273     cout << " GetVector debug 333" << endl;
274           return false;
275   }
276
277   // get type
278   TextParserValueType type = tp->getType(label, &ierr);
279   if( ierr != TP_NO_ERROR ) return false;
280   if( type != TP_VECTOR_NUMERIC ) return false;
281
282   // split
283   vector<std::string> vec_value;
284   if( (ierr = tp->splitVector(value, vec_value)) != TP_NO_ERROR ) return false;
285
286   // check number of vector element
287   if( vec_value.size() != nvec ) return false;
288
289   // string to real
290   for(int i=0; i<vec_value.size(); i++ )
291   {
292     vec[i] = tp->convertDouble(vec_value[i], &ierr);
293     if( ierr != TP_NO_ERROR ) return false;
294   }
295
296   return true;
297 }
```

6.16.3.11 bool cio_TextParser::GetVector (const std::string *label*, std::string * *vec*, const int *nvec*)

TextParser 入力ファイルからベクトル値を取得する（文字列型）

引数

in	*label*	取得するベクトルのラベル（絶対パス）
out	*vec*	ベクトル格納配列ポインタ
in	*nvec*	ベクトルサイズ

cio_TextParser.C の 302 行で定義されています。

参照先 chkLabel(), と tp.

```
303 {
304   int ierr = TP_NO_ERROR;
305   std::string value;
306
307   if( !tp ) return false;
308
309   // ラベルがあるかチェック
310   if( !chkLabel(label)){
311           return false;
312   }
313
314   // get value
315   if( (ierr = tp->getValue(label, value)) != TP_NO_ERROR ) return false;
```

```
316
317   // get type
318   TextParserValueType type = tp->getType(label, &ierr);
319   if( ierr != TP_NO_ERROR ) return false;
320   if( type != TP_VECTOR_NUMERIC ) return false;
321
322   // split
323   vector<std::string> vec_value;
324   if( (ierr = tp->splitVector(value, vec_value)) != TP_NO_ERROR ) return false;
325
326   // check number of vector element
327   if( vec_value.size() != nvec ) return false;
328
329   // string to string
330   for(int i=0;i<vec_value.size();i++ )
331   {
332     vec[i] = vec_value[i];
333   }
334
335   return true;
336 }
```

6.16.3.12 int cio_TextParser::readTPfile (const std::string *filename*)

TextParser オブジェクトに入力ファイルをセットする

引数

in	*filename*	入力ファイル名

戻り値

エラーコード	

cio_TextParser.C の 489 行で定義されています。

参照先 tp.

参照元 cio_DFI::ReadInit().

```
490 {
491   int ierr = TP_NO_ERROR;
492   if( !tp ) return TP_ERROR;
493
494   // read
495   if( (ierr = tp->read(filename)) != TP_NO_ERROR )
496   {
497     cout << "ERROR : in input file: " << filename << endl
498     << "  ERROR CODE = "<< ierr << endl;
499     return ierr;
500   }
501   return ierr;
502 }
```

6.16.3.13 int cio_TextParser::remove ( ) [inline]

テキストパーサーの内容を破棄

cio_TextParser.h の 140 行で定義されています。

参照元 cio_DFI::ReadInit().

```
141   {
142     return tp->remove();
143   }
```

6.16.4 変数

6.16.4.1 TextParser* cio_TextParser::tp [private]

テキストパーサ

cio_TextParser.h の 33 行で定義されています。

参照元 chkLabel(), chkNode(), countLabels(), GetNodeStr(), getTPinstance(), GetValue(), GetVector(), と readTPfile().

このクラスの説明は次のファイルから生成されました:

- cio_TextParser.h
- cio_TextParser.C

6.17 クラス cio_TimeSlice

`#include <cio_TimeSlice.h>`

Public メソッド

- cio_TimeSlice ()
- ~cio_TimeSlice ()
- CIO::E_CIO_ERRORCODE Read (cio_TextParser tpCntl)
  TimeSlice **要素を読込む** *(inde.dfi)*
- CIO::E_CIO_ERRORCODE Write (FILE *fp, const unsigned tab)
  DFI **ファイル**:*TimeSlice* **要素を出力する**
- CIO::E_CIO_ERRORCODE getVectorMinMax (const unsigned step, double &vec_min, double &vec_max)
  DFI **に出力されている** *minmax* **の合成値を取得**
- CIO::E_CIO_ERRORCODE getMinMax (const unsigned step, const int compNo, double &min_value, double &max_value)
- void AddSlice (int step, double time, double *minmax, int Ncomp, bool avr_mode, int step_avr, double time_avr)
  SliceList **への追加**

Public 変数

- vector< cio_Slice > SliceList

6.17.1 説明

index.dfi ファイルの TimeSlice

cio_TimeSlice.h の 62 行で定義されています。

6.17.2 コンストラクタとデストラクタ

6.17.2.1 cio_TimeSlice::cio_TimeSlice ( )

コンストラクタ

cio_TimeSlice.C の 212 行で定義されています。

参照先 SliceList.

```
213 {
214   SliceList.clear();
215 }
```

6.17.2.2 cio_TimeSlice::∼cio_TimeSlice ( )

デストラクタ

cio_TimeSlice.C の 219 行で定義されています。

```
220 {
221
222 }
```

6.17.3 関数

6.17.3.1 void cio_TimeSlice::AddSlice (int *step*, double *time*, double * *minmax*, int *Ncomp*, bool *avr_mode*, int *step_avr*, double *time_avr*)

SliceList への追加

引数

in	*step*	ステップ番号
in	*time*	時刻
in	*minmax*	minmax
in	*Ncomp*	コンポーネント数
in	*avr_mode*	Average があるかないかのフラグ
in	*step_avr*	Average step
in	*time_avr*	Average time

cio_TimeSlice.C の 341 行で定義されています。

参照先 cio_Slice::AveragedStep, cio_Slice::AveragedTime, cio_Slice::avr_mode, cio_Slice::Max, cio_Slice::Min, SliceList, cio_Slice::step, cio_Slice::time, cio_Slice::VectorMax, と cio_Slice::VectorMin.

参照元 cio_DFI::WriteData().

```
348 {
349
350   cio_Slice slice;
351
352   slice.step = step;
353   slice.time = time;
354
355   //minmax のセット
356   if( minmax ) {
357     //成分が 1 個の場合
358     if( Ncomp == 1 ) {
359       slice.Min.push_back(minmax[0]);
360       slice.Max.push_back(minmax[1]);
361     } else {
362     //成分が複数個の場合
363       for(int i=0; i<Ncomp; i++) {
364         slice.Min.push_back(minmax[i*2]);
365         slice.Max.push_back(minmax[i*2+1]);
366       }
367       slice.VectorMin=minmax[6];
368       slice.VectorMax=minmax[7];
369     }
370   }
371
372   //average のセット
373   slice.avr_mode = avr_mode;
374   if( !avr_mode ) {
375     slice.AveragedStep=step_avr;
376     slice.AveragedTime=time_avr;
377   } else {
378     slice.AveragedStep=0;
379     slice.AveragedTime=0.0;
380   }
381
382   SliceList.push_back(slice);
383
384 }
```

6.17.3.2 CIO::E_CIO_ERRORCODE cio_TimeSlice::getMinMax (const unsigned *step*, const int *compNo*, double & *min_value*, double & *max_value*)

brief DFI に出力されている minmax と minmax の合成値を取得

引数

in	*step*	取得するステップ
in	*compNo*	取得する成分番号 (0〜n)
out	*min_value*	取得した min
out	*max_value*	取得した max

戻り値

error code 取得出来たときは E_CIO_SUCCESS

cio_TimeSlice.C の 322 行で定義されています。

参照先 CIO::E_CIO_ERROR, CIO::E_CIO_SUCCESS, と SliceList.

参照元 cio_DFI::getMinMax().

```
326 {
327
328   for(int i=0;SliceList.size(); i++) {
329     if( (int)step == SliceList[i].step ) {
330       min_value=SliceList[i].Min[compNo];
331       max_value=SliceList[i].Max[compNo];
332       return CIO::E_CIO_SUCCESS;
333     }
334   }
335
336   return CIO::E_CIO_ERROR;
337
338 }
```

6.17.3.3 CIO::E_CIO_ERRORCODE cio_TimeSlice::getVectorMinMax (const unsigned *step*, double & *vec_min*, double & *vec_max*)

DFI に出力されている minmax の合成値を取得

引数

in	*step*	取得するステップ
out	*vec_min*	取得した min の合成値
out	*vec_max*	取得した min の合成値

戻り値

error code 取得出来たときは E_CIO_SUCCESS

cio_TimeSlice.C の 304 行で定義されています。

参照先 CIO::E_CIO_ERROR, CIO::E_CIO_SUCCESS, と SliceList.

参照元 cio_DFI::getVectorMinMax().

```
307 {
308   for(int i=0;SliceList.size(); i++) {
309     if( (int)step == SliceList[i].step ) {
310       vec_min=SliceList[i].VectorMin;
311       vec_max=SliceList[i].VectorMax;
312       return CIO::E_CIO_SUCCESS;
313     }
314   }
315
316   return CIO::E_CIO_ERROR;
317 }
```

6.17.3.4 CIO::E_CIO_ERRORCODE cio_TimeSlice::Read (cio_TextParser *tpCntl*)

TimeSlice 要素を読込む (inde.dfi)

引数

in	*tpCntl*	cio_TextParser クラス

戻り値

error code

cio_TimeSlice.C の 227 行で定義されています。

参照先 cio_TextParser::chkNode(), cio_TextParser::countLabels(), CIO::E_CIO_ERROR_READ_DFI_NO_SLICE, CIO::E_CIO_SUCCESS, cio_TextParser::GetNodeStr(), cio_Slice::Read(), と SliceList.

参照元 cio_DFI::ReadInit().

```
228 {
229
230   std::string str;
231   std::string label_base,label_leaf;
232
233   cio_Slice slice;
234
235   int nnode=0;
236
237   CIO::E_CIO_ERRORCODE iret;
238
239   //TimeSlice
240   nnode=0;
241   label_base = "/TimeSlice";
242   if ( tpCntl.chkNode(label_base) )  //があれば
243   {
244     nnode = tpCntl.countLabels(label_base);
245   }
246
247   for (int i=0; i<nnode; i++) {
248
249     int ncnt=0;
250
251     if(!tpCntl.GetNodeStr(label_base,i+1,&str))
252     {
253       printf("\tCIO Parsing error : No Elem name\n");
254       return CIO::E_CIO_ERROR_READ_DFI_NO_SLICE;
255     }
256     if( strcasecmp(str.substr(0,5).c_str(), "Slice") ) continue;
257     label_leaf=label_base+"/"+str;
258
259     //Slice 要素の読込み
260     iret = slice.Read(tpCntl,label_leaf);
261
262     if( iret == CIO::E_CIO_SUCCESS ) {
263       SliceList.push_back(slice);
264     } else return iret;
265
266   }
267
268   return CIO::E_CIO_SUCCESS;
269
270 }
```

6.17.3.5 CIO::E_CIO_ERRORCODE cio_TimeSlice::Write (FILE * *fp*, const unsigned *tab*)

DFI ファイル:TimeSlice 要素を出力する

引数

in	*fp*	ファイルポインタ
in	*tab*	インデント

戻り値

true:出力成功 false:出力失敗

cio_TimeSlice.C の 275 行で定義されています。

参照先 _CIO_WRITE_TAB, CIO::E_CIO_ERROR, CIO::E_CIO_SUCCESS, と SliceList.

参照元 cio_DFI::WriteIndexDfiFile().

