Cartesian Input/Output Library

1.3.6

作成： Doxygen 1.8.4

Thu Oct 3 2013 13:54:52

Contents

1 ネームスペース索引 **1**

1.1 ネームスペース一覧 1

2 階層索引 **3**

2.1 クラス階層 3

3 構成索引 **5**

3.1 構成 5

4 ファイル索引 **7**

4.1 ファイル一覧 7

5 ネームスペース **9**

5.1 ネームスペース CIO 9

5.1.1 説明 10

5.1.2 列挙型 10

5.1.2.1 E_CIO_ARRAYSHAPE 10

5.1.2.2 E_CIO_DTYPE 11

5.1.2.3 E_CIO_ENDIANTYPE 11

5.1.2.4 E_CIO_ERRORCODE 12

5.1.2.5 E_CIO_FORMAT 15

5.1.2.6 E_CIO_ONOFF 15

5.1.2.7 E_CIO_READTYPE 15

5.1.3 関数 16

5.1.3.1 cioPath_ConnectPath 16

5.1.3.2 cioPath_DirName 16

5.1.3.3 cioPath_FileName 17

5.1.3.4 cioPath_getDelimChar 17

5.1.3.5 cioPath_getDelimString 18

5.1.3.6 cioPath_hasDrive 18

5.1.3.7 cioPath_isAbsolute 18

5.1.3.8 vfvPath_emitDrive 18

6 クラス **21**

6.1 クラス cio_ActiveSubDomain . . . 21
6.1.1 説明 . . . 21
6.1.2 コンストラクタとデストラクタ . . . 21
6.1.2.1 cio_ActiveSubDomain . . . 21
6.1.2.2 cio_ActiveSubDomain . . . 22
6.1.2.3 ∼cio_ActiveSubDomain . . . 23
6.1.3 関数 . . . 23
6.1.3.1 clear . . . 23
6.1.3.2 GetPos . . . 23
6.1.3.3 operator!= . . . 23
6.1.3.4 operator== . . . 24
6.1.3.5 SetPos . . . 24
6.1.4 変数 . . . 25
6.1.4.1 m_pos . . . 25
6.2 クラス cio_Array . . . 25
6.2.1 説明 . . . 28
6.2.2 コンストラクタとデストラクタ . . . 28
6.2.2.1 ∼cio_Array . . . 28
6.2.2.2 cio_Array . . . 28
6.2.2.3 cio_Array . . . 28
6.2.3 関数 . . . 29
6.2.3.1 _getArraySize . . . 29
6.2.3.2 _getArraySizeInt . . . 29
6.2.3.3 copyArray . . . 29
6.2.3.4 copyArray . . . 30
6.2.3.5 getArrayLength . . . 30
6.2.3.6 getArrayShape . . . 30
6.2.3.7 getArrayShapeString . . . 30
6.2.3.8 getArraySize . . . 30
6.2.3.9 getArraySizeInt . . . 31
6.2.3.10 getData . . . 31
6.2.3.11 getDataType . . . 32
6.2.3.12 getDataTypeString . . . 32
6.2.3.13 getGc . . . 33
6.2.3.14 getGcInt . . . 33
6.2.3.15 getHeadIndex . . . 33
6.2.3.16 getNcomp . . . 33
6.2.3.17 getNcompInt . . . 34
6.2.3.18 getTailIndex . . . 34

6.2.3.19 instanceArray . . . 34
6.2.3.20 instanceArray . . . 35
6.2.3.21 instanceArray . . . 35
6.2.3.22 instanceArray . . . 35
6.2.3.23 instanceArray . . . 36
6.2.3.24 instanceArray . . . 36
6.2.3.25 instanceArray . . . 36
6.2.3.26 instanceArray . . . 36
6.2.3.27 instanceArray . . . 37
6.2.3.28 instanceArray . . . 37
6.2.3.29 instanceArray . . . 37
6.2.3.30 instanceArray . . . 37
6.2.3.31 interp_coarse . . . 38
6.2.3.32 readBinary . . . 39
6.2.3.33 setHeadIndex . . . 39
6.2.3.34 writeBinary . . . 39
6.2.4 変数 . . . 39
6.2.4.1 m_dtype . . . 39
6.2.4.2 m_gc . . . 40
6.2.4.3 m_gcl . . . 40
6.2.4.4 m_gcl . . . 40
6.2.4.5 m_headIndex . . . 40
6.2.4.6 m_ncomp . . . 40
6.2.4.7 m_ncompl . . . 40
6.2.4.8 m_shape . . . 40
6.2.4.9 m_sz . . . 40
6.2.4.10 m_Sz . . . 41
6.2.4.11 m_szl . . . 41
6.2.4.12 m_Szl . . . 41
6.2.4.13 m_tailIndex . . . 41
6.3 クラス cio_DFI . . . 41
6.3.1 説明 . . . 46
6.3.2 コンストラクタとデストラクタ . . . 47
6.3.2.1 cio_DFI . . . 47
6.3.2.2 ~cio_DFI . . . 47
6.3.3 関数 . . . 47
6.3.3.1 AddUnit . . . 47
6.3.3.2 CheckReadRank . . . 47
6.3.3.3 CheckReadType . . . 48
6.3.3.4 cio_Create_dfiProcessInfo . . . 49

6.3.3.5 ConvDatatypeE2S . . . 50
6.3.3.6 ConvDatatypeS2E . . . 50
6.3.3.7 CreateReadStartEnd . . . 51
6.3.3.8 Generate_DFI_Name . . . 52
6.3.3.9 Generate_Directory_Path . . . 52
6.3.3.10 Generate_FieldFileName . . . 53
6.3.3.11 get_cio_Datasize . . . 54
6.3.3.12 GetArrayShape . . . 54
6.3.3.13 GetArrayShapeString . . . 55
6.3.3.14 GetcioDomain . . . 55
6.3.3.15 GetcioFileInfo . . . 55
6.3.3.16 GetcioFilePath . . . 55
6.3.3.17 GetcioMPI . . . 56
6.3.3.18 GetcioProcess . . . 56
6.3.3.19 GetcioTimeSlice . . . 56
6.3.3.20 GetcioUnit . . . 57
6.3.3.21 getComponentVariable . . . 57
6.3.3.22 GetDataType . . . 57
6.3.3.23 GetDataTypeString . . . 57
6.3.3.24 GetDFIGlobalDivision . . . 58
6.3.3.25 GetDFIGlobalVoxel . . . 58
6.3.3.26 getMinMax . . . 58
6.3.3.27 GetNumComponent . . . 59
6.3.3.28 GetUnit . . . 59
6.3.3.29 GetUnitElem . . . 59
6.3.3.30 getVectorMinMax . . . 60
6.3.3.31 getVersionInfo . . . 60
6.3.3.32 MakeDirectory . . . 60
6.3.3.33 MakeDirectoryPath . . . 61
6.3.3.34 MakeDirectorySub . . . 61
6.3.3.35 normalizeBaseTime . . . 62
6.3.3.36 normalizeDelteT . . . 62
6.3.3.37 normalizeIntervalTime . . . 62
6.3.3.38 normalizeLastTime . . . 63
6.3.3.39 normalizeStartTime . . . 63
6.3.3.40 normalizeTime . . . 63
6.3.3.41 read_averaged . . . 63
6.3.3.42 read_Datarecord . . . 64
6.3.3.43 read_HeaderRecord . . . 64
6.3.3.44 ReadData . . . 64

6.3.3.45 ReadData . . . 65
6.3.3.46 ReadData . . . 65
6.3.3.47 ReadData . . . 67
6.3.3.48 ReadData . . . 67
6.3.3.49 ReadFieldData . . . 69
6.3.3.50 ReadInit . . . 71
6.3.3.51 setComponentVariable . . . 74
6.3.3.52 setIntervalStep . . . 75
6.3.3.53 setIntervalTime . . . 75
6.3.3.54 SetTimeSliceFlag . . . 76
6.3.3.55 write_averaged . . . 77
6.3.3.56 write_DataRecord . . . 77
6.3.3.57 write_HeaderRecord . . . 77
6.3.3.58 WriteData . . . 78
6.3.3.59 WriteData . . . 78
6.3.3.60 WriteData . . . 80
6.3.3.61 WriteFieldData . . . 81
6.3.3.62 WriteIndexDfiFile . . . 82
6.3.3.63 WriteInit . . . 83
6.3.3.64 WriteInit . . . 84
6.3.3.65 WriteProcDfiFile . . . 86
6.3.3.66 WriteProcDfiFile . . . 86
6.3.4 変数 . . . 88
6.3.4.1 DFI_Domain . . . 88
6.3.4.2 DFI_Finfo . . . 88
6.3.4.3 DFI_Fpath . . . 88
6.3.4.4 DFI_MPI . . . 88
6.3.4.5 DFI_Process . . . 88
6.3.4.6 DFI_TimeSlice . . . 89
6.3.4.7 DFI_Unit . . . 89
6.3.4.8 m_comm . . . 89
6.3.4.9 m_directoryPath . . . 89
6.3.4.10 m_indexDfiName . . . 89
6.3.4.11 m_intervalMngr . . . 89
6.3.4.12 m_RankID . . . 89
6.3.4.13 m_read_type . . . 90
6.3.4.14 m_readRankList . . . 90
6.4 クラス cio_DFI_BOV . . . 90
6.4.1 説明 . . . 92
6.4.2 コンストラクタとデストラクタ . . . 92

6.4.2.1 cio_DFI_BOV . . . 92
6.4.2.2 cio_DFI_BOV . . . 92
6.4.2.3 ∼cio_DFI_BOV . . . 92
6.4.3 関数 . . . 93
6.4.3.1 read_averaged . . . 93
6.4.3.2 read_Datarecord . . . 94
6.4.3.3 read_HeaderRecord . . . 95
6.4.3.4 write_averaged . . . 95
6.4.3.5 write_DataRecord . . . 96
6.4.3.6 write_HeaderRecord . . . 96
6.5 クラス cio_DFI_SPH . . . 97
6.5.1 説明 . . . 99
6.5.2 列挙型 . . . 99
6.5.2.1 DataDims . . . 99
6.5.2.2 RealType . . . 99
6.5.3 コンストラクタとデストラクタ . . . 99
6.5.3.1 cio_DFI_SPH . . . 99
6.5.3.2 cio_DFI_SPH . . . 100
6.5.3.3 ∼cio_DFI_SPH . . . 101
6.5.4 関数 . . . 101
6.5.4.1 read_averaged . . . 101
6.5.4.2 read_Datarecord . . . 102
6.5.4.3 read_HeaderRecord . . . 103
6.5.4.4 write_averaged . . . 106
6.5.4.5 write_DataRecord . . . 107
6.5.4.6 write_HeaderRecord . . . 107
6.6 クラス cio_Domain . . . 109
6.6.1 説明 . . . 110
6.6.2 コンストラクタとデストラクタ . . . 110
6.6.2.1 cio_Domain . . . 110
6.6.2.2 cio_Domain . . . 110
6.6.2.3 ∼cio_Domain . . . 111
6.6.3 関数 . . . 111
6.6.3.1 Read . . . 111
6.6.3.2 Write . . . 112
6.6.4 変数 . . . 113
6.6.4.1 ActiveSubdomainFile . . . 113
6.6.4.2 GlobalDivision . . . 113
6.6.4.3 GlobalOrigin . . . 113
6.6.4.4 GlobalRegion . . . 114

6.6.4.5 GlobalVoxel . . . 114
6.7 クラス cio_FileInfo . . . 114
6.7.1 説明 . . . 115
6.7.2 コンストラクタとデストラクタ . . . 115
6.7.2.1 cio_FileInfo . . . 115
6.7.2.2 cio_FileInfo . . . 115
6.7.2.3 ∼cio_FileInfo . . . 116
6.7.3 関数 . . . 116
6.7.3.1 getComponentVariable . . . 116
6.7.3.2 Read . . . 117
6.7.3.3 setComponentVariable . . . 120
6.7.3.4 Write . . . 120
6.7.4 変数 . . . 121
6.7.4.1 ArrayShape . . . 121
6.7.4.2 Component . . . 122
6.7.4.3 ComponentVariable . . . 122
6.7.4.4 DataType . . . 122
6.7.4.5 DirectoryPath . . . 122
6.7.4.6 Endian . . . 122
6.7.4.7 FileFormat . . . 122
6.7.4.8 GuideCell . . . 123
6.7.4.9 Prefix . . . 123
6.7.4.10 TimeSliceDirFlag . . . 123
6.8 クラス cio_FilePath . . . 123
6.8.1 説明 . . . 123
6.8.2 コンストラクタとデストラクタ . . . 124
6.8.2.1 cio_FilePath . . . 124
6.8.2.2 cio_FilePath . . . 124
6.8.2.3 ∼cio_FilePath . . . 124
6.8.3 関数 . . . 124
6.8.3.1 Read . . . 124
6.8.3.2 Write . . . 125
6.8.4 変数 . . . 125
6.8.4.1 ProcDFIFile . . . 125
6.9 クラス cio_Interval_Mngr . . . 126
6.9.1 説明 . . . 127
6.9.2 列挙型 . . . 127
6.9.2.1 type_IO_spec . . . 127
6.9.3 コンストラクタとデストラクタ . . . 128
6.9.3.1 cio_Interval_Mngr . . . 128

6.9.3.2 ~cio_Interval_Mngr . . . 128
6.9.4 関数 . . . 128
6.9.4.1 calcNextStep . . . 128
6.9.4.2 calcNextTime . . . 129
6.9.4.3 dmod . . . 129
6.9.4.4 getIntervalStep . . . 129
6.9.4.5 getIntervalTime . . . 129
6.9.4.6 getMode . . . 129
6.9.4.7 getStartStep . . . 130
6.9.4.8 getStartTime . . . 130
6.9.4.9 initTrigger . . . 130
6.9.4.10 isLastStep . . . 131
6.9.4.11 isLastTime . . . 131
6.9.4.12 isStarted . . . 131
6.9.4.13 isTriggered . . . 131
6.9.4.14 normalizeBaseTime . . . 133
6.9.4.15 normalizeDelteT . . . 133
6.9.4.16 normalizeIntervalTime . . . 133
6.9.4.17 normalizeLastTime . . . 133
6.9.4.18 normalizeStartTime . . . 133
6.9.4.19 normalizeTime . . . 134
6.9.4.20 setInterval . . . 134
6.9.4.21 setLast . . . 134
6.9.4.22 setMode . . . 135
6.9.4.23 setStart . . . 135
6.9.5 変数 . . . 135
6.9.5.1 m_base_step . . . 135
6.9.5.2 m_base_time . . . 135
6.9.5.3 m_dt . . . 135
6.9.5.4 m_intvl_step . . . 136
6.9.5.5 m_intvl_time . . . 136
6.9.5.6 m_last_step . . . 136
6.9.5.7 m_last_time . . . 136
6.9.5.8 m_mode . . . 136
6.9.5.9 m_start_step . . . 136
6.9.5.10 m_start_time . . . 136
6.10 クラス cio_MPI . . . 137
6.10.1 説明 . . . 137
6.10.2 コンストラクタとデストラクタ . . . 137
6.10.2.1 cio_MPI . . . 137

6.10.2.2 cio_MPI . . . 137
6.10.2.3 ~cio_MPI . . . 138
6.10.3 関数 . . . 138
6.10.3.1 Read . . . 138
6.10.3.2 Write . . . 139
6.10.4 変数 . . . 139
6.10.4.1 NumberOfGroup . . . 139
6.10.4.2 NumberOfRank . . . 139
6.11 クラス cio_Process . . . 139
6.11.1 説明 . . . 141
6.11.2 型定義 . . . 141
6.11.2.1 headT . . . 141
6.11.3 コンストラクタとデストラクタ . . . 141
6.11.3.1 cio_Process . . . 141
6.11.3.2 ~cio_Process . . . 141
6.11.4 関数 . . . 141
6.11.4.1 CheckReadRank . . . 141
6.11.4.2 CheckStartEnd . . . 142
6.11.4.3 CreateHeadMap . . . 143
6.11.4.4 CreateHeadMap . . . 144
6.11.4.5 CreateRankList . . . 144
6.11.4.6 CreateRankList . . . 145
6.11.4.7 CreateRankMap . . . 146
6.11.4.8 CreateRankMap . . . 147
6.11.4.9 CreateSubDomainInfo . . . 148
6.11.4.10 isMatchEndianSbdmMagick . . . 148
6.11.4.11 Read . . . 149
6.11.4.12 ReadActiveSubdomainFile . . . 150
6.11.4.13 Write . . . 151
6.11.5 変数 . . . 151
6.11.5.1 m_rankMap . . . 152
6.11.5.2 RankList . . . 152
6.12 クラス cio_Rank . . . 152
6.12.1 説明 . . . 152
6.12.2 コンストラクタとデストラクタ . . . 153
6.12.2.1 cio_Rank . . . 153
6.12.2.2 ~cio_Rank . . . 153
6.12.3 関数 . . . 153
6.12.3.1 Read . . . 153
6.12.3.2 Write . . . 154

6.12.4 変数 . . . 155
6.12.4.1 HeadIndex . . . 155
6.12.4.2 HostName . . . 155
6.12.4.3 RankID . . . 155
6.12.4.4 TailIndex . . . 155
6.12.4.5 VoxelSize . . . 155
6.13 クラス cio_Slice . . . 156
6.13.1 説明 . . . 156
6.13.2 コンストラクタとデストラクタ . . . 156
6.13.2.1 cio_Slice . . . 156
6.13.2.2 ∼cio_Slice . . . 157
6.13.3 関数 . . . 157
6.13.3.1 Read . . . 157
6.13.3.2 Write . . . 159
6.13.4 変数 . . . 159
6.13.4.1 AveragedStep . . . 159
6.13.4.2 AveragedTime . . . 160
6.13.4.3 avr_mode . . . 160
6.13.4.4 Max . . . 160
6.13.4.5 Min . . . 160
6.13.4.6 step . . . 160
6.13.4.7 time . . . 160
6.13.4.8 VectorMax . . . 160
6.13.4.9 VectorMin . . . 160
6.14 クラス cio_TextParser . . . 161
6.14.1 説明 . . . 161
6.14.2 コンストラクタとデストラクタ . . . 162
6.14.2.1 cio_TextParser . . . 162
6.14.2.2 ∼cio_TextParser . . . 162
6.14.3 関数 . . . 162
6.14.3.1 chkLabel . . . 162
6.14.3.2 chkNode . . . 163
6.14.3.3 countLabels . . . 164
6.14.3.4 GetNodeStr . . . 165
6.14.3.5 getTPinstance . . . 166
6.14.3.6 GetValue . . . 166
6.14.3.7 GetValue . . . 167
6.14.3.8 GetValue . . . 168
6.14.3.9 GetVector . . . 168
6.14.3.10 GetVector . . . 169

6.14.3.11 GetVector . . . 170
6.14.3.12 readTPfile . . . 170
6.14.3.13 remove . . . 171
6.14.4 変数 . . . 171
6.14.4.1 tp . . . 171
6.15 クラス cio_TimeSlice . . . 171
6.15.1 説明 . . . 172
6.15.2 コンストラクタとデストラクタ . . . 172
6.15.2.1 cio_TimeSlice . . . 172
6.15.2.2 ∼cio_TimeSlice . . . 172
6.15.3 関数 . . . 172
6.15.3.1 AddSlice . . . 172
6.15.3.2 getMinMax . . . 173
6.15.3.3 getVectorMinMax . . . 174
6.15.3.4 Read . . . 174
6.15.3.5 Write . . . 175
6.15.4 変数 . . . 176
6.15.4.1 SliceList . . . 176
6.16 クラス テンプレート cio_TypeArray< T > . . . 176
6.16.1 説明 . . . 178
6.16.2 コンストラクタとデストラクタ . . . 178
6.16.2.1 cio_TypeArray . . . 178
6.16.2.2 cio_TypeArray . . . 178
6.16.2.3 ∼cio_TypeArray . . . 178
6.16.2.4 cio_TypeArray . . . 179
6.16.3 関数 . . . 179
6.16.3.1 _val . . . 179
6.16.3.2 _val . . . 179
6.16.3.3 copyArray . . . 179
6.16.3.4 copyArray . . . 180
6.16.3.5 getData . . . 181
6.16.3.6 hval . . . 181
6.16.3.7 hval . . . 181
6.16.3.8 readBinary . . . 181
6.16.3.9 val . . . 182
6.16.3.10 val . . . 182
6.16.3.11 writeBinary . . . 182
6.16.4 変数 . . . 182
6.16.4.1 m_data . . . 182
6.16.4.2 m_outptr . . . 183

6.17 クラス cio_Unit . . . 183
6.17.1 説明 . . . 183
6.17.2 コンストラクタとデストラクタ . . . 183
6.17.2.1 cio_Unit . . . 183
6.17.2.2 ～cio_Unit . . . 184
6.17.3 関数 . . . 184
6.17.3.1 GetUnit . . . 184
6.17.3.2 GetUnitElem . . . 184
6.17.3.3 Read . . . 185
6.17.3.4 Write . . . 186
6.17.4 変数 . . . 186
6.17.4.1 UnitList . . . 186
6.18 クラス cio_UnitElem . . . 187
6.18.1 説明 . . . 187
6.18.2 コンストラクタとデストラクタ . . . 187
6.18.2.1 cio_UnitElem . . . 187
6.18.2.2 cio_UnitElem . . . 188
6.18.2.3 ～cio_UnitElem . . . 188
6.18.3 関数 . . . 188
6.18.3.1 Read . . . 188
6.18.3.2 Write . . . 189
6.18.4 変数 . . . 189
6.18.4.1 BsetDiff . . . 189
6.18.4.2 difference . . . 189
6.18.4.3 Name . . . 190
6.18.4.4 reference . . . 190
6.18.4.5 Unit . . . 190

7 ファイル **191**

7.1 cio_ActiveSubDomain.C . . . 191
7.1.1 説明 . . . 191
7.2 cio_ActiveSubDomain.h . . . 191
7.3 cio_Array.h . . . 192
7.3.1 関数 . . . 193
7.3.1.1 cio_interp_ijkn_r4_ . . . 193
7.3.1.2 cio_interp_ijkn_r8_ . . . 193
7.3.1.3 cio_interp_nijk_r4_ . . . 193
7.3.1.4 cio_interp_nijk_r8_ . . . 193
7.4 cio_Array_inline.h . . . 193
7.4.1 マクロ定義 . . . 194

7.4.1.1 CIO_INLINE . . . 194
7.4.1.2 CIO_MEMFUN . . . 194
7.5 cio_Define.h . . . 194
7.5.1 説明 . . . 197
7.5.2 マクロ定義 . . . 197
7.5.2.1 _CIO_IDX_IJ . . . 197
7.5.2.2 _CIO_IDX_IJK . . . 197
7.5.2.3 _CIO_IDX_IJKN . . . 199
7.5.2.4 _CIO_IDX_NIJ . . . 199
7.5.2.5 _CIO_IDX_NIJK . . . 200
7.5.2.6 _CIO_TAB_STR . . . 200
7.5.2.7 _CIO_WRITE_TAB . . . 200
7.5.2.8 D_CIO_BIG . . . 201
7.5.2.9 D_CIO_EXT_BOV . . . 201
7.5.2.10 D_CIO_EXT_SPH . . . 201
7.5.2.11 D_CIO_FLOAT32 . . . 201
7.5.2.12 D_CIO_FLOAT64 . . . 201
7.5.2.13 D_CIO_IJNK . . . 201
7.5.2.14 D_CIO_INT16 . . . 201
7.5.2.15 D_CIO_INT32 . . . 201
7.5.2.16 D_CIO_INT64 . . . 202
7.5.2.17 D_CIO_INT8 . . . 202
7.5.2.18 D_CIO_LITTLE . . . 202
7.5.2.19 D_CIO_NIJK . . . 202
7.5.2.20 D_CIO_OFF . . . 202
7.5.2.21 D_CIO_ON . . . 202
7.5.2.22 D_CIO_UINT16 . . . 202
7.5.2.23 D_CIO_UINT32 . . . 202
7.5.2.24 D_CIO_UINT64 . . . 202
7.5.2.25 D_CIO_UINT8 . . . 202
7.6 cio_DFI.C . . . 203
7.6.1 説明 . . . 203
7.7 cio_DFI.h . . . 203
7.7.1 説明 . . . 204
7.8 cio_DFI_BOV.C . . . 204
7.8.1 説明 . . . 204
7.9 cio_DFI_BOV.h . . . 205
7.9.1 説明 . . . 205
7.10 cio_DFI_inline.h . . . 205
7.10.1 マクロ定義 . . . 206

7.10.1.1 CIO_INLINE . . . 206
7.11 cio_DFI_Read.C . . . 206
7.11.1 説明 . . . 206
7.12 cio_DFI_SPH.C . . . 206
7.12.1 説明 . . . 206
7.13 cio_DFI_SPH.h . . . 207
7.13.1 説明 . . . 207
7.14 cio_DFI_Write.C . . . 207
7.14.1 説明 . . . 208
7.15 cio_Domain.C . . . 208
7.15.1 説明 . . . 208
7.16 cio_Domain.h . . . 208
7.16.1 説明 . . . 209
7.17 cio_endianUtil.h . . . 209
7.17.1 説明 . . . 210
7.17.2 マクロ定義 . . . 210
7.17.2.1 BSWAP16 . . . 210
7.17.2.2 BSWAP32 . . . 210
7.17.2.3 BSWAP64 . . . 210
7.17.2.4 BSWAP_X_16 . . . 211
7.17.2.5 BSWAP_X_32 . . . 211
7.17.2.6 BSWAP_X_64 . . . 211
7.17.2.7 BSWAPVEC . . . 211
7.17.2.8 CIO_INLINE . . . 211
7.17.2.9 DBSWAPVEC . . . 211
7.17.2.10 SBSWAPVEC . . . 212
7.18 cio_FileInfo.C . . . 212
7.18.1 説明 . . . 212
7.19 cio_FileInfo.h . . . 212
7.19.1 説明 . . . 213
7.20 cio_FilePath.C . . . 213
7.20.1 説明 . . . 213
7.21 cio_FilePath.h . . . 213
7.21.1 説明 . . . 214
7.22 cio_interp_ijkn.h . . . 214
7.23 cio_interp_nijk.h . . . 214
7.24 cio_Interval_Mngr.h . . . 214
7.25 cio_MPI.C . . . 214
7.25.1 説明 . . . 214
7.26 cio_MPI.h . . . 215

7.26.1 説明 . . . 215
7.27 cio_PathUtil.h . . . 215
7.27.1 マクロ定義 . . . 216
7.27.1.1 MAXPATHLEN . . . 216
7.28 cio_Process.C . . . 216
7.28.1 説明 . . . 216
7.29 cio_Process.h . . . 217
7.29.1 説明 . . . 217
7.30 cio_TextParser.C . . . 217
7.30.1 説明 . . . 217
7.31 cio_TextParser.h . . . 218
7.31.1 説明 . . . 218
7.32 cio_TimeSlice.C . . . 218
7.32.1 説明 . . . 219
7.33 cio_TimeSlice.h . . . 219
7.33.1 説明 . . . 219
7.34 cio_TypeArray.h . . . 219
7.35 cio_Unit.C . . . 220
7.35.1 説明 . . . 221
7.36 cio_Unit.h . . . 221
7.36.1 説明 . . . 221
7.37 cio_Version.h . . . 221
7.37.1 説明 . . . 222
7.37.2 マクロ定義 . . . 222
7.37.2.1 CIO_REVISION . . . 222
7.37.2.2 CIO_VERSION_NO . . . 222
7.38 mpi_stubs.h . . . 222
7.38.1 マクロ定義 . . . 222
7.38.1.1 MPI_CHAR . . . 222
7.38.1.2 MPI_COMM_WORLD . . . 223
7.38.1.3 MPI_INT . . . 223
7.38.1.4 MPI_SUCCESS . . . 223
7.38.2 型定義 . . . 223
7.38.2.1 MPI_Comm . . . 223
7.38.2.2 MPI_Datatype . . . 223
7.38.3 関数 . . . 223
7.38.3.1 MPI_Allgather . . . 223
7.38.3.2 MPI_Comm_rank . . . 223
7.38.3.3 MPI_Comm_size . . . 223
7.38.3.4 MPI_Gather . . . 224

7.38.3.5 MPI_Init 224

索引 225

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。

CIO 9

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。

cio_ActiveSubDomain 21
cio_Array 25
  cio_TypeArray< T > 176
cio_DFI 41
  cio_DFI_BOV 90
  cio_DFI_SPH 97
cio_Domain 109
cio_FileInfo 114
cio_FilePath 123
cio_Interval_Mngr 126
cio_MPI 137
cio_Process 139
cio_Rank 152
cio_Slice 156
cio_TextParser 161
cio_TimeSlice 171
cio_Unit 183
cio_UnitElem 187

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。

cio_ActiveSubDomain 21
cio_Array 25
cio_DFI 41
cio_DFI_BOV 90
cio_DFI_SPH 97
cio_Domain 109
cio_FileInfo 114
cio_FilePath 123
cio_Interval_Mngr 126
cio_MPI 137
cio_Process 139
cio_Rank 152
cio_Slice 156
cio_TextParser 161
cio_TimeSlice 171
cio_TypeArray< T > 176
cio_Unit 183
cio_UnitElem 187

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。

cio_ActiveSubDomain.C
　　Cio_ActiveSubDomain class 関数 . . . 191
cio_ActiveSubDomain.h . . . 191
cio_Array.h . . . 192
cio_Array_inline.h . . . 193
cio_Define.h
　　CIO の定義マクロ記述ヘッダーファイル . . . 194
cio_DFI.C
　　Cio_DFI Class . . . 203
cio_DFI.h
　　Cio_DFI Class Header . . . 203
cio_DFI_BOV.C
　　Cio_DFI_BOV Class . . . 204
cio_DFI_BOV.h
　　Cio_DFI_BOV Class Header . . . 205
cio_DFI_inline.h . . . 205
cio_DFI_Read.C
　　Cio_DFI Class . . . 206
cio_DFI_SPH.C
　　Cio_DFI_SPH Class . . . 206
cio_DFI_SPH.h
　　Cio_DFI_SPH Class Header . . . 207
cio_DFI_Write.C
　　Cio_DFI Class . . . 207
cio_Domain.C
　　Cio_Domain Class . . . 208
cio_Domain.h
　　Cio_Domain Class Header . . . 208
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 . . . 209
cio_FileInfo.C
　　Cio_FileInfo Class . . . 212
cio_FileInfo.h
　　Cio_FileInfo Class Header . . . 212
cio_FilePath.C
　　Cio_FilePath Class . . . 213
cio_FilePath.h
　　Cio_FilePath Class Header . . . 213

cio_interp_ijkn.h . . . 214
cio_interp_nijk.h . . . 214
cio_Interval_Mngr.h . . . 214
cio_MPI.C
Cio_MPI Class . . . 214
cio_MPI.h
Cio_MPI Class Header . . . 215
cio_PathUtil.h . . . 215
cio_Process.C
Cio_Rank & cio_Process Class . . . 216
cio_Process.h
Cio_RANK & cio_Process Class Header . . . 217
cio_TextParser.C
TextParser Control class . . . 217
cio_TextParser.h
TextParser Control class Header . . . 218
cio_TimeSlice.C
Cio_Slice Class . . . 218
cio_TimeSlice.h
Cio_Slice & cio_TimeSliceClass Header . . . 219
cio_TypeArray.h . . . 219
cio_Unit.C
Cio_Unit Class . . . 220
cio_Unit.h
Cio_UnitElem & cio_Unit Class Header . . . 221
cio_Version.h . . . 221
mpi_stubs.h . . . 222