```
277 {
278
279   fprintf(fp, "TimeSlice {\n");
280   fprintf(fp, "\n");
281
282   for(int i=0; i<SliceList.size(); i++) {
283
284     _CIO_WRITE_TAB(fp, tab);
285     fprintf(fp, "Slice[@] {\n");
286
287     //Slice 要素の出力
288     if( SliceList[i].Write(fp,tab+1) != CIO::E_CIO_SUCCESS) return
      CIO::E_CIO_ERROR;
289
290     _CIO_WRITE_TAB(fp, tab);
291     fprintf(fp, "}\n");
292
293   }
294
295   fprintf(fp, "}\n\n");
296   fclose(fp);
297
298   return CIO::E_CIO_SUCCESS;
299 }
```

6.17.4 変数

6.17.4.1 vector<cio_Slice> cio_TimeSlice::SliceList

cio_TimeSlice.h の 66 行で定義されています。

参照元 AddSlice(), cio_TimeSlice(), getMinMax(), getVectorMinMax(), Read(), cio_DFI_BOV::read_averaged(), cio_DFI_BOV::read_HeaderRecord(), Write(), と cio_DFI_AVS::write_avs_header().

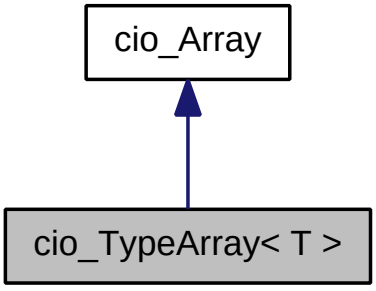
このクラスの説明は次のファイルから生成されました:

- cio_TimeSlice.h
- cio_TimeSlice.C

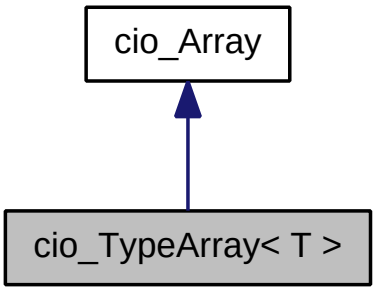
6.18 クラス テンプレート cio_TypeArray< T >

```
#include <cio_TypeArray.h>
```

cio_TypeArray< T > に対する継承グラフ



cio_TypeArray< T > のコラボレーション図



Public メソッド

- cio_TypeArray (CIO::E_CIO_DTYPE dtype, CIO::E_CIO_ARRAYSHAPE shape, size_t ix, size_t jx, size_t kx, size_t gc, size_t ncomp=1)

  コンストラクタ
- cio_TypeArray (T *data, CIO::E_CIO_DTYPE dtype, CIO::E_CIO_ARRAYSHAPE shape, size_t ix, size_t jx, size_t kx, size_t gc, size_t ncomp=1)

  コンストラクタ
- virtual ~cio_TypeArray ()

  デストラクタ
- T * getData (bool extract=false)

  実データのポインタを取得
- const T & val (int i, int j, int k, int l=0) const
- T & val (int i, int j, int k, int l=0)
- const T & hval (int i, int j, int k, int l=0) const
- T & hval (int i, int j, int k, int l=0)
- const T & _val (size_t i, size_t j, size_t k, size_t l=0) const
- T & _val (size_t i, size_t j, size_t k, size_t l=0)
- virtual int copyArray (cio_Array *dst, bool ignoreGc=false)

  配列コピー *(***自信を** *dst* **にコピー。** *head/tail* **を考慮した重複範囲をコピー***)*
- virtual int copyArray (int sta[3], int end[3], cio_Array *dst)

  範囲指定での配列コピー *(***自信を** *dst* **にコピー。** *head/tail* **を考慮した重複範囲をコピー***)*
- virtual int copyArrayNcomp (cio_Array *dst, int comp, bool ignoreGc=false)

  指定成分の配列コピー *(***自信を** *dst* **にコピー。** *head/tail* **を考慮した重複範囲をコピー***)*
- virtual int copyArrayNcomp (int sta[3], int end[3], cio_Array *dst, int comp)

  指定成分の範囲指定での配列コピー *(***自信を** *dst* **にコピー。** *head/tail* **を考慮した重複範囲をコピー***)*
- virtual size_t readBinary (FILE *fp, bool bMatchEndian)

  配列サイズ分のバイナリデータを読み込み *(***戻り値は読み込んだ要素数***)*
- virtual size_t writeBinary (FILE *fp)

  配列サイズ分のバイナリデータを書き出す *(***戻り値は読み込んだ要素数***)*
- virtual size_t writeAscii (FILE *fp)

  配列サイズ分の *ascii* **データを書き出す** *(***戻り値は読み込んだ要素数***)*

Protected メソッド

- cio_TypeArray ()

  デフォルトコンストラクタ

Protected 変数

- bool m_outptr

  実データポインタタイプ

- T * m_data

  実データ配列

Additional Inherited Members

6.18.1 説明

template<class T>class cio_TypeArray< T >

cio_TypeArray.h の 15 行で定義されています。

6.18.2 コンストラクタとデストラクタ

6.18.2.1 template<class T> cio_TypeArray< T >::cio_TypeArray (CIO::E_CIO_DTYPE *dtype*, CIO::E_CIO_ARRAYSHAPE *shape*, size_t *ix*, size_t *jx*, size_t *kx*, size_t *gc*, size_t *ncomp* = 1) [inline]

コンストラクタ

cio_TypeArray.h の 28 行で定義されています。

参照先 cio_TypeArray< T >::m_data, cio_Array::m_gc, cio_Array::m_ncomp, cio_TypeArray< T >::m_outptr, と cio_Array::m_sz.

```
35     : cio_Array(dtype,shape,ix,jx,kx,gc,ncomp)
36   {
37
38     m_outptr=false;
39
40     size_t nw=1;
41     for( int i=0;i<3;i++ )
42     {
43       nw *= (m_sz[i]+2*m_gc);
44     }
45     nw *= m_ncomp;
46     m_data = new T[nw];
47     memset(m_data,0,sizeof(T)*nw);
48   }
```

6.18.2.2 template<class T> cio_TypeArray< T >::cio_TypeArray (T * *data*, CIO::E_CIO_DTYPE *dtype*, CIO::E_CIO_ARRAYSHAPE *shape*, size_t *ix*, size_t *jx*, size_t *kx*, size_t *gc*, size_t *ncomp* = 1) [inline]

コンストラクタ

cio_TypeArray.h の 51 行で定義されています。

参照先 cio_TypeArray< T >::m_data, と cio_TypeArray< T >::m_outptr.

```
59     : cio_Array(dtype,shape,ix,jx,kx,gc,ncomp)
60   {
61
62     m_outptr=true;
63
64     m_data = data;
65   }
```

6.18.2.3 template<class T> virtual cio_TypeArray< T >::~cio_TypeArray ( ) [inline], [virtual]

デストラクタ

cio_TypeArray.h の 68 行で定義されています。

参照先 cio_TypeArray< T >::m_data, と cio_TypeArray< T >::m_outptr.

```
69   {
70     if( m_data )
71     {
72       if( m_data && !m_outptr )
73       {
74         delete [] m_data;
75       }
76     }
77   }
```

6.18.2.4 template<class T> cio_TypeArray< T >::cio_TypeArray ( ) [inline], [protected]

デフォルトコンストラクタ

cio_TypeArray.h の 141 行で定義されています。

参照先 cio_TypeArray< T >::m_data.

```
141                   : cio_Array()
142   {
143     m_data  = NULL;
144   }
```

6.18.3 関数

6.18.3.1 template<class T > cio_TypeArray< T >::_val (size_t *i*, size_t *j*, size_t *k*, size_t *l* = 0) const

参照 (ガイドセルを含む) ガイドセルを含む配列全体の最小インデクスを (0,0,0) とする IJKN のとき val(i,j,k,n) NIJK のとき val(n,i,j,k)

cio_Array_inline.h の 365 行で定義されています。

```
366 {
367   return _val(i,j,k,n);
368 }
```

6.18.3.2 template<class T > cio_TypeArray< T >::_val (size_t *i*, size_t *j*, size_t *k*, size_t *l* = 0)

cio_Array_inline.h の 354 行で定義されています。

```
355 {
356   return m_data[ m_Sz[2] * m_Sz[1] * m_Sz[0] * n
357                + m_Sz[1] * m_Sz[0] * k
358                          + m_Sz[0] * j
359                                    + i ];
360 }
```

6.18.3.3 template<class T > cio_TypeArray< T >::copyArray (cio_Array * *dst*, bool *ignoreGc* = false) [virtual]

配列コピー (自信を dst にコピー。head/tail を考慮した重複範囲をコピー)

cio_Arrayを実装しています。

cio_Array_inline.h の 373 行で定義されています。

参照先 cio_Array::getGcInt(), cio_Array::getHeadIndex(), と cio_Array::getTailIndex().

```
374 {
375   cio_TypeArray<T> *src = this;
376
377   // コピーの範囲
378   int       gcS    = src->getGcInt();
379   const int *headS = src->getHeadIndex();
380   const int *tailS = src->getTailIndex();
381   int       gcD    = dst->getGcInt();
382   const int *headD = dst->getHeadIndex();
383   const int *tailD = dst->getTailIndex();
384   if( ignoreGc )
385   {
386     gcS = gcD = 0;
387   }
388   int sta[3],end[3];
389   for( int i=0;i<3;i++ )
390   {
391     sta[i] = (headS[i]-gcS>=headD[i]-gcD) ? headS[i]-gcS : headD[i]-gcD;
392     end[i] = (tailS[i]+gcS<=tailD[i]+gcD) ? tailS[i]+gcS : tailD[i]+gcD;
393   }
394
395   return copyArray(sta,end,dst);
396 }
```

6.18.3.4 template<class T > cio_TypeArray< T >::copyArray (int *sta[3]*, int *end[3]*, cio_Array * *dst*) [virtual]

範囲指定での配列コピー (自信を dst にコピー。head/tail を考慮した重複範囲をコピー)

cio_Arrayを実装しています。

cio_Array_inline.h の 401 行で定義されています。

参照先 CIO::E_CIO_IJKN, cio_Array::getArrayShape(), cio_Array::getDataType(), cio_Array::getGcInt(), cio_Array::getHeadIndex(), cio_Array::getNcomp(), cio_Array::getTailIndex(), と cio_TypeArray< T >::hval().

```
402 {
403   cio_TypeArray<T> *src = this;
404
405   //mod.s
406   cio_TypeArray<T> *dst = dynamic_cast<cio_TypeArray<T>*>(dstptr);
407   if( !dst )
408   {
409     return 1;
410   }
411
412   // データタイプのチェック
413   if( src->getDataType() != dst->getDataType() )
414   {
415     return 2;
416   }
417   CIO::E_CIO_DTYPE dtype = src->getDataType();
418
419   // 配列形状
420   if( src->getArrayShape() != dst->getArrayShape() )
421   {
422     return 3;
423   }
424   CIO::E_CIO_ARRAYSHAPE shape = src->getArrayShape();
425
426   // 成分数
427   if( src->getNcomp() != src->getNcomp() )
428   {
429     return 4;
430   }
431   int ncomp = src->getNcomp();
432
433   // コピーの範囲
434   int       gcS    = src->getGcInt();
435   const int *headS = src->getHeadIndex();
436   const int *tailS = src->getTailIndex();
437   int       gcD    = dst->getGcInt();
438   const int *headD = dst->getHeadIndex();
439   const int *tailD = dst->getTailIndex();
440   int sta[3],end[3];
441   for( int i=0;i<3;i++ )
442   {
443     sta[i] = (headS[i]-gcS>=headD[i]-gcD) ? headS[i]-gcS : headD[i]-gcD;
444     end[i] = (tailS[i]+gcS<=tailD[i]+gcD) ? tailS[i]+gcS : tailD[i]+gcD;
445   }
446   for( int i=0;i<3;i++ )
447   {
```

```
448     sta[i] = (_sta[i]>=sta[i]) ? _sta[i] : sta[i];
449     end[i] = (_end[i]<=end[i]) ? _end[i] : end[i];
450   }
451
452   // コピー
453   if( m_shape == CIO::E_CIO_IJKN )
454   {
455     for( int n=0;n<ncomp;n++ ){
456     for( int k=sta[2];k<=end[2];k++ ){
457     for( int j=sta[1];j<=end[1];j++ ){
458     for( int i=sta[0];i<=end[0];i++ ){
459       dst->hval(i,j,k,n) = src->hval(i,j,k,n);
460     }}}}
461   }
462   else
463   {
464     for( int k=sta[2];k<=end[2];k++ ){
465     for( int j=sta[1];j<=end[1];j++ ){
466     for( int i=sta[0];i<=end[0];i++ ){
467     for( int n=0;n<ncomp;n++ ){
468       dst->hval(n,i,j,k) = src->hval(n,i,j,k);
469     }}}}
470   }
471
472   return 0;
473 }
```

6.18.3.5 template<class T > cio_TypeArray< T >::copyArrayNcomp (cio_Array * *dst*, int *comp*, bool *ignoreGc* = false) [virtual]

指定成分の配列コピー (自信を dst にコピー。head/tail を考慮した重複範囲をコピー)

cio_Arrayを実装しています。

cio_Array_inline.h の 479 行で定義されています。

参照先 cio_Array::getGcInt(), cio_Array::getHeadIndex(), と cio_Array::getTailIndex().