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 }
- 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_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)

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 86 行で定義されています。

```
87   {
88     E_CIO_ARRAYSHAPE_UNKNOWN=-1,
89     E_CIO_IJKN=0,
90     E_CIO_NIJK
91   };
```

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 70 行で定義されています。

```
71   {
72     E_CIO_DTYPE_UNKNOWN = 0,
73     E_CIO_INT8,
74     E_CIO_INT16,
75     E_CIO_INT32,
76     E_CIO_INT64,
77     E_CIO_UINT8,
78     E_CIO_UINT16,
79     E_CIO_UINT32,
80     E_CIO_UINT64,
81     E_CIO_FLOAT32,
82     E_CIO_FLOAT64
83   };
```

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 94 行で定義されています。

```
95   {
96     E_CIO_ENDIANTYPE_UNKNOWN=-1,
97     E_CIO_LITTLE=0,
98     E_CIO_BIG
99   };
```

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 112 行で定義されています。

```
113   {
114     E_CIO_SUCCESS                          = 1
115 ,   E_CIO_ERROR                            = -1
116 ,   E_CIO_ERROR_READ_DFI_GLOBALORIGIN      = 1000
117 ,   E_CIO_ERROR_READ_DFI_GLOBALREGION      = 1001
118 ,   E_CIO_ERROR_READ_DFI_GLOBALVOXEL       = 1002
119 ,   E_CIO_ERROR_READ_DFI_GLOBALDIVISION    = 1003
120 ,   E_CIO_ERROR_READ_DFI_DIRECTORYPATH     = 1004
121 ,   E_CIO_ERROR_READ_DFI_TIMESLICEDIRECTORY = 1005
122 ,   E_CIO_ERROR_READ_DFI_PREFIX            = 1006
123 ,   E_CIO_ERROR_READ_DFI_FILEFORMAT        = 1007
124 ,   E_CIO_ERROR_READ_DFI_GUIDECELL         = 1008
125 ,   E_CIO_ERROR_READ_DFI_DATATYPE          = 1009
126 ,   E_CIO_ERROR_READ_DFI_ENDIAN            = 1010
127 ,   E_CIO_ERROR_READ_DFI_ARRAYSHAPE        = 1011
128 ,   E_CIO_ERROR_READ_DFI_COMPONENT         = 1012
129 ,   E_CIO_ERROR_READ_DFI_FILEPATH_PROCESS  = 1013
130 ,   E_CIO_ERROR_READ_DFI_NO_RANK           = 1014
131 ,   E_CIO_ERROR_READ_DFI_ID                = 1015
132 ,   E_CIO_ERROR_READ_DFI_HOSTNAME          = 1016
133 ,   E_CIO_ERROR_READ_DFI_VOXELSIZE         = 1017
134 ,   E_CIO_ERROR_READ_DFI_HEADINDEX         = 1018
135 ,   E_CIO_ERROR_READ_DFI_TAILINDEX         = 1019
136 ,   E_CIO_ERROR_READ_DFI_NO_SLICE          = 1020
137 ,   E_CIO_ERROR_READ_DFI_STEP              = 1021
138 ,   E_CIO_ERROR_READ_DFI_TIME              = 1022
139 ,   E_CIO_ERROR_READ_DFI_NO_MINMAX         = 1023
140 ,   E_CIO_ERROR_READ_DFI_MIN               = 1024
141 ,   E_CIO_ERROR_READ_DFI_MAX               = 1025
142 ,   E_CIO_ERROR_READ_INDEXFILE_OPENERROR   = 1050
143 ,   E_CIO_ERROR_TEXTPARSER                 = 1051
144 ,   E_CIO_ERROR_READ_FILEINFO              = 1052
145 ,   E_CIO_ERROR_READ_FILEPATH              = 1053
146 ,   E_CIO_ERROR_READ_UNIT                  = 1054
147 ,   E_CIO_ERROR_READ_TIMESLICE             = 1055
148 ,   E_CIO_ERROR_READ_PROCFILE_OPENERROR    = 1056
149 ,   E_CIO_ERROR_READ_DOMAIN                = 1057
150 ,   E_CIO_ERROR_READ_MPI                   = 1058
151 ,   E_CIO_ERROR_READ_PROCESS               = 1059
152 ,   E_CIO_ERROR_READ_FIELDDATA_FILE        = 1900
153 ,   E_CIO_ERROR_READ_SPH_FILE              = 2000
154 ,   E_CIO_ERROR_READ_SPH_REC1              = 2001
155 ,   E_CIO_ERROR_READ_SPH_REC2              = 2002
156 ,   E_CIO_ERROR_READ_SPH_REC3              = 2003
157 ,   E_CIO_ERROR_READ_SPH_REC4              = 2004
158 ,   E_CIO_ERROR_READ_SPH_REC5              = 2005
159 ,   E_CIO_ERROR_READ_SPH_REC6              = 2006
160 ,   E_CIO_ERROR_READ_SPH_REC7              = 2007
161 ,   E_CIO_ERROR_UNMATCH_VOXELSIZE          = 2050
162 ,   E_CIO_ERROR_NOMATCH_ENDIAN             = 2051
163 ,   E_CIO_ERROR_READ_BOV_FILE              = 2100
164 ,   E_CIO_ERROR_READ_FIELD_HEADER_RECORD   = 2102
165 ,   E_CIO_ERROR_READ_FIELD_DATA_RECORD     = 2103
166 ,   E_CIO_ERROR_READ_FIELD_AVERAGED_RECORD = 2104
167 //,   E_CIO_ERROR_DATATYPE                  = 2500 ///< DataType error
168 ,   E_CIO_ERROR_MISMATCH_NP_SUBDOMAIN      = 3003
169 ,   E_CIO_ERROR_INVALID_DIVNUM             = 3011
170 ,   E_CIO_ERROR_OPEN_SBDM                  = 3012
171 ,   E_CIO_ERROR_READ_SBDM_HEADER           = 3013
172 ,   E_CIO_ERROR_READ_SBDM_FORMAT           = 3014
173 ,   E_CIO_ERROR_READ_SBDM_DIV              = 3015
174 ,   E_CIO_ERROR_READ_SBDM_CONTENTS         = 3016
175 ,   E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO        = 3017
176 ,   E_CIO_ERROR_MAKEDIRECTORY              = 3100
177 ,   E_CIO_ERROR_OPEN_FIELDDATA             = 3101
178 ,   E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD  = 3102
179 ,   E_CIO_ERROR_WRITE_FIELD_DATA_RECORD    = 3103
180 ,   E_CIO_ERROR_WRITE_FIELD_AVERAGED_RECORD = 3104
181 ,   E_CIO_ERROR_WRITE_SPH_REC1             = 3201
182 ,   E_CIO_ERROR_WRITE_SPH_REC2             = 3202
183 ,   E_CIO_ERROR_WRITE_SPH_REC3             = 3203
184 ,   E_CIO_ERROR_WRITE_SPH_REC4             = 3204
185 ,   E_CIO_ERROR_WRITE_SPH_REC5             = 3205
186 ,   E_CIO_ERROR_WRITE_SPH_REC6             = 3206
187 ,   E_CIO_ERROR_WRITE_SPH_REC7             = 3207
188 ,   E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY   = 3500
189 ,   E_CIO_ERROR_WRITE_PROCFILE_OPENERROR   = 3501
190 ,   E_CIO_ERROR_WRITE_DOMAIN               = 3502
191 ,   E_CIO_ERROR_WRITE_MPI                  = 3503
192 ,   E_CIO_ERROR_WRITE_PROCESS              = 3504
```

```
193 ,   E_CIO_ERROR_WRITE_RANKID                = 3505
194 ,   E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY   = 3510
195 ,   E_CIO_ERROR_WRITE_PREFIX_EMPTY          = 3511
196 ,   E_CIO_ERROR_WRITE_INDEXFILE_OPENERROR   = 3512
197 ,   E_CIO_ERROR_WRITE_FILEINFO              = 3513
198 ,   E_CIO_ERROR_WRITE_UNIT                  = 3514
199 ,   E_CIO_ERROR_WRITE_TIMESLICE            = 3515
200 ,   E_CIO_ERROR_WRITE_FILEPATH             = 3516
201 ,   E_CIO_WARN_GETUNIT                      = 4000
202   };
```

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

cio_Define.h の 55 行で定義されています。

```
56   {
57     E_CIO_FMT_UNKNOWN = -1,
58     E_CIO_FMT_SPH,
59     E_CIO_FMT_BOV
60   };
```

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 63 行で定義されています。

```
64   {
65     E_CIO_OFF = 0,
66     E_CIO_ON
67   };
```

5.1.2.7 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 102 行で定義されています。

```
103   {
104     E_CIO_SAMEDIV_SAMERES=1,
105     E_CIO_SAMEDIV_REFINEMENT,
106     E_CIO_DIFFDIV_SAMERES,
107     E_CIO_DIFFDIV_REFINEMENT,
108     E_CIO_READTYPE_UNKNOWN,
109   };
```

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(), 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::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().

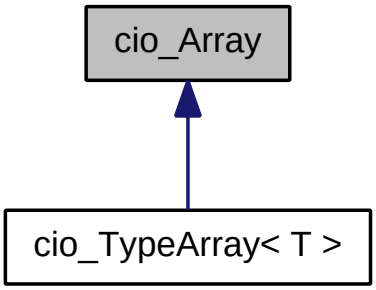
このクラスの説明は次のファイルから生成されました:

- 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 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***)*
- 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_ncompl
  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 14 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 51 行で定義されています。

```
52   {
53   }
```

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 350 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_gcl, m_headIndex, m_ncomp, m_shape, m_sz, m_Sz, と m_tailIndex.

```
351   {
352     m_dtype = CIO::E_CIO_DTYPE_UNKNOWN;
353     m_shape = CIO::E_CIO_ARRAYSHAPE_UNKNOWN;
354     m_sz[0] = m_sz[1] = m_sz[2] = 0;
355     m_Sz[0] = m_Sz[1] = m_Sz[2] = m_Sz[3] = 0;
356     m_gc = 0;
357     m_gcl[0] = m_gcl[1] = m_gcl[2] = m_gcl[3] = 0;
358     m_ncomp = 1;
359     m_headIndex[0] = m_headIndex[1] = m_headIndex[2] = m_headIndex[3] = 0;
360     m_tailIndex[0] = m_tailIndex[1] = m_tailIndex[2] = m_tailIndex[3] = 0;
361   }
```

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 364 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_gcl, m_ncomp, m_ncompI, m_shape, m_sz, m_Sz, m_szI, m_SzI, と setHeadIndex().

```
371   {
372     m_sz[0] = m_szI[0] = ix;
373     m_sz[1] = m_szI[1] = jx;
374     m_sz[2] = m_szI[2] = kx;
375
376     switch(shape)
377     {
378     case CIO::E_CIO_IJKN:
379       m_Sz[0] = m_SzI[0] = ix+2*gc;
380       m_Sz[1] = m_SzI[1] = jx+2*gc;
381       m_Sz[2] = m_SzI[2] = kx+2*gc;
382       m_Sz[3] = m_SzI[3] = ncomp;
383       m_gcl[0] = gc;
384       m_gcl[1] = gc;
385       m_gcl[2] = gc;
386       m_gcl[3] = 0;
387       break;
388     case CIO::E_CIO_NIJK:
389       m_Sz[0] = m_SzI[0] = ncomp;
390       m_Sz[1] = m_SzI[1] = ix+2*gc;
391       m_Sz[2] = m_SzI[2] = jx+2*gc;
392       m_Sz[3] = m_SzI[3] = kx+2*gc;
```

```
393       m_gcl[0] = 0;
394       m_gcl[1] = gc;
395       m_gcl[2] = gc;
396       m_gcl[3] = gc;
397     }
398
399     m_gc = m_gcI = gc;
400     m_ncomp = m_ncompI = ncomp;
401     m_dtype = dtype;
402     m_shape = shape;
403
404     int head[3]={0,0,0};
405     setHeadIndex(head);
406   }
```

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 267 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.

```
268   {
269     switch(m_shape)
270     {
271     case CIO::E_CIO_IJKN:
272       return m_Sz;
273       break;
274     case CIO::E_CIO_NIJK:
275       return m_Sz + 1;
276       break;
277     }
278     return NULL;
279   }
```

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 282 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.

```
283   {
284     switch(m_shape)
285     {
286     case CIO::E_CIO_IJKN:
287       return m_SzI;
288       break;
289     case CIO::E_CIO_NIJK:
290       return m_SzI + 1;
291       break;
292     }
293     return NULL;
294   }
```

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 size_t cio_Array::getArrayLength ( ) const [inline]

配列長を取得

cio_Array.h の 297 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().

```
298   {
299     size_t nw = 1;
300     for( int i=0;i<4;i++ )
301     {
302       nw *= m_Sz[i];
303     }
304     return nw;
305   }
```

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

配列形状の取得

cio_Array.h の 180 行で定義されています。

参照先 m_shape.

参照元 cio_TypeArray< T >::copyArray().

```
181   {
182     return m_shape;
183   }
```

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

配列形状文字列の取得

cio_Array.h の 186 行で定義されています。

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

```
187   {
188     switch(m_shape)
189     {
190     case CIO::E_CIO_IJKN:
191       return "IJKN";
192       break;
193     case CIO::E_CIO_NIJK:
194       return "NIJK";
195       break;
196     }
197     return "Unknown";
198   }
```

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

格子数を取得

cio_Array.h の 225 行で定義されています。

参照先 m_sz.

```
226   {
227     return m_sz;
228   }
```

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

格子数を取得 (int 版)

cio_Array.h の 231 行で定義されています。

参照先 m_szI.

```
232   {
233     return m_szI;
234   }
```

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

データポインタを取得

cio_Array_inline.h の 236 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 cio_TypeArray< T >::getData().

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

```
237 {
238   switch( m_dtype )
239   {
240   case CIO::E_CIO_INT8:
241     {
242       cio_TypeArray<char> *ptr = dynamic_cast<cio_TypeArray<char>*>(this);
243       return ptr->getData( extract );
244     }
245     break;
246   case CIO::E_CIO_INT16:
247     {
248       cio_TypeArray<short> *ptr = dynamic_cast<cio_TypeArray<short>*>(this);
249       return ptr->getData( extract );
250     }
251     break;
252   case CIO::E_CIO_INT32:
253     {
254       cio_TypeArray<int> *ptr = dynamic_cast<cio_TypeArray<int>*>(this);
255       return ptr->getData( extract );
256     }
257     break;
258   case CIO::E_CIO_INT64:
259     {
260       cio_TypeArray<long long> *ptr = dynamic_cast<cio_TypeArray<long long>*>(this);
261       return ptr->getData( extract );
262     }
263     break;
264   case CIO::E_CIO_UINT8:
265     {
266       cio_TypeArray<unsigned char> *ptr = dynamic_cast<
    cio_TypeArray<unsigned char>*>(this);
267       return ptr->getData( extract );
268     }
269     break;
270   case CIO::E_CIO_UINT16:
271     {
272       cio_TypeArray<unsigned short> *ptr = dynamic_cast<
    cio_TypeArray<unsigned short>*>(this);
273       return ptr->getData( extract );
274     }
275     break;
276   case CIO::E_CIO_UINT32:
277     {
278       cio_TypeArray<unsigned int> *ptr = dynamic_cast<
    cio_TypeArray<unsigned int>*>(this);
279       return ptr->getData( extract );
280     }
```

```
281     break;
282   case CIO::E_CIO_UINT64:
283     {
284       cio_TypeArray<unsigned long long> *ptr = dynamic_cast<
    cio_TypeArray<unsigned long long>*>(this);
285       return ptr->getData( extract );
286     }
287     break;
288   case CIO::E_CIO_FLOAT32:
289     {
290       cio_TypeArray<float> *ptr = dynamic_cast<cio_TypeArray<float>*>(this);
291       return ptr->getData( extract );
292     }
293     break;
294   case CIO::E_CIO_FLOAT64:
295     {
296       cio_TypeArray<double> *ptr = dynamic_cast<cio_TypeArray<double>*>(this);
297       return ptr->getData( extract );
298     }
299     break;
300   }
301
302   return NULL;
303 }
```

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

データタイプの取得

cio_Array.h の 135 行で定義されています。

参照先 m_dtype.

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

```
136   {
137     return m_dtype;
138   }
```

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

データタイプ文字列の取得

cio_Array.h の 141 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.

```
142   {
143     switch( m_dtype )
144     {
145     case CIO::E_CIO_INT8:
146       return "INT8";
147       break;
148     case CIO::E_CIO_INT16:
149       return "INT16";
150       break;
151     case CIO::E_CIO_INT32:
152       return "INT32";
153       break;
154     case CIO::E_CIO_INT64:
155       return "INT64";
156       break;
157     case CIO::E_CIO_UINT8:
158       return "UINT8";
159       break;
160     case CIO::E_CIO_UINT16:
161       return "UINT16";
162       break;
163     case CIO::E_CIO_UINT32:
164       return "UINT32";
165       break;
166     case CIO::E_CIO_UINT64:
167       return "UINT64";
```

```
168       break;
169     case CIO::E_CIO_FLOAT32:
170       return "FLOAT32";
171       break;
172     case CIO::E_CIO_FLOAT64:
173       return "FLOAT64";
174       break;
175     }
176     return "Unknown";
177   }
```

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

ガイドセル数を取得

cio_Array.h の 201 行で定義されています。

参照先 m_gc.

```
202   {
203     return m_gc;
204   }
```

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

ガイドセル数を取得 (int 版)

cio_Array.h の 207 行で定義されています。

参照先 m_gcI.

参照元 cio_TypeArray< T >::copyArray().

```
208   {
209     return m_gcI;
210   }
```

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

head インデクスを取得

cio_Array.h の 237 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().

```
238   {
239     switch(m_shape)
240     {
241     case CIO::E_CIO_IJKN:
242       return m_headIndex;
243       break;
244     case CIO::E_CIO_NIJK:
245       return m_headIndex + 1;
246       break;
247     }
248     return NULL;
249   }
```

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

成分数を取得

cio_Array.h の 213 行で定義されています。

参照先 m_ncomp.

参照元 cio_TypeArray< T >::copyArray().

```
214   {
215     return m_ncomp;
216   }
```

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

成分数を取得 (int 版)

cio_Array.h の 219 行で定義されています。

参照先 m_ncompI.

```
220   {
221     return m_ncompI;
222   }
```

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

tail インデクスを取得

cio_Array.h の 252 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().

```
253   {
254     switch(m_shape)
255     {
256     case CIO::E_CIO_IJKN:
257       return m_tailIndex;
258       break;
259     case CIO::E_CIO_NIJK:
260       return m_tailIndex + 1;
261       break;
262     }
263     return NULL;
264   }
```

6.2.3.19 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 22 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(), と cio_DFI::WriteData().

```
29 {
30   cio_Array *ptr = NULL;
31   switch( dtype )
32   {
33   case CIO::E_CIO_INT8:
34     ptr = new cio_TypeArray<char>(dtype,shape,ix,jx,kx,gc,ncomp);
35     break;
36   case CIO::E_CIO_INT16:
37     ptr = new cio_TypeArray<short>(dtype,shape,ix,jx,kx,gc,ncomp);
38     break;
39   case CIO::E_CIO_INT32:
```

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

6.2.3.20 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 85 行で定義されています。

```
90 {
91   return instanceArray(dtype,shape,sz[0],sz[1],sz[2],gc,ncomp);
92 }
```

6.2.3.21 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 96 行で定義されています。

```
103 {
104   return instanceArray(dtype,shape,size_t(ix),size_t(jx),size_t(kx),size_t(gc),size_t(ncomp));
105 }
```

6.2.3.22 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 109 行で定義されています。

```
114 {
115   return instanceArray(dtype,shape,size_t(sz[0]),size_t(sz[1]),size_t(sz[2]),size_t(gc),size_t(ncomp));
116 }
```

6.2.3.23 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

6.2.3.24 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

6.2.3.25 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

6.2.3.26 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 121 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.

```
128 {
129   cio_Array *ptr = NULL;
130   CIO::E_CIO_DTYPE dtype = CIO::E_CIO_DTYPE_UNKNOWN;
131
132   if( typeid(data) == typeid(char*) )
133   {
134     dtype = CIO::E_CIO_INT8;
135   }
136   else if( typeid(data) == typeid(short*) )
137   {
138     dtype = CIO::E_CIO_INT16;
139   }
140   else if( typeid(data) == typeid(int*) )
141   {
142     dtype = CIO::E_CIO_INT32;
143   }
144   else if( typeid(data) == typeid(long long*) )
145   {
146     dtype = CIO::E_CIO_INT64;
147   }
148   else if( typeid(data) == typeid(unsigned char*) )
149   {
150     dtype = CIO::E_CIO_UINT8;
151   }
152   else if( typeid(data) == typeid(unsigned short*) )
153   {
154     dtype = CIO::E_CIO_UINT16;
155   }
156   else if( typeid(data) == typeid(unsigned int*) )
157   {
158     dtype = CIO::E_CIO_UINT32;
159   }
160   else if( typeid(data) == typeid(unsigned long long*) )
161   {
162     dtype = CIO::E_CIO_UINT64;
163   }
164   else if( typeid(data) == typeid(float*) )
165   {
166     dtype = CIO::E_CIO_FLOAT32;
167   }
168   else if( typeid(data) == typeid(double*) )
```

```
169   {
170     dtype = CIO::E_CIO_FLOAT64;
171   }
172 
173   if( dtype != CIO::E_CIO_DTYPE_UNKNOWN )
174   {
175     ptr = new cio_TypeArray<T>(data,dtype,shape,ix,jx,kx,gc,ncomp);
176   }
177 
178 #ifdef _CIO_DEBUG
179   if( ptr )
180   {
181     printf("dtype = %d\n",(int)dtype);
182     printf("shape = %d\n",(int)shape);
183     printf("ixjxkx = %d %d %d\n",(int)ix,(int)jx,(int)kx);
184     printf("gc = %d\n",(int)gc);
185     printf("ncomp = %d\n",(int)ncomp);
186     size_t *m_Sz=ptr->m_Sz;
187     size_t *m_gcl=ptr->m_gcl;
188     printf("Sz  = %d %d %d %d\n",(int)m_Sz[0],(int)m_Sz[1],(int)m_Sz[2],(int)m_Sz[3]);
189     printf("gcl = %d %d %d %d\n",(int)m_gcl[0],(int)m_gcl[1],(int)m_gcl[2],(int)m_gcl[3]);
190   }
191 #endif
192 
193   return ptr;
194 }
```

6.2.3.27 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

6.2.3.28 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 199 行で定義されています。

```
204 {
205   return instanceArray(data,shape,sz[0],sz[1],sz[2],gc,ncomp);
206 }
```

6.2.3.29 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 211 行で定義されています。

```
218 {
219   return instanceArray(data,shape,size_t(ix),size_t(jx),size_t(kx),size_t(gc),size_t(ncomp));
220 }
```

6.2.3.30 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 225 行で定義されています。

```
230 {
231   return instanceArray(data,shape,size_t(sz[0]),size_t(sz[1]),size_t(sz[2]),size_t(gc),size_t(ncomp));
232 }
```

6.2.3.31 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 469 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().

```
470 {
471   err = 1;
472
473   // データタイプ
474   // 実数型のみ対応
475   CIO::E_CIO_DTYPE dtype = src->getDataType();
476   if( dtype != CIO::E_CIO_FLOAT32 && dtype != CIO::E_CIO_FLOAT64 )
477   {
478     err = -1;
479     return NULL;
480   }
481
482   // 配列形状
483   CIO::E_CIO_ARRAYSHAPE shape = src->getArrayShape();
484
485   // 成分数
486   // 成分数は 1 か 3 のみ対応
487   int ncomp = src->getNcomp();
488   if( ncomp != 1 && ncomp != 3 )
489   {
490     err = -1;
491     return NULL;
492   }
493
494   // その他の情報の取得
495   int gcS = src->getGc();
496   void *ptrS = src->getData();
497   const int *szS = src->getArraySizeInt();
498   const int *headS = src->getHeadIndex();
499   const int *tailS = src->getTailIndex();
500
501   // 密配列のインスタンス
502   int gcD = gcS*2;
503   int szD[3] = {szS[0]*2, szS[1]*2, szS[2]*2};
504   cio_Array *dst = cio_Array::instanceArray( dtype, shape, szD, gcD, ncomp );
505   void *ptrD = dst->getData();
506
507   // head インデクスのセット
508   int headD[3];
509   for( int i=0;i<3;i++ )
510   {
511     headD[i] = headS[i]*2;
512     if( !head0start )
513     {
514       headD[i] -= 1;
515     }
516   }
517   dst->setHeadIndex( headD );
518
519   // f90 コードのコール（配列形状、実数型毎）
520   if( shape == CIO::E_CIO_IJKN )
521   {
522     if( dtype == CIO::E_CIO_FLOAT32 )
523     {
524       cio_interp_ijkn_r4_(szS,&gcS,szD,&gcD,&ncomp,(float*)ptrS,(float*)ptrD);
525     }
526     else
527     {
528       cio_interp_ijkn_r8_(szS,&gcS,szD,&gcD,&ncomp,(double*)ptrS,(double*)ptrD);
529     }
530   }
531   else
532   {
533     if( dtype == CIO::E_CIO_FLOAT32 )
534     {
535       cio_interp_nijk_r4_(szS,&gcS,szD,&gcD,&ncomp,(float*)ptrS,(float*)ptrD);
536     }
537     else
538     {
539       cio_interp_nijk_r8_(szS,&gcS,szD,&gcD,&ncomp,(double*)ptrS,(double*)ptrD);
540     }
541   }
542
```

```
543   return dst;
544 }
```

6.2.3.32 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().

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

head/tail をセット

cio_Array.h の 308 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().

```
309   {
310     switch(m_shape)
311     {
312     case CIO::E_CIO_IJKN:
313       m_headIndex[0] = head[0];
314       m_headIndex[1] = head[1];
315       m_headIndex[2] = head[2];
316       m_headIndex[3] = 0;
317       m_tailIndex[0] = m_headIndex[0] + m_sz[0] - 1;
318       m_tailIndex[1] = m_headIndex[1] + m_sz[1] - 1;
319       m_tailIndex[2] = m_headIndex[2] + m_sz[2] - 1;
320       m_tailIndex[3] = 0;
321       break;
322     case CIO::E_CIO_NIJK:
323       m_headIndex[0] = 0;
324       m_headIndex[1] = head[0];
325       m_headIndex[2] = head[1];
326       m_headIndex[3] = head[2];
327       m_tailIndex[0] = 0;
328       m_tailIndex[1] = m_headIndex[1] + m_sz[0] - 1;
329       m_tailIndex[2] = m_headIndex[2] + m_sz[1] - 1;
330       m_tailIndex[3] = m_headIndex[3] + m_sz[2] - 1;
331     }
332   }
```

6.2.3.34 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(), 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 416 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 422 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 438 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 431 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 451 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 434 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_ncompl [protected]

成分数 (int)

cio_Array.h の 447 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 419 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 425 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 428 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 441 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 444 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 454 行で定義されています。

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

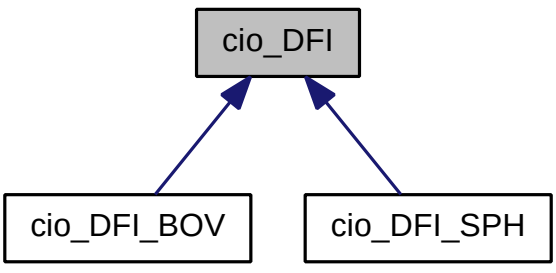
このクラスの説明は次のファイルから生成されました:

- cio_Array.h
- cio_Array_inline.h

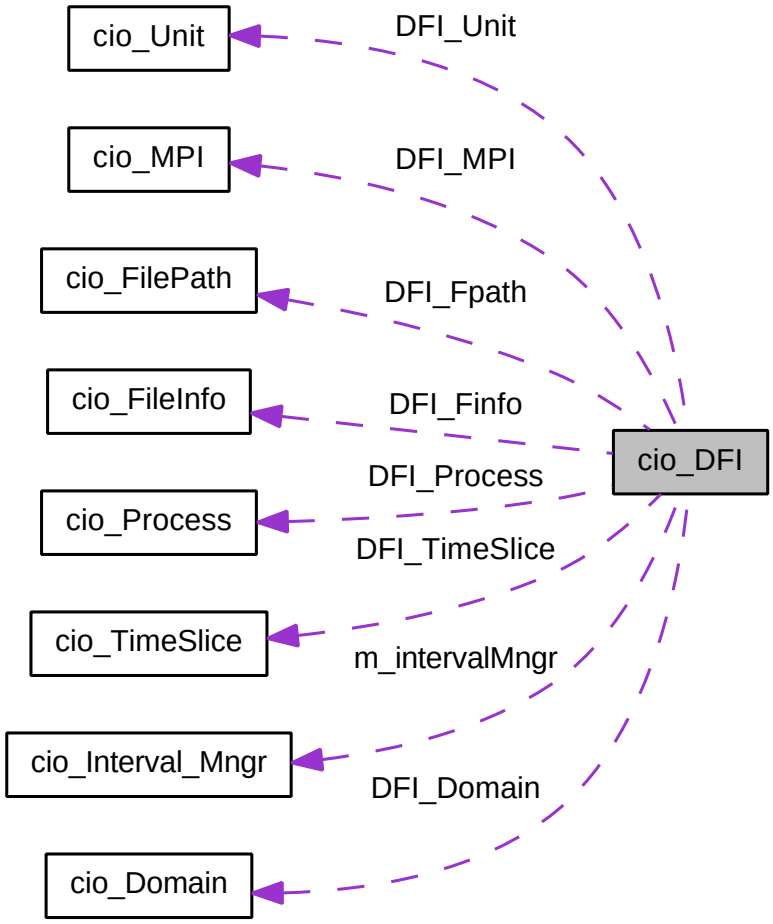
6.3 クラス cio_DFI

`#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
- const cio_Unit * GetcioUnit ()

  cio_Unit クラスのポインタを取得
- const cio_Domain * GetcioDomain ()

  cio_Domain クラスのポインタ取得
- const cio_MPI * GetcioMPI ()

  cio_MPI クラスのポインタ取得
- const cio_TimeSlice * GetcioTimeSlice ()

  cio_TimeSlice クラスのポインタ取得
- const cio_Process * GetcioProcess ()

  cio_Process クラスのポインタ取得
- 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

- 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 force=true, 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, bool force)

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

  proc DFI **ファイル出力コントロール** *(float)*
- CIO::E_CIO_ERRORCODE WriteProcDfiFile (const MPI_Comm comm, bool out_host=false, double *org=NULL)

  proc DFI **ファイル出力コントロール** *(double* **版***)*
- 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
- 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, bool force, const bool avr_mode, const unsigned step_avr, TimeAvrT time_avr)

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 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
- 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
- cio_Interval_Mngr m_intervalMngr

  インターバルマネージャー

6.3.1 説明

CIO main class

cio_DFI.h の 46 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 23 行で定義されています。