```
480 {
481   cio_TypeArray<T> *src = this;
482
483   // コピーの範囲
484   int       gcS    = src->getGcInt();
485   const int *headS = src->getHeadIndex();
486   const int *tailS = src->getTailIndex();
487   int       gcD    = dst->getGcInt();
488   const int *headD = dst->getHeadIndex();
489   const int *tailD = dst->getTailIndex();
490   if( ignoreGc )
491   {
492     gcS = gcD = 0;
493   }
494   int sta[3],end[3];
495   for( int i=0;i<3;i++ )
496   {
497     sta[i] = (headS[i]-gcS>=headD[i]-gcD) ? headS[i]-gcS : headD[i]-gcD;
498     end[i] = (tailS[i]+gcS<=tailD[i]+gcD) ? tailS[i]+gcS : tailD[i]+gcD;
499   }
500
501   return copyArrayNcomp(sta,end,dst,comp);
502 }
```

6.18.3.6 template<class T > cio_TypeArray< T >::copyArrayNcomp (int *sta[3]*, int *end[3]*, cio_Array * *dst*, int *comp*) [virtual]

指定成分の範囲指定での配列コピー (自信を dst にコピー。head/tail を考慮した重複範囲をコピー)

cio_Arrayを実装しています。

cio_Array_inline.h の 507 行で定義されています。

参照先 CIO::E_CIO_IJKN, cio_Array::getArrayShape(), cio_Array::getDataType(), cio_Array::getGcInt(), cio_Array::getHeadIndex(), cio_Array::getNcomp(), cio_Array::getTailIndex(), と cio_TypeArray< T >::hval().

```
508 {
509   cio_TypeArray<T> *src = this;
510
511   cio_TypeArray<T> *dst = dynamic_cast<cio_TypeArray<T>*>(dstptr);
512   if( !dst )
513   {
514     return 1;
515   }
516
517   // データタイプのチェック
518   if( src->getDataType() != dst->getDataType() )
519   {
520     return 2;
521   }
522   CIO::E_CIO_DTYPE dtype = src->getDataType();
523
524   //配列形状
525   if( src->getArrayShape() != dst->getArrayShape() )
526   {
527     return 3;
528   }
529   CIO::E_CIO_ARRAYSHAPE shape = src->getArrayShape();
530
531   //成分数
532   if( src->getNcomp() != src->getNcomp() )
533   {
534     return 4;
535   }
536
537   //コピーの範囲
538   int       gcS    = src->getGcInt();
539   const int *headS = src->getHeadIndex();
540   const int *tailS = src->getTailIndex();
541   int       gcD    = dst->getGcInt();
542   const int *headD = dst->getHeadIndex();
543   const int *tailD = dst->getTailIndex();
544   int sta[3],end[3];
545   for( int i=0;i<3;i++ )
546   {
547     sta[i] = (headS[i]-gcS>=headD[i]-gcD) ? headS[i]-gcS : headD[i]-gcD;
548     end[i] = (tailS[i]+gcS<=tailD[i]+gcD) ? tailS[i]+gcS : tailD[i]+gcD;
549   }
550   for( int i=0;i<3;i++ )
551   {
552     sta[i] = (_sta[i]>=sta[i]) ? _sta[i] : sta[i];
553     end[i] = (_end[i]<=end[i]) ? _end[i] : end[i];
554   }
555
556   // コピー
557   if( m_shape == CIO::E_CIO_IJKN )
558   {
559     for( int k=sta[2];k<=end[2];k++ ){
560     for( int j=sta[1];j<=end[1];j++ ){
561     for( int i=sta[0];i<=end[0];i++ ){
562       dst->hval(i,j,k,comp) = src->hval(i,j,k,0);
563     }}}
564   }
565   else
566   {
567     for( int k=sta[2];k<=end[2];k++ ){
568     for( int j=sta[1];j<=end[1];j++ ){
569     for( int i=sta[0];i<=end[0];i++ ){
570       dst->hval(comp,i,j,k) = src->hval(0,i,j,k);
571     }}}
572   }
573
574   return 0;
575
576 }
```

6.18.3.7 template<class T> T* cio_TypeArray< T >::getData (bool *extract* = false) [inline]

実データのポインタを取得

cio_TypeArray.h の 80 行で定義されています。

参照先 cio_TypeArray< T >::m_data.

参照元 cio_Array::getData(), と cio_DFI::setGridData().

```
81   {
82     T *ptr = m_data;
```

```
83     if( extract )
84     {
85       m_data = NULL;
86     }
87     return ptr;
88   }
```

6.18.3.8 template<class T > cio_TypeArray< T >::hval (int *i*, int *j*, int *k*, int *l* = 0) const

参照 (head インデクス考慮版) 実セルの最小インデクスを (head[0],head[1],head[2]) とする IJKN のとき val(i,j,k,n) NIJK のとき val(n,i,j,k)

cio_Array_inline.h の 346 行で定義されています。

参照元 cio_TypeArray< T >::copyArray(), と cio_TypeArray< T >::copyArrayNcomp().

```
347 {
348   return hval(i,j,k,n);
349 }
```

6.18.3.9 template<class T > cio_TypeArray< T >::hval (int *i*, int *j*, int *k*, int *l* = 0)

cio_Array_inline.h の 335 行で定義されています。

```
336 {
337   return m_data[ m_Sz[2] * m_Sz[1] * m_Sz[0] * size_t(n-m_headIndex[3]+m_gcl[3])
338                  + m_Sz[1] * m_Sz[0] * size_t(k-m_headIndex[2]+m_gcl[2])
339                            + m_Sz[0] * size_t(j-m_headIndex[1]+m_gcl[1])
340                                      + size_t(i-m_headIndex[0]+m_gcl[0]) ];
341 }
```

6.18.3.10 template<class T > size_t cio_TypeArray< T >::readBinary (FILE * *fp*, bool *bMatchEndian*) [virtual]

配列サイズ分のバイナリデータを読み込み (戻り値は読み込んだ要素数)

cio_Arrayを実装しています。

cio_Array_inline.h の 659 行で定義されています。

参照先 BSWAPVEC, DBSWAPVEC, と SBSWAPVEC.

```
660 {
661   if( !fp ) return size_t(0);
662   size_t ndata = getArrayLength();
663   size_t nread = fread(m_data,sizeof(T),ndata,fp);
664   if( !bMatchEndian )
665   {
666     size_t bsz = sizeof(T);
667     if( bsz == 2 )
668     {
669       SBSWAPVEC(m_data,nread);
670     }
671     else if( bsz == 4 )
672     {
673       BSWAPVEC(m_data,nread);
674     }
675     else if( bsz == 8 )
676     {
677       DBSWAPVEC(m_data,nread);
678     }
679   }
680   return nread;
681 }
```

6.18.3.11 template<class T > cio_TypeArray< T >::val (int *i,* int *j,* int *k,* int *l* = 0) const

参照 実セルの最小インデクスを (0,0,0) とする IJKN のとき val(i,j,k,n) NIJK のとき val(n,i,j,k)

cio_Array_inline.h の 327 行で定義されています。

参照元 cio_DFI::setGridData(), cio_DFI::VolumeDataDivide(), と cio_DFI_PLOT3D::write_Func().

```
328 {
329   return val(i,j,k,n);
330 }
```

6.18.3.12 template<class T > cio_TypeArray< T >::val (int *i,* int *j,* int *k,* int *l* = 0)

cio_Array_inline.h の 316 行で定義されています。

```
317 {
318   return m_data[ m_Sz[2] * m_Sz[1] * m_Sz[0] * size_t(n+m_gcl[3])
319                          + m_Sz[1] * m_Sz[0] * size_t(k+m_gcl[2])
320                                    + m_Sz[0] * size_t(j+m_gcl[1])
321                                              + size_t(i+m_gcl[0]) ];
322 }
```

6.18.3.13 template<class T > size_t cio_TypeArray< T >::writeAscii (FILE * *fp*) [virtual]

配列サイズ分の ascii データを書き出す (戻り値は読み込んだ要素数)

cio_Arrayを実装しています。

cio_Array_inline.h の 693 行で定義されています。

```
694 {
695   if( !fp ) return size_t(0);
696   //return fwrite(m_data,sizeof(T),getArrayLength(),fp);
697
698   for(int i=0; i<getArrayLength(); i++) {
699     fprintf(fp,"%e\n",(float)m_data[i]);
700   }
701
702   return getArrayLength();
703
704 }
```

6.18.3.14 template<class T > size_t cio_TypeArray< T >::writeBinary (FILE * *fp*) [virtual]

配列サイズ分のバイナリデータを書き出す (戻り値は読み込んだ要素数)

cio_Arrayを実装しています。

cio_Array_inline.h の 685 行で定義されています。

```
686 {
687   if( !fp ) return size_t(0);
688   return fwrite(m_data,sizeof(T),getArrayLength(),fp);
689 }
```

6.18.4 変数

6.18.4.1 template<class T> T* cio_TypeArray< T >::m_data [protected]

実データ配列

cio_TypeArray.h の 158 行で定義されています。

参照元 cio_TypeArray< T >::cio_TypeArray(), cio_TypeArray< T >::getData(), と cio_TypeArray< T >::~cio_TypeArray().

6.18.4.2 template<class T> bool cio_TypeArray< T >::m_outptr [protected]

実データポインタタイプ

cio_TypeArray.h の 155 行で定義されています。

参照元 cio_TypeArray< T >::cio_TypeArray(), と cio_TypeArray< T >::~cio_TypeArray().

このクラスの説明は次のファイルから生成されました:

- cio_TypeArray.h
- cio_Array_inline.h

6.19 クラス cio_Unit

`#include <cio_Unit.h>`

Public メソッド

- cio_Unit ()
- ~cio_Unit ()
- CIO::E_CIO_ERRORCODE Read (cio_TextParser tpCntl)

  read Unit(inde.dfi)
- CIO::E_CIO_ERRORCODE GetUnitElem (const std::string Name, cio_UnitElem &unit)

  該当する*UnitElem* **の取り出し**
- CIO::E_CIO_ERRORCODE GetUnit (const std::string Name, std::string &unit, double &ref, double &diff, bool &bSetDiff)

  単位の取り出し *("m","cm",,,,,)*
- CIO::E_CIO_ERRORCODE Write (FILE *fp, const unsigned tab)

  DFI **ファイル***:Unit* **要素を出力する**

Public 変数

- map< std::string, cio_UnitElem > UnitList

6.19.1 説明

index.dfi ファイルの Unit

cio_Unit.h の 68 行で定義されています。

6.19.2 コンストラクタとデストラクタ

6.19.2.1 cio_Unit::cio_Unit ( )

コンストラクタ

cio_Uit class

cio_Unit.C の 122 行で定義されています。

```
123 {
124
125 }
```

6.19.2.2 cio_Unit::∼cio_Unit ( )

デストラクタ

cio_Unit.C の 129 行で定義されています。

参照先 UnitList.

```
130 {
131
132   UnitList.clear();
133
134 }
```

6.19.3 関数

6.19.3.1 CIO::E_CIO_ERRORCODE cio_Unit::GetUnit (const std::string *Name*, std::string & *unit*, double & *ref*, double & *diff*, bool & *bSetDiff*)

単位の取り出し ("m","cm",,,,,)

引数

in	*Name*	取り出す単位の種類
out	*unit*	単位文字列
out	*ref*	reference
out	*diff*	difference
out	*bSetDiff*	difference 有無フラグ true:あり、false:なし

戻り値

error code

cio_Unit.C の 198 行で定義されています。

参照先 CIO::E_CIO_SUCCESS, CIO::E_CIO_WARN_GETUNIT, と UnitList.

参照元 cio_DFI::GetUnit().

```
203 {
204   map<std::string,cio_UnitElem>::iterator it;
205
206   //Name をキーにして cio_UnitElem を検索
207   it=UnitList.find(Name);
208
209   //見つからなかった場合は空白を返す
210   if( it == UnitList.end() ) {
211     return CIO::E_CIO_WARN_GETUNIT;
212   }
213
214   //単位を返す
215   unit=(*it).second.Unit;
216   ref =(*it).second.reference;
217   diff=(*it).second.difference;
218   BsetDiff=(*it).second.BsetDiff;
219
220   return CIO::E_CIO_SUCCESS;
221
222 }
```

6.19.3.2 CIO::E_CIO_ERRORCODE cio_Unit::GetUnitElem (const std::string *Name*, cio_UnitElem & *unit*)

該当するUnitElem の取り出し

引数

in	*Name*	取り出す単位の種類
out	*unit*	取得した cio_UnitElem クラス

戻り値

error code

cio_Unit.C の 176 行で定義されています。

参照先 CIO::E_CIO_ERROR, CIO::E_CIO_SUCCESS, と UnitList.

参照元 cio_DFI::GetUnitElem().