参照先 CIO::E_CIO_READTYPE_UNKNOWN, m_RankID, と m_read_type.

```
24 {
25
26  m_read_type = CIO::E_CIO_READTYPE_UNKNOWN;
27  m_RankID = 0;
28
29 }
```

6.3.2.2 cio_DFI::∼cio_DFI ( )

デストラクタ

cio_DFI.C の 34 行で定義されています。

```
35 {
36
37 }
```

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 850 行で定義されています。

参照先 DFI_Unit, と cio_Unit::UnitList.

```
855 {
856
858   cio_UnitElem unit =  cio_UnitElem(Name,Unit,reference,difference,BsetDiff);
859
861   DFI_Unit.UnitList.insert(map<std::string,cio_UnitElem>::value_type(Name,unit));
862
863 }
```

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 936 行で定義されています。

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

```
941 {
942
943   return DFI_Process.CheckReadRank(dfi_domain,head,tail,readflag,readRankList);
944
945 }
```

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 587 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().

```
591 {
592
593   bool isSameDiv=true;
594
595   //分割数チェック
596   for(int i=0; i<3; i++ ) {
597     if( DFI_GlobalDivision[i] != G_Div[i] ) {
598       isSameDiv = false;
599     }
600   }
601
602   if( isSameDiv ) {
603     if( G_voxel[0] == DFI_GlobalVoxel[0]   &&
604         G_voxel[1] == DFI_GlobalVoxel[1]   &&
605         G_voxel[2] == DFI_GlobalVoxel[2]   ) return CIO::E_CIO_SAMEDIV_SAMERES;
606
607     if( G_voxel[0] == DFI_GlobalVoxel[0]*2 &&
608         G_voxel[1] == DFI_GlobalVoxel[1]*2 &&
609         G_voxel[2] == DFI_GlobalVoxel[2]*2 ) return
    CIO::E_CIO_SAMEDIV_REFINEMENT;
610   } else {
```

```
611     if( G_voxel[0] == DFI_GlobalVoxel[0]   &&
612         G_voxel[1] == DFI_GlobalVoxel[1]   &&
613         G_voxel[2] == DFI_GlobalVoxel[2]   ) return CIO::E_CIO_DIFFDIV_SAMERES;
614
615     if( G_voxel[0] == DFI_GlobalVoxel[0]*2 &&
616         G_voxel[1] == DFI_GlobalVoxel[1]*2 &&
617         G_voxel[2] == DFI_GlobalVoxel[2]*2 ) return
      CIO::E_CIO_DIFFDIV_REFINEMENT;
618   }
619
620   return CIO::E_CIO_READTYPE_UNKNOWN;
621 }
```

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 534 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().

```
536 {
537
538   cio_Rank G_Rank;
539
540   int RankID;
541   MPI_Comm_rank( comm, &RankID );
542
543   int nrank;
544   MPI_Comm_size( comm, &nrank );
545
546   if( nrank > 1 ) {
547     int *headtail = NULL;
548     if( RankID == 0 ) {
549       headtail = new int[6*nrank];
550     }
551
552     int sbuff[6];
553     for(int i=0; i<3; i++) {
554       sbuff[i]   = DFI_Process.RankList[RankID].HeadIndex[i];
555       sbuff[i+3] = DFI_Process.RankList[RankID].TailIndex[i];
556     }
557
558     MPI_Gather(sbuff,6,MPI_INT,headtail,6,MPI_INT,0,comm);
559
560     if( RankID == 0 ) {
561       for(int i=0; i<nrank; i++) {
562         G_Rank.RankID=i;
563         for(int j=0; j<3; j++) {
564           G_Rank.HeadIndex[j]=headtail[i*6+j];
565           G_Rank.TailIndex[j]=headtail[i*6+j+3];
566           G_Rank.VoxelSize[j]=G_Rank.TailIndex[j]-G_Rank.HeadIndex[j]+1;
567         }
568         G_Process.RankList.push_back(G_Rank);
569       }
570     }
571
572     if ( RankID == 0 ) delete [] headtail;
573
574   } else {
575     G_Rank.RankID=0;
576     for(int i=0; i<3; i++) {
577       G_Rank.HeadIndex[i]=DFI_Process.RankList[0].HeadIndex[i];
578       G_Rank.TailIndex[i]=DFI_Process.RankList[0].TailIndex[i];
579       G_Rank.VoxelSize[i]=G_Rank.TailIndex[i]-G_Rank.HeadIndex[i]+1;
580     }
581     G_Process.RankList.push_back(G_Rank);
582   }
583 }
```

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 485 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().

```
486 {
487   if     ( Dtype == CIO::E_CIO_INT8    ) return D_CIO_INT8;
488   else if( Dtype == CIO::E_CIO_INT16   ) return D_CIO_INT16;
489   else if( Dtype == CIO::E_CIO_INT32   ) return D_CIO_INT32;
490   else if( Dtype == CIO::E_CIO_INT64   ) return D_CIO_INT64;
491   else if( Dtype == CIO::E_CIO_UINT8   ) return D_CIO_UINT8;
492   else if( Dtype == CIO::E_CIO_UINT16  ) return D_CIO_UINT16;
493   else if( Dtype == CIO::E_CIO_UINT32  ) return D_CIO_UINT32;
494   else if( Dtype == CIO::E_CIO_UINT64  ) return D_CIO_UINT64;
495   else if( Dtype == CIO::E_CIO_FLOAT32 ) return D_CIO_FLOAT32;
496   else if( Dtype == CIO::E_CIO_FLOAT64 ) return D_CIO_FLOAT64;
497   else return "dummy";
498
499 }
```

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 466 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().

```
467 {
468
469   if     ( !strcasecmp(datatype.c_str(),"Int8"   ) ) return CIO::E_CIO_INT8;
470   else if( !strcasecmp(datatype.c_str(),"Int16"  ) ) return CIO::E_CIO_INT16;
471   else if( !strcasecmp(datatype.c_str(),"Int32"  ) ) return CIO::E_CIO_INT32;
472   else if( !strcasecmp(datatype.c_str(),"Int64"  ) ) return CIO::E_CIO_INT64;
473   else if( !strcasecmp(datatype.c_str(),"UInt8"  ) ) return CIO::E_CIO_UINT8;
474   else if( !strcasecmp(datatype.c_str(),"UInt16" ) ) return CIO::E_CIO_UINT16;
475   else if( !strcasecmp(datatype.c_str(),"UInt32" ) ) return CIO::E_CIO_UINT32;
476   else if( !strcasecmp(datatype.c_str(),"UInt64" ) ) return CIO::E_CIO_UINT64;
477   else if( !strcasecmp(datatype.c_str(),"Float32") ) return CIO::E_CIO_FLOAT32;
478   else if( !strcasecmp(datatype.c_str(),"Float64") ) return CIO::E_CIO_FLOAT64;
479
480   return CIO::E_CIO_DTYPE_UNKNOWN;
481 }
```

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 625 行で定義されています。

参照先 DFI_Domain, と cio_Domain::GlobalVoxel.

参照元 ReadData().

```
637 {
638
639   int src_head[3],src_tail[3],src_gc;
640   if( !isSame ) {
641   // 粗密のとき密に変換、ガイドセルは倍にする
642     src_gc = DFI_gc*2;
643     for(int i=0; i<3; i++) {
644       src_head[i]=DFI_head[i]*2-1;
645       src_tail[i]=DFI_tail[i]*2;
646     }
647   } else {
648   // 粗密でない時各値をコピー
649     src_gc = DFI_gc;
650     for(int i=0; i<3; i++) {
651       src_head[i]=DFI_head[i];
652       src_tail[i]=DFI_tail[i];
653     }
654   }
655
656 //スタート、エンドをセット
657   for(int i=0; i<3; i++) {
658     copy_sta[i] = max(head[i],src_head[i]);
659     copy_end[i] = min(tail[i],src_tail[i]);
660
661     //仮想セルが読込みの実セル内のときの処理（スタート）
662     if( copy_sta[i] == 1 ) {
663       copy_sta[i] -= min(gc,src_gc);
664     } else if( head[i]>src_head[i] ) {
665       copy_sta[i] = max(head[i]-gc,src_head[i]);
666     }
667
668     //仮想セルが読込みの実セル内のときの処理
669     if( ( isSame  && copy_end[i] == DFI_Domain.GlobalVoxel[i] )  ||
670         ( !isSame && copy_end[i] == DFI_Domain.GlobalVoxel[i]*2 ) ) {
671       copy_end[i] += min(gc,src_gc);
672     } else if( tail[i]<src_tail[i] ) {
673       copy_end[i] = min(tail[i]+gc,src_tail[i]);
674     }
675
676     //read satrt/end のセット
677     if( !isSame ) {
678       if( copy_sta[i]>0 ) read_sta[i] = (copy_sta[i]+1)/2;
679       else                read_sta[i] =  copy_sta[i]/2;
680
681       if( copy_end[i]>0 ) read_end[i] = (copy_end[i]+1)/2;
682       else                read_end[i] =  copy_end[i]/2;
683
684     } else {
685       read_sta[i] = copy_sta[i];
686       read_end[i] = copy_end[i];
```

```
687     }
688
689   }
690 }
```

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 825 行で定義されています。

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

```
826 {
827
828   // directory path
829   std::string dirName  = CIO::cioPath_DirName(prefix);
830
831   // file extension
832   std::string dfiname = CIO::cioPath_FileName(prefix,".dfi");
833
834   // filename
835   std::string fname = CIO::cioPath_ConnectPath( dirName, dfiname );
836
837 #if 0 // for debug
838   printf("prefix    =%s\n", prefix.c_str() );
839   printf("  dirName =%s\n", dirName.c_str() );
840   printf("  dfiname =%s\n", dfiname.c_str() );
841   printf("  fname   =%s\n", fname.c_str() );
842   printf("\n");
843 #endif
844
845   return fname;
846 }
```

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 796 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().

```
797 {
798
799   // dfi のパスと DirectoryPath を連結する関数
800   // ただし、絶対パスのときは dfi のパスは無視
801   // CIO::cioPath_isAbsolute が true のとき絶対パス
802   // DirectoryPath + TimeSliceDir
803   std::string path = m_directoryPath;
804   if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON )
805   {
806     //path = CIO::cioPath_ConnectPath(path, m_timeSliceDir);
```

```
807     path = CIO::cioPath_ConnectPath(path, "");
808   }
809
810   // absolute path
811   if( CIO::cioPath_isAbsolute(path) )
812   {
813     return path;
814   }
815
816   // relative path
817   std::string dfidir = CIO::cioPath_DirName(m_indexDfiName);
818   path = CIO::cioPath_ConnectPath(dfidir, path);
819   return path;
820
821 }
```

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

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 694 行で定義されています。

参照先 D_CIO_EXT_BOV, D_CIO_EXT_SPH, DFI_Finfo, cio_FileInfo::DirectoryPath, CIO::E_CIO_FMT_BOV, CIO::E_CIO_FMT_SPH, CIO::E_CIO_ON, cio_FileInfo::FileFormat, cio_FileInfo::Prefix, と cio_FileInfo::TimeSliceDirFlag.

参照元 ReadData(), と WriteData().

```
697 {
698
699   if( DFI_Finfo.DirectoryPath.empty() ) return NULL;
700   if( DFI_Finfo.Prefix.empty() ) return NULL;
701
702   std::string fmt;
703   if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_SPH ) {
704     fmt=D_CIO_EXT_SPH;
705   } else if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_BOV ) {
706     fmt=D_CIO_EXT_BOV;
707   }
708
709   int len = DFI_Finfo.DirectoryPath.size() + DFI_Finfo.Prefix.size() + fmt.size() + 25;
710   // id(6) + step(10) + 1(\0) + "_"(2) + "."(1)+"id"(2)
711   if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON ) len += 11;
712
713   char* tmp = new char[len];
714   memset(tmp, 0, sizeof(char)*len);
715
716   if( mio ) {
717     if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON ) {
718       sprintf(tmp, "%s/%010d/%s_%010d_id%06d.%s",DFI_Finfo.DirectoryPath.c_str(),step,
      DFI_Finfo.Prefix.c_str(),
719               step,RankID,fmt.c_str());
720     } else {
721       sprintf(tmp, "%s/%s_%010d_id%06d.%s",DFI_Finfo.DirectoryPath.c_str(),
      DFI_Finfo.Prefix.c_str(),
722               step,RankID,fmt.c_str());
723     }
724   } else {
725     if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON ) {
726       sprintf(tmp, "%s/%010d/%s_%010d.%s",DFI_Finfo.DirectoryPath.c_str(),step,
      DFI_Finfo.Prefix.c_str(),
727               step,fmt.c_str());
728     } else {
729       sprintf(tmp, "%s/%s_%010d.%s",DFI_Finfo.DirectoryPath.c_str(),DFI_Finfo.
      Prefix.c_str(),
```

```
730             step,fmt.c_str());
731     }
732   }
733
734   std::string fname(tmp);
735   if( tmp ) delete [] tmp;
736
737   return fname;
738 }
```

6.3.3.11 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 502 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(), と cio_DFI_SPH::write_DataRecord().

```
503 {
504
505   if     ( Dtype == CIO::E_CIO_INT8    ) return sizeof(char);
506   else if( Dtype == CIO::E_CIO_INT16   ) return sizeof(short);
507   else if( Dtype == CIO::E_CIO_INT32   ) return sizeof(int);
508   else if( Dtype == CIO::E_CIO_INT64   ) return sizeof(long long);
509   else if( Dtype == CIO::E_CIO_UINT8   ) return sizeof(unsigned char);
510   else if( Dtype == CIO::E_CIO_UINT16  ) return sizeof(unsigned short);
511   else if( Dtype == CIO::E_CIO_UINT32  ) return sizeof(unsigned int);
512   else if( Dtype == CIO::E_CIO_UINT64  ) return sizeof(unsigned long long);
513   else if( Dtype == CIO::E_CIO_FLOAT32 ) return sizeof(float);
514   else if( Dtype == CIO::E_CIO_FLOAT64 ) return sizeof(double);
515   else return 0;
516
517 }
```

6.3.3.12 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 438 行で定義されています。

参照先 cio_FileInfo::ArrayShape, と DFI_Finfo.

```
439 {
440   return (CIO::E_CIO_ARRAYSHAPE)DFI_Finfo.ArrayShape;
441 }
```

6.3.3.13 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 429 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.

```
430 {
431   if( DFI_Finfo.ArrayShape == CIO::E_CIO_IJKN ) return D_CIO_IJNK;
432   if( DFI_Finfo.ArrayShape == CIO::E_CIO_NIJK ) return D_CIO_NIJK;
433   return " ";
434 }
```

6.3.3.14 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 244 行で定義されています。

参照先 DFI_Domain.

```
245 {
246   return &DFI_Domain;
247 }
```

6.3.3.15 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 220 行で定義されています。

参照先 DFI_Finfo.

```
221 {
222   return &DFI_Finfo;
223 }
```

6.3.3.16 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 228 行で定義されています。

参照先 DFI_Fpath.

```
229 {
230   return &DFI_Fpath;
231 }
```

6.3.3.17 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 252 行で定義されています。

参照先 DFI_MPI.

```
253 {
254   return &DFI_MPI;
255 }
```

6.3.3.18 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 269 行で定義されています。

参照先 DFI_Process.

```
270 {
271   return &DFI_Process;
272 }
```

6.3.3.19 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 260 行で定義されています。

参照先 DFI_TimeSlice.

```
261 {
262   return &DFI_TimeSlice;
263 }
```

6.3.3.20 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 236 行で定義されています。

参照先 DFI_Unit.

```
237 {
238   return &DFI_Unit;
239 }
```

6.3.3.21 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 903 行で定義されています。

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

```
904 {
905
906   return DFI_Finfo.getComponentVariable(pcomp);
907
908 }
```

6.3.3.22 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 452 行で定義されています。

参照先 cio_FileInfo::DataType, と DFI_Finfo.

```
453 {
454   return (CIO::E_CIO_DTYPE)DFI_Finfo.DataType;
455 }
```

6.3.3.23 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 445 行で定義されています。

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

```
446 {
447   return ConvDatatypeE2S((CIO::E_CIO_DTYPE)DFI_Finfo.DataType);
448 }
```

6.3.3.24 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 528 行で定義されています。

参照先 DFI_Domain, と cio_Domain::GlobalDivision.

```
529 {
530   return DFI_Domain.GlobalDivision;
531 }
```

6.3.3.25 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 521 行で定義されています。

参照先 DFI_Domain, と cio_Domain::GlobalVoxel.

```
522 {
523   return DFI_Domain.GlobalVoxel;
524 }
```

6.3.3.26 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 923 行で定義されています。

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

```
927 {
928
929   return DFI_TimeSlice.getMinMax(step,compNo,min_value,max_value);
930
931 }
```

6.3.3.27 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 459 行で定義されています。

参照先 cio_FileInfo::Component, と DFI_Finfo.

```
460 {
461   return DFI_Finfo.Component;
462 }
```

6.3.3.28 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 875 行で定義されています。

参照先 DFI_Unit, と cio_Unit::GetUnit().

```
880 {
881   return DFI_Unit.GetUnit(Name, unit, ref, diff, bSetDiff);
882 }
```

6.3.3.29 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 867 行で定義されています。

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

```
869 {
870   return DFI_Unit.GetUnitElem(Name, unit);
871 }
```

6.3.3.30 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 912 行で定義されています。

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

```
915 {
916
917   return DFI_TimeSlice.getVectorMinMax(step,vec_min,vec_max);
918
919 }
```

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

バージョンを出力する

cio_DFI.h の 900 行で定義されています。

参照先 CIO_VERSION_NO.

```
901   {
902     std::string str(CIO_VERSION_NO);
903     return str;
904   }
```

6.3.3.32 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 743 行で定義されています。

参照先 MakeDirectorySub().

参照元 MakeDirectoryPath(), と WriteData().

```
744 {
745   int ret = MakeDirectorySub(path);
746   if( ret != 0 )
747   {
748      // 既存以外のエラー
749      if ( EEXIST != errno )
750      {
751         printf( "\tError(errno)=[%s]\n", strerror(errno) );
752         return 0;
753      }
754   }
755
756   // failed
757   return 1;
758 }
```

6.3.3.33 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 762 行で定義されています。

参照先 Generate_Directory_Path(), と MakeDirectory().

```
763 {
764   // DirectoryPath with TimeSlice
765   std::string path = Generate_Directory_Path();
766
767   return MakeDirectory(path);
768 }
```

6.3.3.34 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 771 行で定義されています。

参照先 CIO::cioPath_DirName().

参照元 MakeDirectory().

```
772 {
773
774   umask(022);
775
776   int ret = mkdir(path.c_str(), 0777);
777   if( ret != 0 )
778   {
779     if( errno == EEXIST ) return 0;
780
781     std::string parent = CIO::cioPath_DirName(path);
782     int ret2 = MakeDirectorySub( parent );
783     if( ret2 != 0 )
784     {
785       return ret2;
786     }
787     ret = MakeDirectorySub( path );
788   }
789
790   return ret;
791
792 }
```

6.3.3.35 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

cio_DFI.C の 1003 行で定義されています。

参照先 m_intervalMngr, と cio_Interval_Mngr::normalizeBaseTime().

```
1004 {
1005   m_intervalMngr.normalizeBaseTime(scale);
1006 }
```

6.3.3.36 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

cio_DFI.C の 1031 行で定義されています。

参照先 m_intervalMngr, と cio_Interval_Mngr::normalizeDelteT().

```
1032 {
1033   m_intervalMngr.normalizeDelteT(scale);
1034 }
```

6.3.3.37 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

cio_DFI.C の 1010 行で定義されています。

参照先 m_intervalMngr, と cio_Interval_Mngr::normalizeIntervalTime().

```
1011 {
1012   m_intervalMngr.normalizeIntervalTime(scale);
1013 }
```

6.3.3.38 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

cio_DFI.C の 1024 行で定義されています。

参照先 m_intervalMngr, と cio_Interval_Mngr::normalizeLastTime().

```
1025 {
1026   m_intervalMngr.normalizeLastTime(scale);
1027 }
```

6.3.3.39 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

cio_DFI.C の 1017 行で定義されています。

参照先 m_intervalMngr, と cio_Interval_Mngr::normalizeStartTime().

```
1018 {
1019   m_intervalMngr.normalizeStartTime(scale);
1020 }
```

6.3.3.40 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

cio_DFI.C の 996 行で定義されています。

参照先 m_intervalMngr, と cio_Interval_Mngr::normalizeTime().

```
997 {
998   return m_intervalMngr.normalizeTime(scale);
999 }
```

6.3.3.41 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, と cio_DFI_BOVで実装されています。

参照元 ReadFieldData().

6.3.3.42 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, と cio_DFI_BOVで実装されています。

参照元 ReadFieldData().

6.3.3.43 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, と cio_DFI_BOVで実装されています。

参照元 ReadFieldData().

6.3.3.44 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 }
```

6.3.3.45 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 }
```

6.3.3.46 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().

6.3.3.47 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

6.3.3.48 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 }
```

6.3.3.49 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_LITTLE, 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  //読み込みバッファ
190   cio_Array* buf = cio_Array::instanceArray
191                    ( DFI_Finfo.DataType
192                    , DFI_Finfo.ArrayShape
193                    , szB
194                    , 0
195                    , DFI_Finfo.Component );
196
197   int szS[3];
198   int headS[3];
199   for(int i=0; i<3; i++) {
200     szS[i]=end[i]-sta[i]+1;
201     headS[i]=sta[i];
202   }
203
204   cio_Array* src = cio_Array::instanceArray
205                    ( DFI_Finfo.DataType
206                    , DFI_Finfo.ArrayShape
207                    , szS
208                    , 0
209                    , DFI_Finfo.Component );
210   src->setHeadIndex( headS );
211
212
213   //data 読込み
214   //if( !read_Datarecord(fp, matchEndian, buf, headB, voxsize[2], src ) ) {
215   ret = read_Datarecord(fp, matchEndian, buf, headB, voxsize[2], src );
216   if( ret != CIO::E_CIO_SUCCESS) {
217     ret = CIO::E_CIO_ERROR_READ_FIELD_DATA_RECORD;
218     fclose(fp);
219     printf("ERROR Data Record Read error!!!!\n");
220     delete buf;
221     return NULL;
222   }
223
224   //read average
225   if( !avr_mode ) {
226     //if( !read_averaged(fp, matchEndian, step, avr_step, avr_time) )
227     ret = read_averaged(fp, matchEndian, step, avr_step, avr_time);
228     if( ret !=CIO::E_CIO_SUCCESS )
229     {
230       ret = CIO::E_CIO_ERROR_READ_FIELD_AVERAGED_RECORD;
231       delete buf;
232       return src;
233     }
234   }
235
236   fclose(fp);
237   delete buf;
238
239   return src;
240
241 }
```

6.3.3.50 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 41 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().

```
46 {
47
49   std::string dirName = CIO::cioPath_DirName(DfiName);
50
51   int RankID;
52   MPI_Comm_rank( comm, &RankID );
53
54   cio_TextParser tpCntl;
55
58   tpCntl.getTPinstance();
```

```
59
60   FILE*fp = NULL;
61   if( !(fp=fopen(DfiName.c_str(),"rb")) ) {
62     printf("Can't open file. (%s)\n",DfiName.c_str());
63     ret = CIO::E_CIO_ERROR_READ_INDEXFILE_OPENERROR;
64     return NULL;
65   }
66   fclose(fp);
67
69   int ierror = 0;
70   ierror = tpCntl.readTPfile(DfiName);
71   if ( ierror )
72   {
73     printf("\tinput file not found '%s'\n",DfiName.c_str());
74     ret = CIO::E_CIO_ERROR_TEXTPARSER;
75     return NULL;
76   }
77
79   cio_FileInfo F_info;
80   if( F_info.Read(tpCntl) != CIO::E_CIO_SUCCESS )
81   {
82     printf("\tFileInfo Data Read error %s\n",DfiName.c_str());
83     ret = CIO::E_CIO_ERROR_READ_FILEINFO;
84     return NULL;
85   }
86
88   cio_FilePath F_path;
89   if( F_path.Read(tpCntl) != CIO::E_CIO_SUCCESS )
90   {
91     printf("\tFilePath Data Read error %s\n",DfiName.c_str());
92     ret = CIO::E_CIO_ERROR_READ_FILEPATH;
93     return NULL;
94   }
95
97   cio_Unit unit;
98   if( unit.Read(tpCntl) != CIO::E_CIO_SUCCESS )
99   {
100    printf("\tUnit Data Read error %s\n",DfiName.c_str());
101    ret = CIO::E_CIO_ERROR_READ_UNIT;
102    return NULL;
103  }
104
106  cio_TimeSlice TimeSlice;
107  if( TimeSlice.Read(tpCntl) != CIO::E_CIO_SUCCESS )
108  {
109    printf("\tTimeSlice Data Read error %s\n",DfiName.c_str());
110    ret = CIO::E_CIO_ERROR_READ_TIMESLICE;
111    return NULL;
112  }
113
115  tpCntl.remove();
116
118  std::string dfiname  = CIO::cioPath_FileName(F_path.ProcDFIFile,".dfi");
119  std::string procfile = CIO::cioPath_ConnectPath(dirName,dfiname);
120
123  tpCntl.getTPinstance();
124
125  fp = NULL;
126  if( !(fp=fopen(procfile.c_str(),"rb")) ) {
127    printf("Can't open file. (%s)\n",procfile.c_str());
128    ret = CIO::E_CIO_ERROR_READ_PROCFILE_OPENERROR;
129    return NULL;
130  }
131  fclose(fp);
132
134  ierror = tpCntl.readTPfile(procfile);
135  if ( ierror )
136  {
137    printf("\tinput file not found '%s'\n",procfile.c_str());
138    ret = CIO::E_CIO_ERROR_TEXTPARSER;
139    return NULL;
140  }
141
143  cio_Domain domain;
144  if( domain.Read(tpCntl) != CIO::E_CIO_SUCCESS )
145  {
146    printf("\tDomain Data Read error %s\n",procfile.c_str());
147    ret = CIO::E_CIO_ERROR_READ_DOMAIN;
148    return NULL;
149  }
150
152  cio_MPI mpi;
153  if( mpi.Read(tpCntl,domain) != CIO::E_CIO_SUCCESS )
154  {
155    printf("\tMPI Data Read error %s\n",procfile.c_str());
156    ret = CIO::E_CIO_ERROR_READ_MPI;
157    return NULL;
```

```
158   }
159
161   cio_Process process;
162   if( process.Read(tpCntl) != CIO::E_CIO_SUCCESS )
163   {
164     printf("\tProcess Data Read error %s\n",procfile.c_str());
165     ret = CIO::E_CIO_ERROR_READ_PROCESS;
166     return NULL;
167   }
168
170   tpCntl.remove();
171
173   cio_DFI *dfi = NULL;
174   if( F_info.FileFormat == CIO::E_CIO_FMT_SPH ) {
175     dfi = new cio_DFI_SPH(F_info, F_path, unit, domain, mpi, TimeSlice, process);
176   } else if( F_info.FileFormat == CIO::E_CIO_FMT_BOV ) {
177     dfi = new cio_DFI_BOV(F_info, F_path, unit, domain, mpi, TimeSlice, process);
178   } else {
179     return NULL;
180   }
181
182   //読込みタイプのチェック
183   dfi->m_read_type = dfi->CheckReadType(G_Voxel,dfi->DFI_Domain.GlobalVoxel,
184                                         G_Div,dfi->DFI_Domain.GlobalDivision);
185   if( dfi->m_read_type == CIO::E_CIO_READTYPE_UNKNOWN ) {
186     //printf("\tDimension size error (%d %d %d)\n",
187     //       G_Voxel[0], G_Voxel[1], G_Voxel[2]);
188     ret = CIO::E_CIO_ERROR_INVALID_DIVNUM;
189     dfi->m_comm = comm;
190     dfi->m_indexDfiName = DfiName;
191     dfi->m_RankID = RankID;
192     return dfi;
193   }
194
195 #if 0
196   if( dfi->m_start_type == E_CIO_SAMEDIV_SAMERES ) {
197     printf("***** SAMEDIV_SAMERES\n");
198   } else if( dfi->m_start_type == E_CIO_SAMEDIV_REFINEMENT ) {
199     printf("***** SAMEDIV_REFINEMENT\n");
200   } else if( dfi->m_start_type == E_CIO_DIFFDIV_SAMERES ) {
201     printf("***** DIFFDIV_SAMERES\n");
202   } else if( dfi->m_start_type == E_CIO_DIFFDIV_REFINEMENT ) {
203     printf("***** DIFFDIV_REFINEMENT\n");
204   }
205 #endif
206
207   dfi->m_comm = comm;
208   dfi->m_indexDfiName = DfiName;
209   dfi->m_RankID = RankID;
210
211   ret = CIO::E_CIO_SUCCESS;
212
213   return dfi;
214
215 }
```

6.3.3.51 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 894 行で定義されています。

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

```
895 {
896
897   DFI_Finfo.setComponentVariable(pcomp, compName);
898
899 }
```

6.3.3.52 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）

cio_DFI.C の 949 行で定義されています。

参照先 cio_Interval_Mngr::By_step, cio_Interval_Mngr::initTrigger(), m_intervalMngr, cio_Interval_Mngr::setInterval(), cio_Interval_Mngr::setLast(), cio_Interval_Mngr::setMode(), と cio_Interval_Mngr::setStart().

```
953 {
954   //インターバルモードの登録（ステップ）
955   m_intervalMngr.setMode(cio_Interval_Mngr::By_step);
956
957   //インターバルステップの登録
958   m_intervalMngr.setInterval(interval_step);
959
960   //インターバル開始ステップの登録
961   m_intervalMngr.setStart(start_step);
962
963   //インターバル最終ステップの登録
964   m_intervalMngr.setLast(last_step);
965
966   //トリガーの初期化
967   m_intervalMngr.initTrigger(base_step,0.0,0.0);
968 }
```

6.3.3.53 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）

cio_DFI.C の 972 行で定義されています。

参照先 cio_Interval_Mngr::By_time, cio_Interval_Mngr::initTrigger(), m_intervalMngr, cio_Interval_Mngr::setInterval(), cio_Interval_Mngr::setLast(), cio_Interval_Mngr::setMode(), と cio_Interval_Mngr::setStart().

```
977 {
978   //インターバルモードの登録（タイム）
979   m_intervalMngr.setMode(cio_Interval_Mngr::By_time);
980
981   //インターバルタイムの登録
982   m_intervalMngr.setInterval(interval_time);
983
984   //インターバル開始タイムの登録
985   m_intervalMngr.setStart(start_time);
986
987   //インターバル終了タイムの登録
988   m_intervalMngr.setLast(last_time);
989
990   //トリガーの初期化
991   m_intervalMngr.initTrigger(0,base_time,dt);
992 }
```

6.3.3.54 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 887 行で定義されています。

参照先 DFI_Finfo, と cio_FileInfo::TimeSliceDirFlag.

```
888 {
889   DFI_Finfo.TimeSliceDirFlag == ONOFF;
890 }
```

6.3.3.55 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_SPH, と cio_DFI_BOVで実装されています。

参照元 WriteFieldData().

6.3.3.56 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_SPH, と cio_DFI_BOVで実装されています。

参照元 WriteFieldData().