```
178 {
179   map<std::string,cio_UnitElem>::iterator it;
180
181   //Name をキーにして cio_UnitElem を検索
182   it=UnitList.find(Name);
183
184   //見つからなかった場合は NULL を返す
185   if( it == UnitList.end() ) {
186     return CIO::E_CIO_ERROR;
187   }
188
189   //UnitElem を返す
190   unit = (*it).second;
191
192   return CIO::E_CIO_SUCCESS;
193
194 }
```

6.19.3.3 CIO::E_CIO_ERRORCODE cio_Unit::Read (cio_TextParser *tpCntl*)

read Unit(inde.dfi)

引数

in	*tpCntl*	cio_TextParser クラス

戻り値

error code

UnitElem の読込み

cio_Unit.C の 139 行で定義されています。

参照先 cio_TextParser::chkNode(), cio_TextParser::countLabels(), CIO::E_CIO_SUCCESS, cio_TextParser::GetNodeStr(), cio_UnitElem::Name, cio_UnitElem::Read(), と UnitList.

参照元 cio_DFI::ReadInit().

```
140 {
141
142   std::string str;
143   std::string label_base,label_leaf;
144   int nnode=0;
145   CIO::E_CIO_ERRORCODE iret = CIO::E_CIO_SUCCESS;
146
147   //UnitList
148   label_base = "/UnitList";
149   if ( tpCntl.chkNode(label_base) )  //node があれば
150   {
151     nnode = tpCntl.countLabels(label_base);
152   }
153
154   for(int i=0; i<nnode; i++) {
156     if(!tpCntl.GetNodeStr(label_base,i+1,&str))
157     {
```

```
158       //printf("\tCIO Parsing error : No Elem name\n");
159       return iret;
160     }
161     label_leaf=label_base+"/"+str;
162     cio_UnitElem unit;
163     unit.Name = str;
164     if( unit.Read(tpCntl,label_leaf) == CIO::E_CIO_SUCCESS ) {
165       UnitList.insert(map<std::string,cio_UnitElem>::value_type(str,unit));
166     }
167   }
168
169   return iret;
170
171 }
```

6.19.3.4 CIO::E_CIO_ERRORCODE cio_Unit::Write (FILE * *fp*, const unsigned *tab*)

DFI ファイル:Unit 要素を出力する

引数

in	*fp*	ファイルポインタ
in	*tab*	インデント

戻り値

error code

cio_Unit.C の 313 行で定義されています。

参照先 _CIO_WRITE_TAB, CIO::E_CIO_ERROR, CIO::E_CIO_SUCCESS, と UnitList.

参照元 cio_DFI::WriteIndexDfiFile().

```
315 {
316
317   fprintf(fp, "UnitList {\n");
318   fprintf(fp, "\n");
319
320   map<std::string,cio_UnitElem>::iterator it;
321   for( it=UnitList.begin(); it!=UnitList.end(); it++ ) {
322
323     _CIO_WRITE_TAB(fp, tab+1);
324     fprintf(fp, "%s {\n",(*it).second.Name.c_str());
325
326     if( (*it).second.Write(fp,tab+2) != CIO::E_CIO_SUCCESS ) return
    CIO::E_CIO_ERROR;
327     _CIO_WRITE_TAB(fp, tab+1);
328     fprintf(fp, "}\n");
329   }
330
331   fprintf(fp, "\n");
332   fprintf(fp, "}\n");
333   fprintf(fp, "\n");
334
335   return CIO::E_CIO_SUCCESS;
336
337 }
```

6.19.4 変数

6.19.4.1 map<std::string,cio_UnitElem> cio_Unit::UnitList

cio_Unit.h の 72 行で定義されています。

参照元 cio_DFI::AddUnit(), GetUnit(), GetUnitElem(), Read(), Write(), と ~cio_Unit().

このクラスの説明は次のファイルから生成されました:

- cio_Unit.h
- cio_Unit.C

6.20 クラス cio_UnitElem

`#include <cio_Unit.h>`

Public メソッド

- cio_UnitElem ()
- cio_UnitElem (const std::string _Name, const std::string _Unit, const double _reference, const double _-difference, const bool _BsetDiff)
- ~cio_UnitElem ()
- CIO::E_CIO_ERRORCODE Read (cio_TextParser tpCntl, const std::string label_leaf)

  Unit **要素の読込み**
- CIO::E_CIO_ERRORCODE Write (FILE *fp, const unsigned tab)

  DFI **ファイル**:*Unit* **要素を出力する**

Public 変数

- std::string Name

  単位の種類名 *(Length,Velovity,,,)*
- std::string Unit

  単位のラベル *(m,m/s,Pa,,,,)*
- double reference

  規格化に用いたスケール
- double difference

  差
- bool BsetDiff

  difference **の有無（***false:***なし** *true:***あり）**

6.20.1 説明

cio_Unit.h の 18 行で定義されています。

6.20.2 コンストラクタとデストラクタ

6.20.2.1 cio_UnitElem::cio_UnitElem ( )

コンストラクタ

cio_UnitElem class

cio_Unit.C の 24 行で定義されています。

参照先 BsetDiff, difference, Name, reference, と Unit.

```
25 {
26
27   Name="";
28   Unit="";
29   reference=0.0;
30   difference=0.0;
31   BsetDiff=false;
32
33 }
```

6.20.2.2 cio_UnitElem::cio_UnitElem (const std::string *_Name,* const std::string *_Unit,* const double *_reference,* const double *_difference,* const bool *_BsetDiff*)

コンストラクタ

cio_Unit.C の 37 行で定義されています。

参照先 BsetDiff, difference, Name, reference, と Unit.

```
42 {
43   Name       = _Name;
44   Unit       = _Unit;
45   reference  = _reference;
46   difference = _difference;
47   BsetDiff   = _BsetDiff;
48 }
```

6.20.2.3 cio_UnitElem::~cio_UnitElem ( )

デストラクタ

cio_Unit.C の 53 行で定義されています。

```
54 {
55
56
57 }
```

6.20.3 関数

6.20.3.1 CIO::E_CIO_ERRORCODE cio_UnitElem::Read (cio_TextParser *tpCntl,* const std::string *label_leaf*)

Unit 要素の読込み

引数

in	*tpCntl*	cio_TextParser クラス
in	*label_leaf*	

戻り値

error code

cio_Unit.C の 62 行で定義されています。

参照先 BsetDiff, difference, CIO::E_CIO_SUCCESS, CIO::E_CIO_WARN_GETUNIT, cio_TextParser::GetValue(), reference, と Unit.

参照元 cio_Unit::Read().

```
64 {
65
66   std::string str,label;
67   double dt;
68
69   //単位系のの読込み
70   label = label_leaf + "/Unit";
71   if ( !(tpCntl.GetValue(label, &str )) )
72   {
73     return CIO::E_CIO_WARN_GETUNIT;
74   }
75   Unit=str;
76
77   //値の読込み
78   label = label_leaf + "/Reference";
79   if ( !(tpCntl.GetValue(label, &dt )) )
80   {
81     dt=0.0;
```

```
82   }
83   reference=dt;
84
85   //diff の読込み
86   label = label_leaf + "/Difference";
87   if ( !(tpCntl.GetValue(label, &dt )) )
88   {
89     difference=0.0;
90     BsetDiff=false;
91   } else {
92     difference=dt;
93     BsetDiff=true;
94   }
95
96   return CIO::E_CIO_SUCCESS;
97
98 }
```

6.20.3.2 CIO::E_CIO_ERRORCODE cio_UnitElem::Write (FILE * *fp*, const unsigned *tab*)

DFI ファイル:Unit 要素を出力する

引数

in	*fp*	ファイルポインタ
in	*tab*	インデント

戻り値

error code

cio_Unit.C の 103 行で定義されています。

参照先 _CIO_WRITE_TAB, BsetDiff, difference, CIO::E_CIO_SUCCESS, reference, と Unit.

```
104 {
105
106   _CIO_WRITE_TAB(fp, tab);
107   fprintf(fp, "Unit       = \"%s\"\n",Unit.c_str());
108   _CIO_WRITE_TAB(fp, tab);
109   fprintf(fp, "Reference  = %e\n",reference);
110   if( BsetDiff ) {
111     _CIO_WRITE_TAB(fp, tab);
112     fprintf(fp, "Difference = %e\n",difference);
113   }
114
115   return CIO::E_CIO_SUCCESS;
116
117 }
```

6.20.4 変数

6.20.4.1 bool cio_UnitElem::BsetDiff

difference の有無（false:なし true:あり）

cio_Unit.h の 26 行で定義されています。

参照元 cio_UnitElem(), Read(), と Write().

6.20.4.2 double cio_UnitElem::difference

差

cio_Unit.h の 25 行で定義されています。

参照元 cio_UnitElem(), Read(), と Write().

6.20.4.3 std::string cio_UnitElem::Name

単位の種類名 (Length,Velovity,,,)

cio_Unit.h の 22 行で定義されています。

参照元 cio_UnitElem(), と cio_Unit::Read().

6.20.4.4 double cio_UnitElem::reference

規格化に用いたスケール

cio_Unit.h の 24 行で定義されています。

参照元 cio_UnitElem(), Read(), と Write().

6.20.4.5 std::string cio_UnitElem::Unit

単位のラベル (m,m/s,Pa,,,)

cio_Unit.h の 23 行で定義されています。

参照元 cio_UnitElem(), Read(), と Write().

このクラスの説明は次のファイルから生成されました:

- cio_Unit.h
- cio_Unit.C

Chapter 7

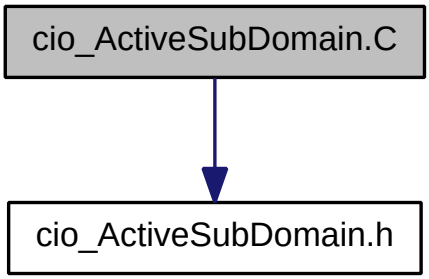
ファイル

7.1 cio_ActiveSubDomain.C

cio_ActiveSubDomain class 関数

```
#include "cio_ActiveSubDomain.h"
```

cio_ActiveSubDomain.C のインクルード依存関係図



7.1.1 説明

cio_ActiveSubDomain class 関数

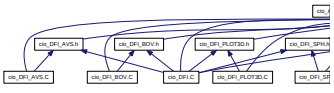
作者

kero

cio_ActiveSubDomain.C で定義されています。

7.2 cio_ActiveSubDomain.h

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。



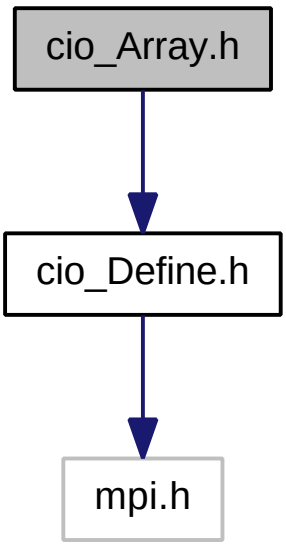
構成

- class cio_ActiveSubDomain

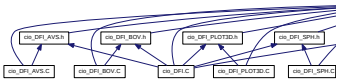
7.3 cio_Array.h

```
#include "cio_Define.h"
```

cio_Array.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 cio_Array

関数

- void cio_interp_ijkn_r4_ (const int ∗szS, const int ∗gcS, const int ∗szD, const int ∗gcD, const int ∗ncomp, float ∗src, float ∗dst)
- void cio_interp_ijkn_r8_ (const int ∗szS, const int ∗gcS, const int ∗szD, const int ∗gcD, const int ∗ncomp, double ∗src, double ∗dst)
- void cio_interp_nijk_r4_ (const int ∗szS, const int ∗gcS, const int ∗szD, const int ∗gcD, const int ∗ncomp, float ∗src, float ∗dst)
- void cio_interp_nijk_r8_ (const int ∗szS, const int ∗gcS, const int ∗szD, const int ∗gcD, const int ∗ncomp, double ∗src, double ∗dst)

7.3.1 関数

7.3.1.1 void cio_interp_ijkn_r4_ (const int * *szS*, const int * *gcS*, const int * *szD*, const int * *gcD*, const int * *ncomp*, float * *src*, float * *dst*)

参照元 cio_Array::interp_coarse().

7.3.1.2 void cio_interp_ijkn_r8_ (const int * *szS*, const int * *gcS*, const int * *szD*, const int * *gcD*, const int * *ncomp*, double * *src*, double * *dst*)

参照元 cio_Array::interp_coarse().

7.3.1.3 void cio_interp_nijk_r4_ (const int * *szS*, const int * *gcS*, const int * *szD*, const int * *gcD*, const int * *ncomp*, float * *src*, float * *dst*)

参照元 cio_Array::interp_coarse().

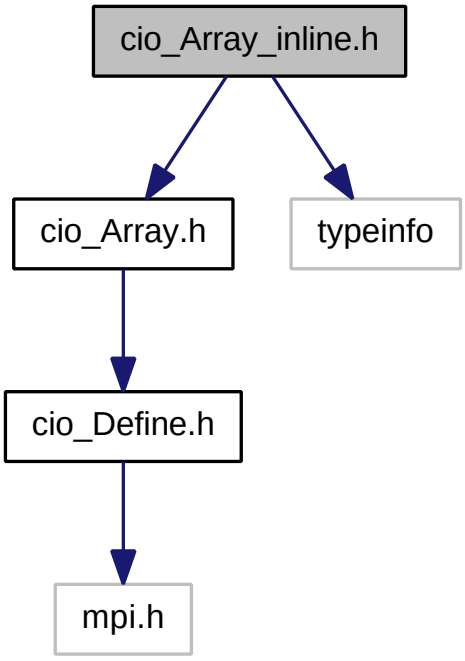
7.3.1.4 void cio_interp_nijk_r8_ (const int * *szS*, const int * *gcS*, const int * *szD*, const int * *gcD*, const int * *ncomp*, double * *src*, double * *dst*)

参照元 cio_Array::interp_coarse().

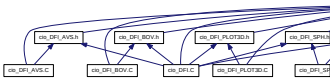
7.4 cio_Array_inline.h