6.3.3.57 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_SPH, と cio_DFI_BOVで実装されています。

参照元 WriteFieldData().

6.3.3.58 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*, bool *force*, 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().

```
140 {
141
142   cio_Array *data = cio_Array::instanceArray
143                     ( val
144                     , DFI_Finfo.ArrayShape
145                     , DFI_Process.RankList[m_RankID].VoxelSize[0]
146                     , DFI_Process.RankList[m_RankID].VoxelSize[1]
147                     , DFI_Process.RankList[m_RankID].VoxelSize[2]
148                     , gc
149                     , DFI_Finfo.Component);
150
151   double d_time = (double)time;
152   double d_time_avr = (double)time_avr;
153   double *d_minmax=NULL;
154   if( minmax ) {
155     if( DFI_Finfo.Component>1 ) {
156       d_minmax = new double[DFI_Finfo.Component*2+2];
157       for(int i=0; i<DFI_Finfo.Component*2+2; i++) {
158         d_minmax[i] = minmax[i];
159       }
160     } else {
161       d_minmax = new double[2];
162       d_minmax[0] = minmax[0];
163       d_minmax[1] = minmax[1];
164     }
165   }
166
167   CIO::E_CIO_ERRORCODE ret;
168   ret = WriteData(step, gc, d_time, data, d_minmax, avr_mode, step_avr, d_time_avr, force);
169
170   //val = (T*)data->getData(true);
171   //data->getData(true);
172
173   if( d_minmax ) delete [] d_minmax;
174
175   delete data;
176   return ret;
177
178 }
```

6.3.3.59 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 *force* = true, 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	*force*	強制出力指示
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().

6.3.3.60 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*, bool *force*)

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
in	*force*	強制出力指示

cio_DFI_Write.C の 203 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(), cio_FileInfo::DataType, DFI_Finfo, DFI_MPI, DFI_Process, DFI_TimeSlice, cio_FileInfo::DirectoryPath, CIO::E_CIO_ERROR_MAKEDIRECTORY, CIO::E_CIO_SUCCESS, Generate_FieldFileName(), cio_FileInfo::GuideCell, cio_Array::instanceArray(), cio_Interval_Mngr::isTriggered(), m_directoryPath, m_indexDfiName, m_intervalMngr, m_RankID, MakeDirectory(), cio_MPI::NumberOfRank, cio_Process::RankList, WriteFieldData(), と WriteIndexDfiFile().

```
212 {
213
214   //インターバルチェック
215   if( !m_intervalMngr.isTriggered(step, time, force ) ) return
      CIO::E_CIO_SUCCESS;
216
217   bool mio=false;
218   if( DFI_MPI.NumberOfRank > 1 ) mio=true;
219   std::string outFile;
220   if( CIO::cioPath_isAbsolute(DFI_Finfo.DirectoryPath) ){
221     outFile = Generate_FieldFileName(m_RankID,step,mio);
222   } else {
223     outFile = m_directoryPath + "/"+ Generate_FieldFileName(m_RankID,step,mio);
224   }
225
226   std::string dir = CIO::cioPath_DirName(outFile);
227
228   if( MakeDirectory(dir) != 1 ) return CIO::E_CIO_ERROR_MAKEDIRECTORY;
229
230   cio_Array *outArray = val;
231   if( gc != DFI_Finfo.GuideCell ) {
232     //出力用バッファのインスタンス
```

```
233     outArray = cio_Array::instanceArray
234                ( DFI_Finfo.DataType
235                , DFI_Finfo.ArrayShape
236                , DFI_Process.RankList[m_RankID].VoxelSize
237                , DFI_Finfo.GuideCell
238                , DFI_Finfo.Component);
239     //配列のコピー val -> outArray
240     int ret = val->copyArray(outArray);
241   }
242
243   //フィールデータの出力
244   CIO::E_CIO_ERRORCODE err = CIO::E_CIO_SUCCESS;
245   err = WriteFieldData(outFile, step, time, outArray, avr_mode, step_avr, time_avr);
246
247   //出力バッファのメモリ解放
248   if( val != outArray ) {
249     delete outArray;
250   }
251
252   if( err != CIO::E_CIO_SUCCESS ) return err;
253
254   //index dfi ファイルのディレクトリ作成
255   cio_DFI::MakeDirectory(m_directoryPath);
256   std::string dfiname = CIO::cioPath_FileName(m_indexDfiName,".dfi");
257   std::string fname = CIO::cioPath_ConnectPath( m_directoryPath, dfiname );
258
259   //Slice へのセット
260   DFI_TimeSlice.AddSlice(step, time, minmax, DFI_Finfo.Component, avr_mode,
261                          step_avr, time_avr);
262
263   //index dfi のファイル出力
264   if( m_RankID == 0 ) {
265     err = WriteIndexDfiFile(fname);
266   }
267
268   return err;
269 }
```

6.3.3.61 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 274 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_SUCCESS, cio_FileInfo::GuideCell, m_RankID, write_averaged(), write_DataRecord(), と write_HeaderRecord().

参照元 WriteData().

```
281 {
282
283   FILE* fp;
284   if( (fp = fopen(fname.c_str(),"wb")) == NULL ) {
285     fprintf(stderr,"Can't open file.(%s)\n",fname.c_str());
```

```
286     return CIO::E_CIO_ERROR_OPEN_FIELDDATA;
287   }
288
289   //ヘッダー出力
290   if( write_HeaderRecord(fp, step, time, m_RankID) != CIO::E_CIO_SUCCESS ) {
291     fclose(fp);
292     return CIO::E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD;
293   }
294
295   //データ出力
296   if( write_DataRecord(fp, val, DFI_Finfo.GuideCell, m_RankID) !=
      CIO::E_CIO_SUCCESS) {
297     fclose(fp);
298     return CIO::E_CIO_ERROR_WRITE_FIELD_DATA_RECORD;
299   }
300
301   //average 出力
302   if( !avr_mode ) {
303     if( write_averaged(fp, step_avr, time_avr) != CIO::E_CIO_SUCCESS ) {
304       fclose(fp);
305       return CIO::E_CIO_ERROR_WRITE_FIELD_AVERAGED_RECORD;
306     }
307   }
308
309   fclose(fp);
310
311   return CIO::E_CIO_SUCCESS;
312
313 }
```

6.3.3.62 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_WRI-TE_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 }
```

6.3.3.63 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 276 行で定義されています。

```
294 {
```

```
295
296   // float 型を double 型に変換して double 版 WriteInit 関数を呼ぶ
297
298   double d_pch[3],d_org[3];
299   for(int i=0; i<3; i++) {
300     d_pch[i]=(double)pitch[i];
301     d_org[i]=(double)G_origin[i];
302   }
303
304   return WriteInit(comm,
305                    DfiName,
306                    Path,
307                    prefix,
308                    format,
309                    GCell,
310                    DataType,
311                    ArrayShape,
312                    nComp,
313                    proc_fname,
314                    G_size,
315                    d_pch,
316                    d_org,
317                    division,
318                    head,
319                    tail,
320                    hostname,
321                    TSliceOnOff);
322
323 }
```

6.3.3.64 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 327 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_BOV, CIO::E_CIO_FMT_SPH, 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.

```
345 {
346
347   cio_DFI *dfi = NULL;
348
349   int RankID;
350   MPI_Comm_rank( comm, &RankID );
351
352   int nrank;
353   MPI_Comm_size( comm, &nrank );
354
355   cio_FileInfo out_F_info;
356   out_F_info.DirectoryPath    = Path;
357   out_F_info.TimeSliceDirFlag = TSliceOnOff;
358   out_F_info.Prefix           = prefix;
359   out_F_info.FileFormat       = format;
360   out_F_info.GuideCell        = GCell;
361   out_F_info.DataType         = DataType;
362   out_F_info.ArrayShape       = ArrayShape;
363   out_F_info.Component        = nComp;
364
365   int idumy = 1;
366   char* cdumy = (char*)(&idumy);
367   if( cdumy[0] == 0x01 ) out_F_info.Endian = CIO::E_CIO_LITTLE;
368   if( cdumy[0] == 0x00 ) out_F_info.Endian = CIO::E_CIO_BIG;
369
370   cio_FilePath out_F_path;
371   out_F_path.ProcDFIFile = proc_fname;
372
373   cio_Unit out_unit;
374
375   cio_MPI out_mpi;
376   out_mpi.NumberOfRank = nrank;
377   out_mpi.NumberOfGroup = 1;
378
379   cio_Domain out_domain;
380   cio_Process out_Process;
381   cio_Rank out_Rank;
382
383   for(int i=0; i<nrank; i++ ) {
384      out_Process.RankList.push_back(out_Rank);
385   }
386
387   out_Process.RankList[RankID].RankID=RankID;
388   out_Process.RankList[RankID].HostName=hostname;
389   for(int i=0; i<3; i++) {
390     out_Process.RankList[RankID].HeadIndex[i]=head[i];
391     out_Process.RankList[RankID].TailIndex[i]=tail[i];
392     out_Process.RankList[RankID].VoxelSize[i]=tail[i]-head[i]+1;
393   }
394
395   for(int i=0; i<3; i++) {
396     out_domain.GlobalVoxel[i]  = G_size[i];
397     out_domain.GlobalDivision[i] = division[i];
398     out_domain.GlobalOrigin[i] = G_origin[i];
399     out_domain.GlobalRegion[i] = pitch[i]*G_size[i];
400   }
401
402   cio_TimeSlice out_TSlice;
403
404   char tmpname[512];
405   memset(tmpname,0x00,sizeof(char)*512);
406   if( gethostname(tmpname, 512) != 0 ) printf("*** error gethostname() \n");
407
408   if( out_F_info.FileFormat == CIO::E_CIO_FMT_SPH ) {
409     dfi = new cio_DFI_SPH(out_F_info, out_F_path, out_unit, out_domain, out_mpi,
410                           out_TSlice, out_Process);
411   } else if( out_F_info.FileFormat == CIO::E_CIO_FMT_BOV ) {
412
413     dfi = new cio_DFI_BOV(out_F_info, out_F_path, out_unit, out_domain, out_mpi,
414                           out_TSlice, out_Process);
415   } else return NULL;
416
417
418   dfi->m_indexDfiName = DfiName;
419   dfi->m_directoryPath = CIO::cioPath_DirName(DfiName);
420   dfi->m_comm = comm;
421   dfi->m_RankID = RankID;
422
423   return dfi;
424
```

```
425 }
```

6.3.3.65 CIO::E_CIO_ERRORCODE cio_DFI::WriteProcDfiFile (const MPI_Comm *comm*, bool *out_host* = false, float * *org* = NULL)

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
in	*org*	原点座標値

org がNULL のときは、WriteInit で渡した、G_origin を出力

戻り値

true:出力成功 false:出力失敗

cio_DFI_Write.C の 78 行で定義されています。

参照先 DFI_Domain, と cio_Domain::GlobalOrigin.

```
81 {
82
83   //orign の再設定
84   double d_org[3];
85   if( org != NULL ) {
86     for(int i=0; i<3; i++) {
87       d_org[i]=(double)org[i];
88     }
89   } else {
90     for(int i=0; i<3; i++) {
91       d_org[i]=DFI_Domain.GlobalOrigin[i];
92     }
93   }
94
95   return WriteProcDfiFile(comm, out_host, d_org);
96
97 }
```

6.3.3.66 CIO::E_CIO_ERRORCODE cio_DFI::WriteProcDfiFile (const MPI_Comm *comm*, bool *out_host* = false, double * *org* = NULL)

proc DFI ファイル出力コントロール (double 版)

引数

in	*comm*	MPI コミュニケータ
in	*out_host*	ホスト名出力フラグ
in	*org*	原点座標値

org がNULL のときは、WriteInit で渡した、G_origin を出力

戻り値

true:出力成功 false:出力失敗
true:出力成功 false:出力失敗

cio_DFI_Write.C の 101 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().

```
104 {
105
106   //proc ファイル名の生成
107   std::string procFileName = CIO::cioPath_DirName(m_indexDfiName)+"/"+
      CIO::cioPath_FileName(DFI_Fpath.ProcDFIFile,".dfi");
108
109   if( procFileName.empty() ) return CIO::E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY;
110
111   int RankID;
112   MPI_Comm_rank( comm, &RankID );
113
114   int nrank;
115   MPI_Comm_size( comm, &nrank );
116
117   cio_MPI out_mpi;
118   out_mpi.NumberOfRank = nrank;
119   out_mpi.NumberOfGroup = 1;
120
121   cio_Domain out_domain;
122   cio_Process out_Process;
123
124   //出力する Process 情報の生成
125   cio_Create_dfiProcessInfo(comm, out_Process);
126
127   //orign の設定
128   if( org!=NULL ) {
129     for(int i=0; i<3; i++) {
130       out_domain.GlobalOrigin[i] = org[i];
131     }
132   } else {
133     for(int i=0; i<3; i++) {
134       out_domain.GlobalOrigin[i] = DFI_Domain.GlobalOrigin[i];
135     }
136   }
137
138   //Domain の設定
139   for(int i=0; i<3; i++) {
140     out_domain.GlobalVoxel[i]    = DFI_Domain.GlobalVoxel[i];
141     out_domain.GlobalDivision[i] = DFI_Domain.GlobalDivision[i];
142     out_domain.GlobalRegion[i]   = DFI_Domain.GlobalRegion[i];
143   }
144
145   //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
146   if( out_host ) {
147     const int LEN=256;
148     char *recbuf = new char[out_Process.RankList.size()*LEN];
149     char  sedbuf[LEN];
150     //sprintf(sedbuf,"%s",hostname.c_str());
151     sprintf(sedbuf,"%s",DFI_Process.RankList[RankID].HostName.c_str());
152     MPI_Gather(sedbuf,LEN,MPI_CHAR,recbuf,LEN,MPI_CHAR,0,MPI_COMM_WORLD);
153
154     for( int i=0; i<out_Process.RankList.size(); i++ ) {
155      char* hn =&(recbuf[i*LEN]);
156      out_Process.RankList[i].HostName=(std::string(hn));
157     }
158
159     if( recbuf ) delete [] recbuf;
160   }
161
162   //proc.df の出力
163   if( RankID == 0 ) {
164
165     FILE* fp = NULL;
166     if( !(fp = fopen(procFileName.c_str(), "w")) )
167     {
168       fprintf(stderr, "Can't open file.(%s)\n", procFileName.c_str());
169       return CIO::E_CIO_ERROR_WRITE_PROCFILE_OPENERROR;
170     }
171
172     //Domain {} の出力
173     if( out_domain.Write(fp, 0) != CIO::E_CIO_SUCCESS )
174     {
175       if (fp) fclose(fp);
176       return CIO::E_CIO_ERROR_WRITE_DOMAIN;
177     }
178
179     //MPI {} の出力
180     if( out_mpi.Write(fp, 0) != CIO::E_CIO_SUCCESS )
181     {
182       fclose(fp);
183       return CIO::E_CIO_ERROR_WRITE_MPI;
184     }
```