```
#include "cio_Array.h"
#include <typeinfo>
```

cio_Array_inline.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 CIO_INLINE inline
- #define CIO_MEMFUN(rettype) CIO_INLINE rettype

7.4.1 マクロ定義

7.4.1.1 #define CIO_INLINE inline

cio_Array_inline.h の 20 行で定義されています。

7.4.1.2 #define CIO_MEMFUN(*rettype*) CIO_INLINE rettype

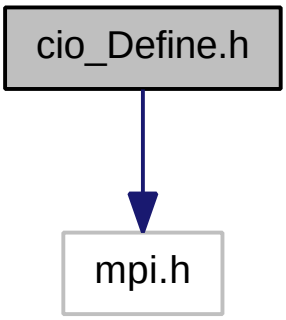
cio_Array_inline.h の 25 行で定義されています。

7.5 cio_Define.h

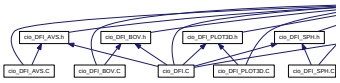
CIO の定義マクロ記述ヘッダーファイル

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

cio_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。



ネームスペース

- CIO

Constant Groups

- CIO

マクロ定義

- #define D_CIO_EXT_SPH "sph"
- #define D_CIO_EXT_BOV "dat"
- #define D_CIO_EXT_FUNC "func"
- #define D_CIO_EXT_VTK "vtk"
- #define D_CIO_ON "on"
- #define D_CIO_OFF "off"
- #define D_CIO_INT8 "Int8"
- #define D_CIO_INT16 "Int16"
- #define D_CIO_INT32 "Int32"
- #define D_CIO_INT64 "Int64"
- #define D_CIO_UINT8 "UInt8"
- #define D_CIO_UINT16 "UInt16"
- #define D_CIO_UINT32 "UInt32"
- #define D_CIO_UINT64 "UInt64"
- #define D_CIO_FLOAT32 "Float32"
- #define D_CIO_FLOAT64 "Float64"
- #define D_CIO_IJNK "ijkn"
- #define D_CIO_NIJK "nijk"
- #define D_CIO_LITTLE "little"
- #define D_CIO_BIG "big"
- #define _CIO_TAB_STR " "
- #define _CIO_IDX_IJK(_I, _J, _K, _NI, _NJ, _NK, _VC)
- #define _CIO_IDX_IJ(_I, _J, _NI, _NJ, _VC)
- #define _CIO_IDX_NIJ(_N, _I, _J, _NI, _NJ, _NN, _VC)
- #define _CIO_IDX_IJKN(_I, _J, _K, _N, _NI, _NJ, _NK, _VC)
- #define _CIO_IDX_NIJK(_N, _I, _J, _K, _NN, _NI, _NJ, _NK, _VC)
- #define _CIO_WRITE_TAB(_FP, _NTAB)

列挙型

- enum CIO::E_CIO_FORMAT { CIO::E_CIO_FMT_UNKNOWN = -1, CIO::E_CIO_FMT_SPH, CIO::E_CIO_FMT_BOV, CIO::E_CIO_FMT_AVS, CIO::E_CIO_FMT_PLOT3D, CIO::E_CIO_FMT_VTK }
- enum CIO::E_CIO_ONOFF { CIO::E_CIO_OFF = 0, CIO::E_CIO_ON }
- enum CIO::E_CIO_DTYPE { CIO::E_CIO_DTYPE_UNKNOWN = 0, CIO::E_CIO_INT8, CIO::E_CIO_INT16, CIO::E_CIO_INT32, CIO::E_CIO_INT64, CIO::E_CIO_UINT8, CIO::E_CIO_UINT16, CIO::E_CIO_UINT32, CIO::E_CIO_UINT64, CIO::E_CIO_FLOAT32, CIO::E_CIO_FLOAT64 }
- enum CIO::E_CIO_ARRAYSHAPE { CIO::E_CIO_ARRAYSHAPE_UNKNOWN =-1, CIO::E_CIO_IJKN =0, CIO::E_CIO_NIJK }
- enum CIO::E_CIO_ENDIANTYPE { CIO::E_CIO_ENDIANTYPE_UNKNOWN =-1, CIO::E_CIO_LITTLE =0, CIO::E_CIO_BIG }

- enum CIO::E_CIO_READTYPE {
  CIO::E_CIO_SAMEDIV_SAMERES =1, CIO::E_CIO_SAMEDIV_REFINEMENT, CIO::E_CIO_DIFFDIV_SAMERES,
  CIO::E_CIO_DIFFDIV_REFINEMENT,
  CIO::E_CIO_READTYPE_UNKNOWN }
- enum CIO::E_CIO_OUTPUT_TYPE { CIO::E_CIO_OUTPUT_TYPE_DEFAULT =-1, CIO::E_CIO_OUTPUT_TYPE_ASCII
  =0, CIO::E_CIO_OUTPUT_TYPE_BINARY, CIO::E_CIO_OUTPUT_TYPE_FBINARY }
- enum CIO::E_CIO_OUTPUT_FNAME { CIO::E_CIO_FNAME_DEFAULT =-1, CIO::E_CIO_FNAME_STEP_RANK
  =0, CIO::E_CIO_FNAME_RANK_STEP }
- enum CIO::E_CIO_ERRORCODE {
  CIO::E_CIO_SUCCESS = 1, CIO::E_CIO_ERROR = -1, CIO::E_CIO_ERROR_READ_DFI_GLOBALORIGIN
  = 1000, CIO::E_CIO_ERROR_READ_DFI_GLOBALREGION = 1001,
  CIO::E_CIO_ERROR_READ_DFI_GLOBALVOXEL = 1002, CIO::E_CIO_ERROR_READ_DFI_GLOBALDIVISION
  = 1003, CIO::E_CIO_ERROR_READ_DFI_DIRECTORYPATH = 1004, CIO::E_CIO_ERROR_READ_DFI_TIMESLICEDIRECT
  = 1005,
  CIO::E_CIO_ERROR_READ_DFI_PREFIX = 1006, CIO::E_CIO_ERROR_READ_DFI_FILEFORMAT =
  1007, CIO::E_CIO_ERROR_READ_DFI_GUIDECELL = 1008, CIO::E_CIO_ERROR_READ_DFI_DATATYPE
  = 1009,
  CIO::E_CIO_ERROR_READ_DFI_ENDIAN = 1010, CIO::E_CIO_ERROR_READ_DFI_ARRAYSHAPE =
  1011, CIO::E_CIO_ERROR_READ_DFI_COMPONENT = 1012, CIO::E_CIO_ERROR_READ_DFI_FILEPATH_PROCESS
  = 1013,
  CIO::E_CIO_ERROR_READ_DFI_NO_RANK = 1014, CIO::E_CIO_ERROR_READ_DFI_ID = 1015,
  CIO::E_CIO_ERROR_READ_DFI_HOSTNAME = 1016, CIO::E_CIO_ERROR_READ_DFI_VOXELSIZE
  = 1017,
  CIO::E_CIO_ERROR_READ_DFI_HEADINDEX = 1018, CIO::E_CIO_ERROR_READ_DFI_TAILINDEX =
  1019, CIO::E_CIO_ERROR_READ_DFI_NO_SLICE = 1020, CIO::E_CIO_ERROR_READ_DFI_STEP =
  1021,
  CIO::E_CIO_ERROR_READ_DFI_TIME = 1022, CIO::E_CIO_ERROR_READ_DFI_NO_MINMAX = 1023,
  CIO::E_CIO_ERROR_READ_DFI_MIN = 1024, CIO::E_CIO_ERROR_READ_DFI_MAX = 1025,
  CIO::E_CIO_ERROR_READ_INDEXFILE_OPENERROR = 1050, CIO::E_CIO_ERROR_TEXTPARSER =
  1051, CIO::E_CIO_ERROR_READ_FILEINFO = 1052, CIO::E_CIO_ERROR_READ_FILEPATH = 1053,
  CIO::E_CIO_ERROR_READ_UNIT = 1054, CIO::E_CIO_ERROR_READ_TIMESLICE = 1055, CIO::E_CIO_ERROR_READ_F
  = 1056, CIO::E_CIO_ERROR_READ_DOMAIN = 1057,
  CIO::E_CIO_ERROR_READ_MPI = 1058, CIO::E_CIO_ERROR_READ_PROCESS = 1059, CIO::E_CIO_ERROR_READ_FIE
  = 1900, CIO::E_CIO_ERROR_READ_SPH_FILE = 2000,
  CIO::E_CIO_ERROR_READ_SPH_REC1 = 2001, CIO::E_CIO_ERROR_READ_SPH_REC2 = 2002,
  CIO::E_CIO_ERROR_READ_SPH_REC3 = 2003, CIO::E_CIO_ERROR_READ_SPH_REC4 = 2004,
  CIO::E_CIO_ERROR_READ_SPH_REC5 = 2005, CIO::E_CIO_ERROR_READ_SPH_REC6 = 2006,
  CIO::E_CIO_ERROR_READ_SPH_REC7 = 2007, CIO::E_CIO_ERROR_UNMATCH_VOXELSIZE = 2050,
  CIO::E_CIO_ERROR_NOMATCH_ENDIAN = 2051, CIO::E_CIO_ERROR_READ_BOV_FILE = 2100,
  CIO::E_CIO_ERROR_READ_FIELD_HEADER_RECORD = 2102, CIO::E_CIO_ERROR_READ_FIELD_DATA_RECORD
  = 2103,
  CIO::E_CIO_ERROR_READ_FIELD_AVERAGED_RECORD = 2104, CIO::E_CIO_ERROR_MISMATCH_NP_SUBDOMAIN
  = 3003, CIO::E_CIO_ERROR_INVALID_DIVNUM = 3011, CIO::E_CIO_ERROR_OPEN_SBDM = 3012,
  CIO::E_CIO_ERROR_READ_SBDM_HEADER = 3013, CIO::E_CIO_ERROR_READ_SBDM_FORMAT =
  3014, CIO::E_CIO_ERROR_READ_SBDM_DIV = 3015, CIO::E_CIO_ERROR_READ_SBDM_CONTENTS
  = 3016,
  CIO::E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO = 3017, CIO::E_CIO_ERROR_MAKEDIRECTORY =
  3100, CIO::E_CIO_ERROR_OPEN_FIELDDATA = 3101, CIO::E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD
  = 3102,
  CIO::E_CIO_ERROR_WRITE_FIELD_DATA_RECORD = 3103, CIO::E_CIO_ERROR_WRITE_FIELD_AVERAGED_RECORD
  = 3104, CIO::E_CIO_ERROR_WRITE_SPH_REC1 = 3201, CIO::E_CIO_ERROR_WRITE_SPH_REC2 =
  3202,
  CIO::E_CIO_ERROR_WRITE_SPH_REC3 = 3203, CIO::E_CIO_ERROR_WRITE_SPH_REC4 = 3204,
  CIO::E_CIO_ERROR_WRITE_SPH_REC5 = 3205, CIO::E_CIO_ERROR_WRITE_SPH_REC6 = 3206,
  CIO::E_CIO_ERROR_WRITE_SPH_REC7 = 3207, CIO::E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY
  = 3500, CIO::E_CIO_ERROR_WRITE_PROCFILE_OPENERROR = 3501, CIO::E_CIO_ERROR_WRITE_DOMAIN
  = 3502,
  CIO::E_CIO_ERROR_WRITE_MPI = 3503, CIO::E_CIO_ERROR_WRITE_PROCESS = 3504, CIO::E_CIO_ERROR_WRITE_

= 3505, CIO::E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY = 3510, CIO::E_CIO_ERROR_WRITE_PREFIX_EMPTY = 3511, CIO::E_CIO_ERROR_WRITE_INDEXFILE_OPENERROR = 3512, CIO::E_CIO_ERROR_WRITE_FILEINFO = 3513, CIO::E_CIO_ERROR_WRITE_UNIT = 3514, CIO::E_CIO_ERROR_WRITE_TIMESLICE = 3515, CIO::E_CIO_ERROR_WRITE_FILEPATH = 3516, CIO::E_CIO_WARN_GETUNIT = 4000 }

7.5.1 説明

CIO の定義マクロ記述ヘッダーファイル

作者

kero

cio_Define.h で定義されています。

7.5.2 マクロ定義

7.5.2.1 #define _CIO_IDX_IJ(_I, _J, _NI, _NJ, _VC)

値:

```
( (long long)((_J)+(_VC)) * (long long)((_NI)+2*(_VC)) \
+ (long long)((_I)+(_VC)) \
)
```

2 次元（スカラー）インデクス (i,j) -> 1 次元インデクス変換マクロ

引数

in	*_I*	i 方向インデクス
in	*_J*	j 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	*_VC*	仮想セル数

戻り値

1 次元インデクス

cio_Define.h の 258 行で定義されています。

7.5.2.2 #define _CIO_IDX_IJK(_I, _J, _K, _NI, _NJ, _NK, _VC)

値:

```
( (long long)((_K)+(_VC)) * (long long)((_NI)+2*(_VC)) * (long long)((_NJ)+2*(_VC)) \
+ (long long)((_J)+(_VC)) * (long long)((_NI)+2*(_VC)) \
+ (long long)((_I)+(_VC)) \
)
```

3 次元（スカラー）インデクス (i,j,k) -> 1 次元インデクス変換マクロ

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

cio_Define.h の 244 行で定義されています。

参照元 cio_Process::CheckStartEnd(), cio_Process::CreateRankList(), と cio_Process::CreateRankMap().

7.5.2.3 #define _CIO_IDX_IJKN(*_I, _J, _K, _N, _NI, _NJ, _NK, _VC*)

値:

```
( (long long)(_N) * (long long)((_NI)+2*(_VC)) * (long long)((_NJ)+2*(_VC)) \
* (long long)((_NK)+2*(_VC)) \
+ _CIO_IDX_IJK(_I,_J,_K,_NI,_NJ,_NK,_VC) \
)
```

3 次元（ベクトル）インデクス (i,j,k,n) -> 1 次元インデクス変換マクロ

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	*_N*	成分インデクス
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

cio_Define.h の 290 行で定義されています。

7.5.2.4 #define _CIO_IDX_NIJ(*_N, _I, _J, _NI, _NJ, _NN, _VC*)

値:

```
( (long long)(_NN)*_CIO_IDX_IJ(_I,_J,_NI,_NJ,_VC) \
 + (long long)(_N) \
)
```

2 次元（スカラー）インデクス (n,i,j) -> 1 次元インデクス変換マクロ

引数

in	*_N*	成分インデクス
in	*_I*	i 方向インデクス
in	*_J*	j 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	*_NN*	成分数
in	*_VC*	仮想セル数

戻り値

1 次元インデクス

cio_Define.h の 273 行で定義されています。

7.5.2.5 #define _CIO_IDX_NIJK(*_N, _I, _J, _K, _NN, _NI, _NJ, _NK, _VC*)

値:

```
( (long long)(_NN) * _CIO_IDX_IJK(_I,_J,_K,_NI,_NJ,_NK,_VC) \
+ (long long)(_N) )
```

3 次元（ベクトル）インデクス (n,i,j,k) -> 1 次元インデクス変換マクロ

引数

in	*_N*	成分インデクス
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	*_NN*	成分数
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

cio_Define.h の 308 行で定義されています。

7.5.2.6 #define _CIO_TAB_STR " "

cio_Define.h の 52 行で定義されています。

7.5.2.7 #define _CIO_WRITE_TAB(*_FP, _NTAB*)

値:

```
{\
 for(int _NTCNT=0; _NTCNT<_NTAB; _NTCNT++) fprintf(_FP,_CIO_TAB_STR); \
}
```

DFI ファイルのTab 出力

引数

in	*_FP*	ファイルポインタ
in	*_NTAB*	インデント数

cio_Define.h の 316 行で定義されています。

参照元 cio_Rank::Write(), cio_FilePath::Write(), cio_MPI::Write(), cio_Slice::Write(), cio_UnitElem::Write(), cio_Domain::Write(), cio_FileInfo::Write(), cio_TimeSlice::Write(), cio_Unit::Write(), と cio_Process::Write().

7.5.2.8 #define D_CIO_BIG "big"

cio_Define.h の 50 行で定義されています。

7.5.2.9 #define D_CIO_EXT_BOV "dat"

cio_Define.h の 26 行で定義されています。

参照元 cio_DFI::Generate_FieldFileName(), と cio_DFI::WriteData().

7.5.2.10 #define D_CIO_EXT_FUNC "func"

cio_Define.h の 28 行で定義されています。

参照元 cio_DFI::Generate_FieldFileName(), と cio_DFI::WriteData().

7.5.2.11 #define D_CIO_EXT_SPH "sph"

cio_Define.h の 25 行で定義されています。

参照元 cio_DFI::Generate_FieldFileName(), と cio_DFI::WriteData().

7.5.2.12 #define D_CIO_EXT_VTK "vtk"

cio_Define.h の 29 行で定義されています。

参照元 cio_DFI::Generate_FieldFileName(), と cio_DFI::WriteData().

7.5.2.13 #define D_CIO_FLOAT32 "Float32"

cio_Define.h の 43 行で定義されています。

参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.14 #define D_CIO_FLOAT64 "Float64"

cio_Define.h の 44 行で定義されています。

参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.15 #define D_CIO_IJNK "ijkn"

cio_Define.h の 46 行で定義されています。

参照元 cio_DFI::GetArrayShapeString().

7.5.2.16 #define D_CIO_INT16 "Int16"

cio_Define.h の 36 行で定義されています。
参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.17 #define D_CIO_INT32 "Int32"

cio_Define.h の 37 行で定義されています。
参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.18 #define D_CIO_INT64 "Int64"

cio_Define.h の 38 行で定義されています。
参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.19 #define D_CIO_INT8 "Int8"

cio_Define.h の 35 行で定義されています。
参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.20 #define D_CIO_LITTLE "little"

cio_Define.h の 49 行で定義されています。

7.5.2.21 #define D_CIO_NIJK "nijk"

cio_Define.h の 47 行で定義されています。
参照元 cio_DFI::GetArrayShapeString().

7.5.2.22 #define D_CIO_OFF "off"

cio_Define.h の 33 行で定義されています。

7.5.2.23 #define D_CIO_ON "on"

cio_Define.h の 32 行で定義されています。

7.5.2.24 #define D_CIO_UINT16 "UInt16"

cio_Define.h の 40 行で定義されています。
参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.25 #define D_CIO_UINT32 "UInt32"

cio_Define.h の 41 行で定義されています。
参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.26 #define D_CIO_UINT64 "UInt64"

cio_Define.h の 42 行で定義されています。

参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.27 #define D_CIO_UINT8 "UInt8"

cio_Define.h の 39 行で定義されています。

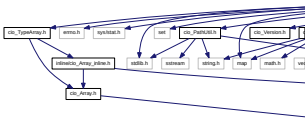
参照元 cio_DFI::ConvDatatypeE2S().

7.6 cio_DFI.C

cio_DFI Class

```
#include "cio_DFI.h"
#include <unistd.h>
#include "cio_DFI_SPH.h"
#include "cio_DFI_BOV.h"
#include "cio_DFI_AVS.h"
#include "cio_DFI_PLOT3D.h"
#include "cio_DFI_VTK.h"
```

cio_DFI.C のインクルード依存関係図



7.6.1 説明

cio_DFI Class

作者

kero

cio_DFI.C で定義されています。

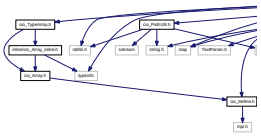
7.7 cio_DFI.h

cio_DFI Class Header

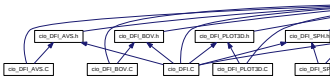
```
#include "cio_Define.h"
```

```
#include <stdlib.h>
#include <errno.h>
#include <sys/stat.h>
#include <typeinfo>
#include <set>
#include <map>
#include <string>
#include "cio_Version.h"
#include "cio_PathUtil.h"
#include "cio_TextParser.h"
#include "cio_ActiveSubDomain.h"
#include "cio_endianUtil.h"
#include "cio_TypeArray.h"
#include "cio_FileInfo.h"
#include "cio_FilePath.h"
#include "cio_Unit.h"
#include "cio_TimeSlice.h"
#include "cio_Domain.h"
#include "cio_MPI.h"
#include "cio_Process.h"
#include "inline/cio_DFI_inline.h"
```

cio_DFI.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 cio_DFI

7.7.1 説明

cio_DFI Class Header

作者

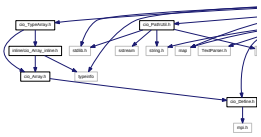
kero

cio_DFI.h で定義されています。

7.8 cio_DFI_AVS.C

cio_DFI_AVS Class

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



7.8.1 説明

cio_DFI_AVS Class

作者

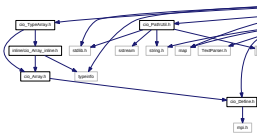
kero

cio_DFI_AVS.C で定義されています。

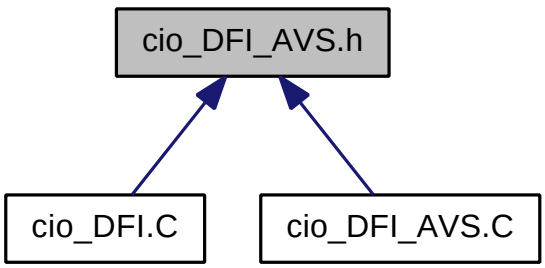
7.9 cio_DFI_AVS.h

cio_DFI_AVS Class Header

```
#include "cio_DFI.h"
```
cio_DFI_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 cio_DFI_AVS

7.9.1 説明

cio_DFI_AVS Class Header

作者

kero

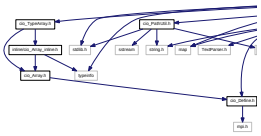
cio_DFI_AVS.h で定義されています。

7.10 cio_DFI_BOV.C

cio_DFI_BOV Class

```
#include "cio_DFI.h"
#include "cio_DFI_BOV.h"
```

cio_DFI_BOV.C のインクルード依存関係図



7.10.1 説明

cio_DFI_BOV Class

作者

kero

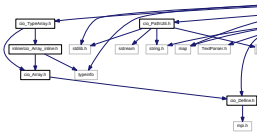
cio_DFI_BOV.C で定義されています。

7.11 cio_DFI_BOV.h

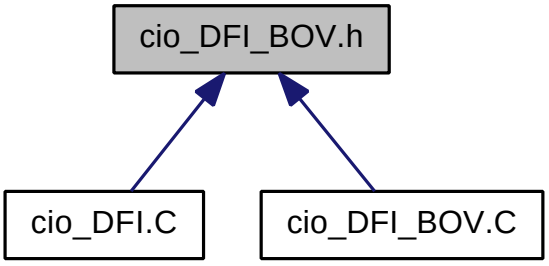
cio_DFI_BOV Class Header

```
#include "cio_DFI.h"
```

cio_DFI_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 cio_DFI_BOV

7.11.1 説明

cio_DFI_BOV Class Header

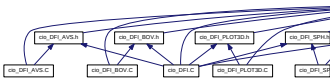
作者

kero

cio_DFI_BOV.h で定義されています。

7.12 cio_DFI_inline.h

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 CIO_INLINE inline

7.12.1 マクロ定義

7.12.1.1 #define CIO_INLINE inline

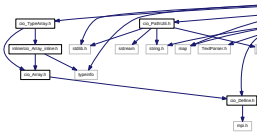
cio_DFI_inline.h の 23 行で定義されています。

7.13 cio_DFI_PLOT3D.C

cio_DFI_PLOT3D Class

```
#include "cio_DFI.h"
#include "cio_DFI_PLOT3D.h"
```

cio_DFI_PLOT3D.C のインクルード依存関係図



7.13.1 説明

cio_DFI_PLOT3D Class

作者

kero

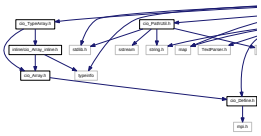
cio_DFI_PLOT3D.C で定義されています。

7.14 cio_DFI_PLOT3D.h

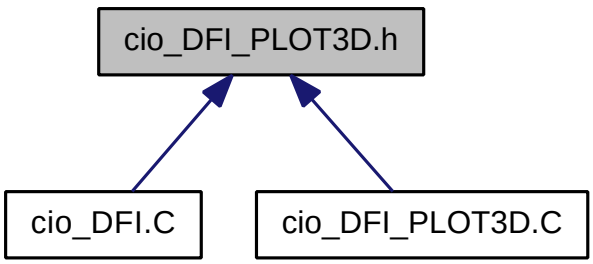
cio_DFI_PLOT3D Class Header

```
#include "cio_DFI.h"
#include "inline/cio_Plot3d_inline.h"
```

cio_DFI_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 cio_DFI_PLOT3D

7.14.1 説明

cio_DFI_PLOT3D Class Header

作者

kero

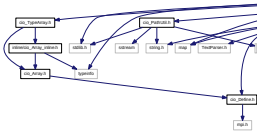
cio_DFI_PLOT3D.h で定義されています。

7.15 cio_DFI_Read.C

cio_DFI Class

```
#include "cio_DFI.h"
```

cio_DFI_Read.C のインクルード依存関係図



7.15.1 説明

cio_DFI Class

作者

kero

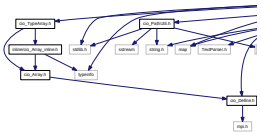
cio_DFI_Read.C で定義されています。

7.16 cio_DFI_SPH.C

cio_DFI_SPH Class

```
#include "cio_DFI.h"
#include "cio_DFI_SPH.h"
```

cio_DFI_SPH.C のインクルード依存関係図



7.16.1 説明

cio_DFI_SPH Class

作者

kero

cio_DFI_SPH.C で定義されています。

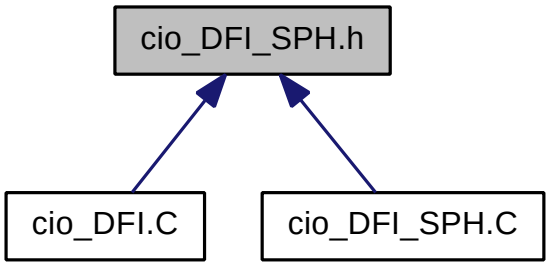
7.17 cio_DFI_SPH.h

cio_DFI_SPH Class Header

```
#include "cio_DFI.h"
```

cio_DFI_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 cio_DFI_SPH

7.17.1 説明

cio_DFI_SPH Class Header

作者

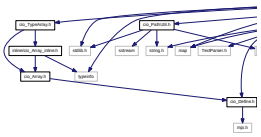
kero

cio_DFI_SPH.h で定義されています。

7.18 cio_DFI_VTK.C

cio_DFI_VTK Class

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



7.18.1 説明

cio_DFI_VTK Class

作者

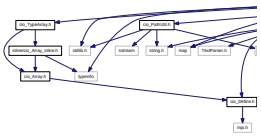
kero

cio_DFI_VTK.C で定義されています。

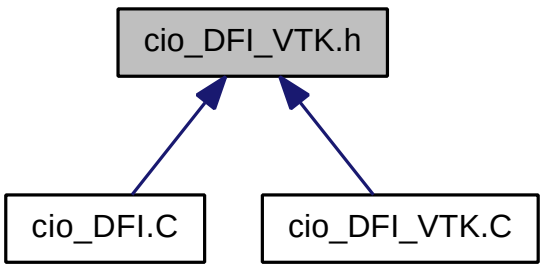
7.19 cio_DFI_VTK.h

cio_DFI_VTK Class Header

```
#include "cio_DFI.h"
```
cio_DFI_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 cio_DFI_VTK

7.19.1 説明

cio_DFI_VTK Class Header

作者

kero

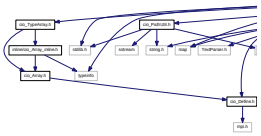
cio_DFI_VTK.h で定義されています。

7.20 cio_DFI_Write.C

cio_DFI Class

```
#include "cio_DFI.h"
```

cio_DFI_Write.C のインクルード依存関係図



7.20.1 説明

cio_DFI Class

作者

kero

cio_DFI_Write.C で定義されています。

7.21 cio_Domain.C

cio_Domain Class

```
#include "cio_DFI.h"
#include <unistd.h>
```

cio_Domain.C のインクルード依存関係図

7.21.1 説明

cio_Domain Class

作者

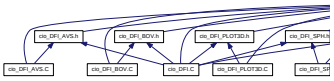
kero

cio_Domain.C で定義されています。

7.22 cio_Domain.h

cio_Domain Class Header

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 cio_Domain

7.22.1 説明

cio_Domain Class Header

作者

kero

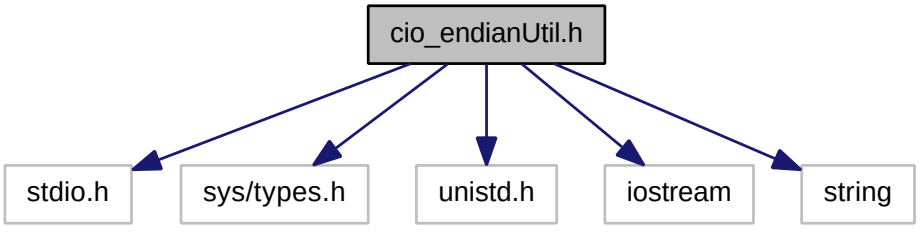
cio_Domain.h で定義されています。

7.23 cio_endianUtil.h

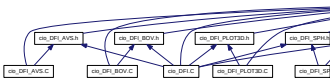
エンディアンユーティリティマクロ・関数ファイル

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <iostream>
#include <string>
```

cio_endianUtil.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 CIO_INLINE inline
- #define BSWAP_X_16(x)
- #define BSWAP16(x)
- #define BSWAP_X_32(x)
- #define BSWAP32(x)
- #define BSWAP_X_64(x)
- #define BSWAP64(x)
- #define SBSWAPVEC(a, n)
- #define BSWAPVEC(a, n)
- #define DBSWAPVEC(a, n)

7.23.1 説明

エンディアンユーティリティマクロ・関数ファイル

作者

kero

cio_endianUtil.h で定義されています。

7.23.2 マクロ定義

7.23.2.1 #define BSWAP16(*x*)

値:

```
{ \
    register unsigned short& _x_v = (unsigned short&)(x); \
    _x_v = BSWAP_X_16(_x_v);}
```

cio_endianUtil.h の 46 行で定義されています。

7.23.2.2 #define BSWAP32(*x*)

値:

```
{register unsigned int& _x_v = (unsigned int&)(x); \
     _x_v = BSWAP_X_32(_x_v);}
```

cio_endianUtil.h の 70 行で定義されています。

参照元 cio_DFI_SPH::read_averaged(), cio_DFI_SPH::read_Datarecord(), と cio_DFI_SPH::read_Header-Record().

7.23.2.3 #define BSWAP64(x)

値:

```
{register unsigned long long& _x_v = (unsigned long long&)(x); \
     _x_v = BSWAP_X_64(_x_v);}
```

cio_endianUtil.h の 104 行で定義されています。

参照元 cio_DFI_SPH::read_averaged(), と cio_DFI_SPH::read_HeaderRecord().

7.23.2.4 #define BSWAP_X_16(x)

値:

```
( (((x) & 0xff00) >>  8) \
    | (((x) & 0x00ff) <<  8) )
```

cio_endianUtil.h の 43 行で定義されています。

7.23.2.5 #define BSWAP_X_32(x)

値:

```
( (((x) & 0xff000000) >> 24) \
    | (((x) & 0x00ff0000) >>  8) \
    | (((x) & 0x0000ff00) <<  8) \
    | (((x) & 0x000000ff) << 24) )
```

cio_endianUtil.h の 65 行で定義されています。

7.23.2.6 #define BSWAP_X_64(x)

値:

```
( (((x) & 0xff00000000000000ull) >> 56) \
    | (((x) & 0x00ff000000000000ull) >> 40) \
    | (((x) & 0x0000ff0000000000ull) >> 24) \
    | (((x) & 0x000000ff00000000ull) >>  8) \
    | (((x) & 0x00000000ff000000ull) <<  8) \
    | (((x) & 0x0000000000ff0000ull) << 24) \
    | (((x) & 0x000000000000ff00ull) << 40) \
    | (((x) & 0x00000000000000ffull) << 56) )
```

cio_endianUtil.h の 95 行で定義されています。

7.23.2.7 #define BSWAPVEC(a, n)

値:

```
do{\
    for(register unsigned int _i=0;_i<(n);_i++){BSWAP32(a[_i]);}\
  }while(0)
```

cio_endianUtil.h の 139 行で定義されています。

参照元 cio_Process::ReadActiveSubdomainFile(), cio_TypeArray< T >::readBinary(), と cio_DFI_VTK::write_-DataRecord().

7.23.2.8 #define CIO_INLINE inline

cio_endianUtil.h の 28 行で定義されています。

7.23.2.9 #define DBSWAPVEC(*a*, *n*)

値:

```
do{\
    for(register unsigned int _i=0;_i<(n);_i++){BSWAP64(a[_i]);}\
  }while(0)
```

cio_endianUtil.h の 156 行で定義されています。

参照元 cio_TypeArray< T >::readBinary(), と cio_DFI_VTK::write_DataRecord().

7.23.2.10 #define SBSWAPVEC(*a*, *n*)

値:

```
do{\
    for(register unsigned int _i=0;_i<(n);_i++){BSWAP16(a[_i]);}\
  }while(0)
```

cio_endianUtil.h の 121 行で定義されています。

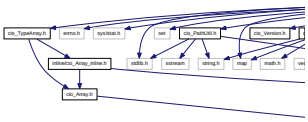
参照元 cio_TypeArray< T >::readBinary().

7.24 cio_FileInfo.C

cio_FileInfo Class

```
#include "cio_DFI.h"
#include <unistd.h>
```

cio_FileInfo.C のインクルード依存関係図



7.24.1 説明

cio_FileInfo Class

作者

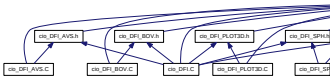
kero

cio_FileInfo.C で定義されています。

7.25 cio_FileInfo.h

cio_FileInfo Class Header

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 cio_FileInfo

7.25.1 説明

cio_FileInfo Class Header

作者

kero

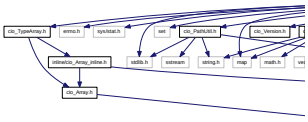
cio_FileInfo.h で定義されています。

7.26 cio_FilePath.C

cio_FilePath Class

```
#include "cio_DFI.h"
#include <unistd.h>
```

cio_FilePath.C のインクルード依存関係図



7.26.1 説明

cio_FilePath Class

作者

kero

cio_FilePath.C で定義されています。

7.27 cio_FilePath.h

cio_FilePath Class Header

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 cio_FilePath

7.27.1 説明

cio_FilePath Class Header

作者

kero

cio_FilePath.h で定義されています。

7.28 cio_interp_ijkn.h

関数

- !CIOlib Cartesian Input Output
  library !Copyright (c) 2013 Advanced Institute for Computational Science

7.28.1 関数

7.28.1.1 ! CIOlib Cartesian Input Output library !Copyright (c)

7.29 cio_interp_nijk.h

関数

- !CIOlib Cartesian Input Output
  library !Copyright (c) 2013 Advanced Institute for Computational Science

7.29.1 関数

7.29.1.1 ! CIOlib Cartesian Input Output library !Copyright (c)

7.30 cio_MPI.C

cio_MPI Class

```
#include "cio_DFI.h"
#include <unistd.h>
```

cio_MPI.C のインクルード依存関係図

7.30.1 説明

cio_MPI Class

作者

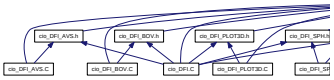
kero

cio_MPI.C で定義されています。

7.31 cio_MPI.h

cio_MPI Class Header

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 cio_MPI

7.31.1 説明

cio_MPI Class Header

作者

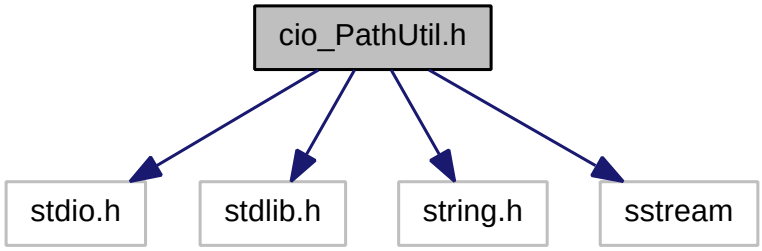
kero

cio_MPI.h で定義されています。

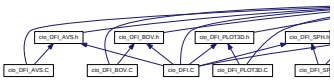
7.32 cio_PathUtil.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sstream>
```

cio_PathUtil.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。



ネームスペース

- CIO

Constant Groups

- CIO

マクロ定義

- #define MAXPATHLEN 512

関数

- char CIO::cioPath_getDelimChar ()
- std::string CIO::cioPath_getDelimString ()
- bool CIO::cioPath_hasDrive (const std::string &path)
- std::string CIO::vfvPath_emitDrive (std::string &path)
- bool CIO::cioPath_isAbsolute (const std::string &path)
- std::string CIO::cioPath_DirName (const std::string &path, const char dc=cioPath_getDelimChar())
- std::string CIO::cioPath_FileName (const std::string &path, const std::string &addext=std::string(""), const char dc=cioPath_getDelimChar())
- std::string CIO::cioPath_ConnectPath (std::string dirName, std::string fname)
- std::string CIO::ExtractPathWithoutExt (const std::string &fn)

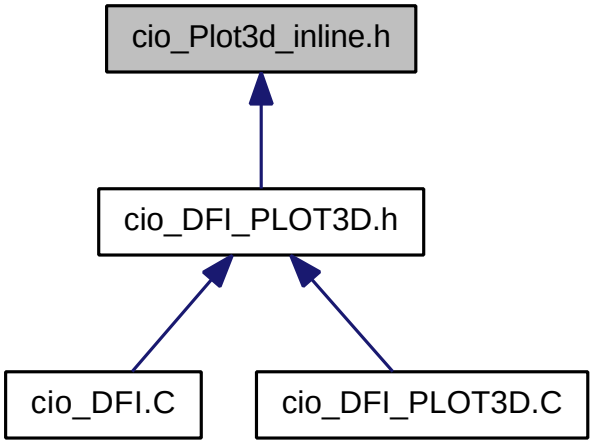
7.32.1 マクロ定義

7.32.1.1 #define MAXPATHLEN 512

cio_PathUtil.h の 17 行で定義されています。

7.33 cio_Plot3d_inline.h

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 CIO_INLINE inline

7.33.1 マクロ定義

7.33.1.1 #define CIO_INLINE inline

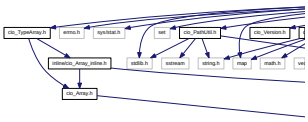
cio_Plot3d_inline.h の 23 行で定義されています。

7.34 cio_Process.C

cio_Rank & cio_Process Class

```
#include "cio_DFI.h"
#include <unistd.h>
```

cio_Process.C のインクルード依存関係図



7.34.1 説明

cio_Rank & cio_Process Class

作者

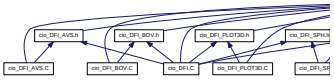
kero

cio_Process.C で定義されています。

7.35 cio_Process.h

cio_RANK & cio_Process Class Header

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 cio_Rank
- class cio_Process

7.35.1 説明

cio_RANK & cio_Process Class Header

作者

kero

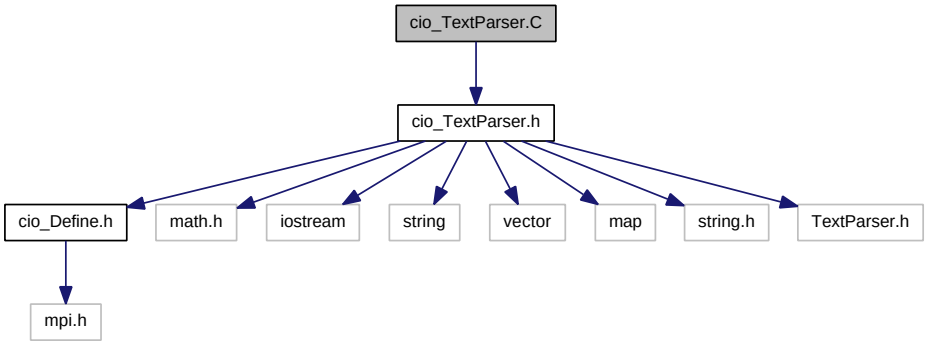
cio_Process.h で定義されています。

7.36 cio_TextParser.C

TextParser Control class.

```
#include "cio_TextParser.h"
```

cio_TextParser.C のインクルード依存関係図



7.36.1 説明

TextParser Control class.

作者

kero

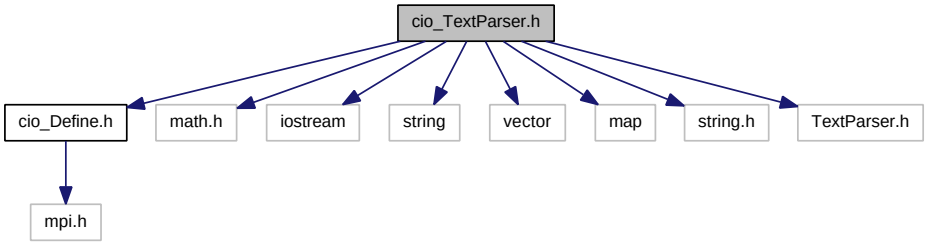
cio_TextParser.C で定義されています。

7.37 cio_TextParser.h

TextParser Control class Header.

```
#include "cio_Define.h"
#include <math.h>
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include "string.h"
#include "TextParser.h"
```

cio_TextParser.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 cio_TextParser

7.37.1 説明

TextParser Control class Header.

作者

kero

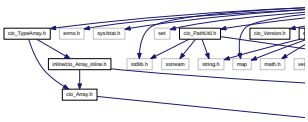
cio_TextParser.h で定義されています。

7.38 cio_TimeSlice.C

cio_Slice Class

```
#include "cio_DFI.h"
#include <unistd.h>
```

cio_TimeSlice.C のインクルード依存関係図



7.38.1 説明

cio_Slice Class

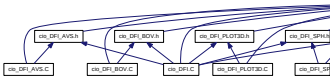
作者

kero

cio_TimeSlice.C で定義されています。

7.39 cio_TimeSlice.h

cio_Slice & cio_TimeSliceClass Header
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 cio_Slice
- class cio_TimeSlice

7.39.1 説明

cio_Slice & cio_TimeSliceClass Header

作者

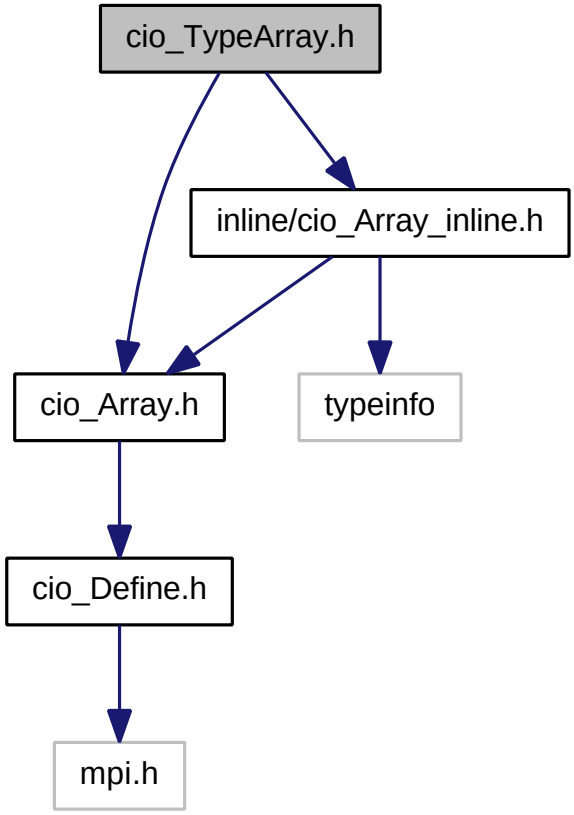
kero

cio_TimeSlice.h で定義されています。

7.40 cio_TypeArray.h

```
#include "cio_Array.h"
#include "inline/cio_Array_inline.h"
```

cio_TypeArray.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

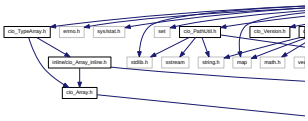
- class cio_TypeArray< T >

7.41 cio_Unit.C

cio_Unit Class

```
#include "cio_DFI.h"
#include <unistd.h>
```

cio_Unit.C のインクルード依存関係図



7.41.1 説明

cio_Unit Class

作者

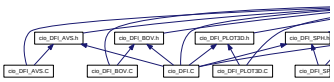
kero

cio_Unit.C で定義されています。

7.42 cio_Unit.h

cio_UnitElem & cio_Unit Class Header

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 cio_UnitElem
- class cio_Unit

7.42.1 説明

cio_UnitElem & cio_Unit Class Header

作者

kero

cio_Unit.h で定義されています。

7.43 cio_Version.h

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 CIO_VERSION_NO "1.4.4"
- #define CIO_REVISION "20131107_2300"

7.43.1 説明

CIO バージョン情報のヘッダーファイル

cio_Version.h で定義されています。

7.43.2 マクロ定義

7.43.2.1 #define CIO_REVISION "20131107_2300"

CIO ライブラリのリビジョン

cio_Version.h の 21 行で定義されています。

7.43.2.2 #define CIO_VERSION_NO "1.4.4"

CIO ライブラリのバージョン

cio_Version.h の 18 行で定義されています。

参照元 cio_DFI::getVersionInfo().

7.44 mpi_stubs.h

マクロ定義

- #define MPI_COMM_WORLD 0
- #define MPI_INT 1
- #define MPI_CHAR 2
- #define MPI_SUCCESS true

型定義

- typedef int MPI_Comm
- typedef int MPI_Datatype

関数

- bool MPI_Init (int *argc, char ***argv)
- int MPI_Comm_rank (MPI_Comm comm, int *rank)
- int MPI_Comm_size (MPI_Comm comm, int *size)
- int MPI_Allgather (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)
- int MPI_Gather (void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)

7.44.1 マクロ定義

7.44.1.1 #define MPI_CHAR 2

mpi_stubs.h の 22 行で定義されています。

参照元 cio_DFI::WriteProcDfiFile().

7.44.1.2 #define MPI_COMM_WORLD 0

mpi_stubs.h の 20 行で定義されています。

参照元 cio_DFI::WriteProcDfiFile().

7.44.1.3 #define MPI_INT 1

mpi_stubs.h の 21 行で定義されています。

参照元 cio_DFI::cio_Create_dfiProcessInfo().

7.44.1.4 #define MPI_SUCCESS true

mpi_stubs.h の 24 行で定義されています。

7.44.2 型定義

7.44.2.1 typedef int MPI_Comm

mpi_stubs.h の 18 行で定義されています。

7.44.2.2 typedef int MPI_Datatype

mpi_stubs.h の 19 行で定義されています。

7.44.3 関数

7.44.3.1 int MPI_Allgather (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, MPI_Comm *comm*) [inline]

mpi_stubs.h の 40 行で定義されています。

```
43 {
44   return 0;
45 }
```

7.44.3.2 int MPI_Comm_rank (MPI_Comm *comm*, int * *rank*) [inline]

mpi_stubs.h の 28 行で定義されています。

参照元 cio_DFI::cio_Create_dfiProcessInfo(), cio_DFI::ReadInit(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

```
29 {
30   *rank = 0;
31   return 0;
32 }
```

7.44.3.3 int MPI_Comm_size (MPI_Comm *comm*, int * *size*) [inline]

mpi_stubs.h の 34 行で定義されています。

参照元 cio_DFI::cio_Create_dfiProcessInfo(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

```
35 {
36   *size = 1;
37   return 0;
38 }
```

7.44.3.4 int MPI_Gather (void * *sendbuf,* int *sendcnt,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcnt,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*) [inline]

mpi_stubs.h の 47 行で定義されています。

参照元 cio_DFI::cio_Create_dfiProcessInfo(), と cio_DFI::WriteProcDfiFile().

```
50 {
51   return 0;
52 }
```

7.44.3.5 bool MPI_Init (int * *argc,* char *** *argv*) [inline]

mpi_stubs.h の 26 行で定義されています。

```
26 { return true; }
```

Index