```
185
186     //Process {} の出力
187     if( out_Process.Write(fp, 0) != CIO::E_CIO_SUCCESS )
188     {
189       fclose(fp);
190       return CIO::E_CIO_ERROR_WRITE_PROCESS;
191     }
192
193     fclose(fp);
194   }
195
196   return CIO::E_CIO_SUCCESS;
197
198 }
```

6.3.4 変数

6.3.4.1 cio_Domain cio_DFI::DFI_Domain [protected]

Domain class.

cio_DFI.h の 61 行で定義されています。

参照元 cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_SPH::cio_DFI_SPH(), CreateReadStartEnd(), GetcioDomain(), GetDFIGlobalDivision(), GetDFIGlobalVoxel(), ReadData(), ReadInit(), cio_DFI_SPH::write_HeaderRecord(), と WriteProcDfiFile().

6.3.4.2 cio_FileInfo cio_DFI::DFI_Finfo [protected]

FileInfo class.

cio_DFI.h の 58 行で定義されています。

参照元 cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_SPH::cio_DFI_SPH(), 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_SPH::write_averaged(), cio_DFI_BOV::write_DataRecord(), cio_DFI_SPH::write_DataRecord(), 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 59 行で定義されています。

参照元 cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_SPH::cio_DFI_SPH(), GetcioFilePath(), WriteIndexDfiFile(), と WriteProcDfiFile().

6.3.4.4 cio_MPI cio_DFI::DFI_MPI [protected]

MPI class.

cio_DFI.h の 62 行で定義されています。

参照元 cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_SPH::cio_DFI_SPH(), GetcioMPI(), と WriteData().

6.3.4.5 cio_Process cio_DFI::DFI_Process [protected]

Process class.

cio_DFI.h の 64 行で定義されています。

参照元 CheckReadRank(), cio_Create_dfiProcessInfo(), cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_SPH::cio_DFI_SPH(), GetcioProcess(), ReadData(), cio_DFI_BOV::write_DataRecord(), cio_DFI_SPH::write_DataRecord(), 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 63 行で定義されています。

参照元 cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_SPH::cio_DFI_SPH(), GetcioTimeSlice(), getMinMax(), getVectorMinMax(), cio_DFI_BOV::read_averaged(), cio_DFI_BOV::read_HeaderRecord(), WriteData(), と WriteIndexDfiFile().

6.3.4.7 cio_Unit cio_DFI::DFI_Unit [protected]

Unit class.

cio_DFI.h の 60 行で定義されています。

参照元 AddUnit(), cio_DFI_BOV::cio_DFI_BOV(), cio_DFI_SPH::cio_DFI_SPH(), GetcioUnit(), GetUnit(), GetUnitElem(), と WriteIndexDfiFile().

6.3.4.8 MPI_Comm cio_DFI::m_comm [protected]

MPI コミュニケータ

cio_DFI.h の 51 行で定義されています。

参照元 ReadInit(), と WriteInit().

6.3.4.9 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 52 行で定義されています。

参照元 Generate_Directory_Path(), WriteData(), と WriteInit().

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

index dfi ファイル名

cio_DFI.h の 53 行で定義されています。

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

6.3.4.11 cio_Interval_Mngr cio_DFI::m_intervalMngr [protected]

インターバルマネージャー

cio_DFI.h の 68 行で定義されています。

参照元 normalizeBaseTime(), normalizeDelteT(), normalizeIntervalTime(), normalizeLastTime(), normalizeStartTime(), normalizeTime(), setIntervalStep(), setIntervalTime(), と WriteData().

6.3.4.12 int cio_DFI::m_RankID [protected]

ランク番号

cio_DFI.h の 56 行で定義されています。

参照元 cio_DFI(), ReadData(), ReadInit(), WriteData(), WriteFieldData(), と WriteInit().

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

読込みタイプ

cio_DFI.h の 54 行で定義されています。

参照元 cio_DFI(), と ReadInit().

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

読込みランクリスト

cio_DFI.h の 66 行で定義されています。

参照元 ReadData().

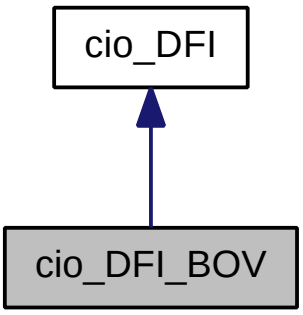
このクラスの説明は次のファイルから生成されました:

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

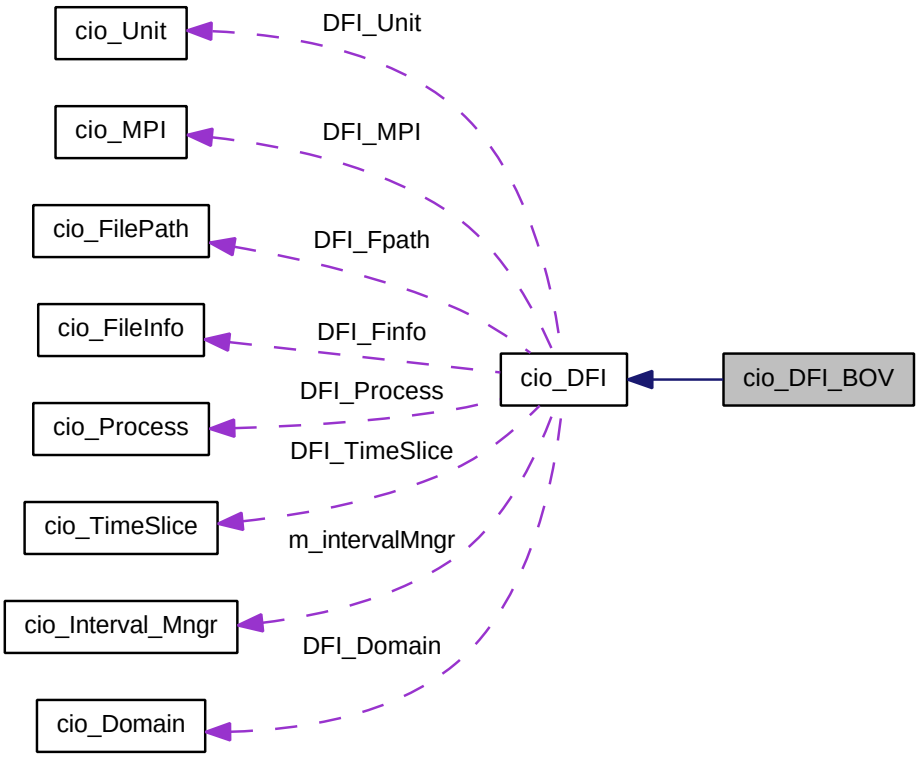
6.4 クラス cio_DFI_BOV

`#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**

Additional Inherited Members

6.4.1 説明

cio_DFI_BOV.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_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 }
```

6.4.2.2 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, と cio_DFI::DFI_Unit.

```
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   };
```

6.4.2.3 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 }
```

6.4.3 関数

6.4.3.1 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 92 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.

```
97 {
98
99   step_avr=0;
100   time_avr=0.0;
101
102   for(int i=0; i<DFI_TimeSlice.SliceList.size(); i++) {
103     if( DFI_TimeSlice.SliceList[i].step == step ) {
104       step_avr=(int)DFI_TimeSlice.SliceList[i].AveragedStep;
105       time_avr=(double)DFI_TimeSlice.SliceList[i].AveragedTime;
106     }
107   }
108
109   return CIO::E_CIO_SUCCESS;
110 }
```

6.4.3.2 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::E_CIO_ERROR_READ_FIELD_DATA_RECORD, CIO::E_CIO_SUCCESS, cio_Array::getArrayLength(), 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   for( int k=0; k<nz; k++ ) {
73     //head インデクスをずらす
74     head[2]=hzB+k;
```

```
75     buf->setHeadIndex(head);
76
77     //1層読み込
78     size_t ndata = buf->getArrayLength();
79     if( buf->readBinary(fp,matchEndian) != ndata ) return
       CIO::E_CIO_ERROR_READ_FIELD_DATA_RECORD;
80
81     // コピー
82     buf->copyArray(src);
83   }
84
85   return CIO::E_CIO_SUCCESS;
86
87 }
```

6.4.3.3 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 }
```

6.4.3.4 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 151 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
154 {
155   return CIO::E_CIO_SUCCESS;
156 }
```

6.4.3.5 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 126 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_HEADER_RECORD, CIO::E_CIO_SUCCESS, cio_DFI::get_cio_Datasize(), cio_Process::RankList, と cio_Array::writeBinary().

```
130 {
131
132   CIO::E_CIO_DTYPE Dtype = (CIO::E_CIO_DTYPE)DFI_Finfo.DataType;
133   int Real_size = get_cio_Datasize(Dtype);
134
135   int size[3];
136   for(int i=0; i<3; i++ ) size[i] = (int)DFI_Process.RankList[n].VoxelSize[i]+(int)(2*gc);
137
138   size_t dLen = (size_t)(size[0] * size[1] * size[2]);
139   if( DFI_Finfo.Component > 1 ) dLen *= 3;
140
141   unsigned int dmy = dLen * Real_size;
142
143   if( val->writeBinary(fp) != dLen ) return
      CIO::E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD;
144
145   return CIO::E_CIO_SUCCESS;
146 }
```

6.4.3.6 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 115 行で定義されています。

参照先 CIO::E_CIO_SUCCESS.

```
119 {
120   return CIO::E_CIO_SUCCESS;
121 }
```

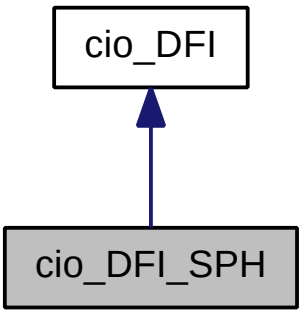
このクラスの説明は次のファイルから生成されました:

- cio_DFI_BOV.h
- cio_DFI_BOV.C

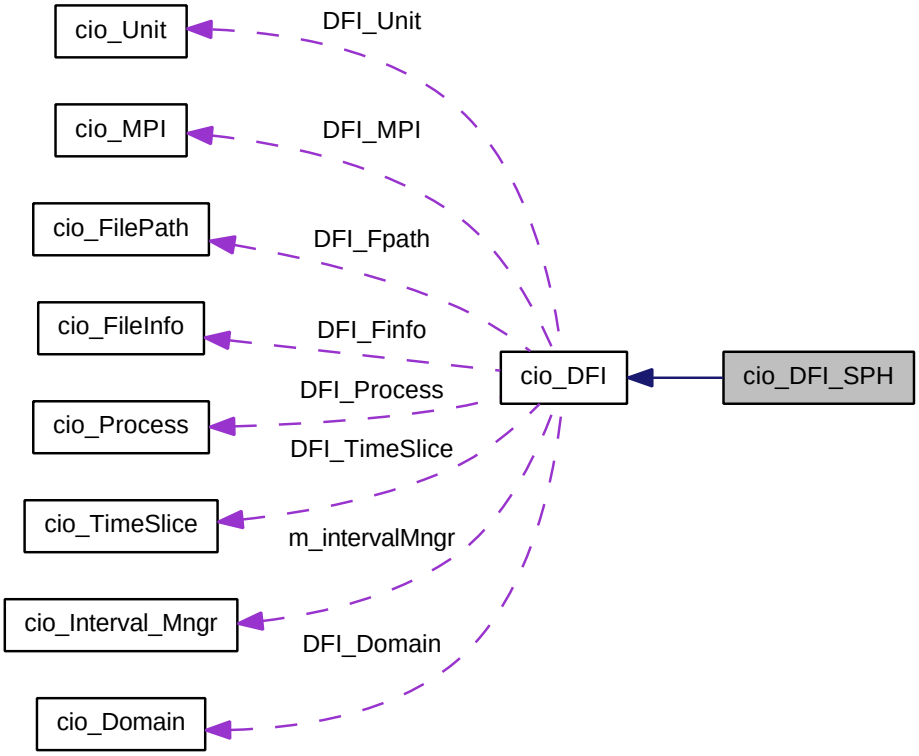
6.5 クラス cio_DFI_SPH

`#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

6.5.1 説明

cio_DFI_SPH.h の 20 行で定義されています。

6.5.2 列挙型

6.5.2.1 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;
```

6.5.2.2 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;
```

6.5.3 コンストラクタとデストラクタ

6.5.3.1 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 }
```

6.5.3.2 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, と cio_DFI::DFI_Unit.

```
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   };
```

6.5.3.3 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 }
```

6.5.4 関数

6.5.4.1 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 }
```

6.5.4.2 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 }
```

6.5.4.3 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 }
```

6.5.4.4 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 423 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.

```
426 {
427   int dType = 0;
428   if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT32 ) dType = 1;
429   if( DFI_Finfo.DataType == CIO::E_CIO_FLOAT64 ) dType = 2;
430
431   unsigned int dmy;
432   int Int_size,Real_size;
433   if ( dType == 1 ) {
434     dmy = 8;
435     Int_size  = sizeof(int);
436     Real_size = sizeof(float);
437   }else{
438     dmy = 16;
439     Int_size  = sizeof(long long);
440     Real_size = sizeof(double);
441   }
442   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) {
443     fclose(fp);
444     return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
445   }
446
447   //averaged step time の出力
448   if( dType == 1 ){
449     //float 型
450     int istep = (int)step_avr;
451     float ttime = (float)time_avr;
452     if( fwrite(&istep, Int_size, 1, fp) != 1 ) {
453       fclose(fp);
454       return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
455     }
456     if( fwrite(&ttime, Real_size, 1, fp) != 1 ) {
457       fclose(fp);
458       return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
459     }
460   } else {
461     //doublet 型
462     long long dstep = (long long)step_avr;
463     double ttime = (double)time_avr;
464     if( fwrite(&dstep, Int_size, 1, fp) != 1 ) {
465       fclose(fp);
466       return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
467     }
468     if( fwrite(&ttime, Real_size, 1, fp) != 1 ) {
469       fclose(fp);
470       return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
471     }
472   }
473
474   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) {
475     fclose(fp);
476     return CIO::E_CIO_ERROR_WRITE_SPH_REC7;
```

```
477   }
478
479   return CIO::E_CIO_SUCCESS;
480
481 }
```

6.5.4.5 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 397 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().

```
401 {
402
403   CIO::E_CIO_DTYPE Dtype = (CIO::E_CIO_DTYPE)DFI_Finfo.DataType;
404   int Real_size = get_cio_Datasize(Dtype);
405
406   int size[3];
407   for(int i=0; i<3; i++ ) size[i] = (int)DFI_Process.RankList[n].VoxelSize[i]+(int)(2*gc);
408
409   size_t dLen = (size_t)(size[0] * size[1] * size[2]);
410   if( DFI_Finfo.Component > 1 ) dLen *= 3;
411
412   unsigned int dmy = dLen * Real_size;
413
414   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC6;
415   if( val->writeBinary(fp) != dLen ) return CIO::E_CIO_ERROR_WRITE_SPH_REC6;
416   if( fwrite(&dmy, sizeof(int), 1, fp) != 1 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC6;
417   return CIO::E_CIO_SUCCESS;
418 }
```

6.5.4.6 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 org[3];
343     for(int i=0; i<3; i++ ) org[i]=(float)DFI_Domain.GlobalOrigin[i];
344     if( fwrite(org, sizeof(float), 3, fp) !=3 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC3;
345   } else {
346     double org[3];
347     for(int i=0; i<3; i++ ) org[i]=(double)DFI_Domain.GlobalOrigin[i];
348     if( fwrite(org, sizeof(double), 3, fp) !=3 ) return
      CIO::E_CIO_ERROR_WRITE_SPH_REC3;
349   }
```

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

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

6.6 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**

6.6.1 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。

6.6.2 コンストラクタとデストラクタ

6.6.2.1 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 }
```

6.6.2.2 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 }
```

6.6.2.3 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 }
```

6.6.3 関数

6.6.3.1 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 }
```

6.6.3.2 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 }
```

6.6.4 変数

6.6.4.1 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().

6.6.4.2 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().

6.6.4.3 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_SPH::write_HeaderRecord(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

6.6.4.4 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_SPH::write_HeaderRecord(), cio_DFI::WriteInit(), と cio_DFI::WriteProcDfiFile().

6.6.4.5 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_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

6.7 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

6.7.1 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。

6.7.2 コンストラクタとデストラクタ

6.7.2.1 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 }
```

6.7.2.2 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	*_TimeSliceDirFlag*	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 TimeSliceDirFlag.

```
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 }
```

6.7.2.3 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 }
```

6.7.3 関数

6.7.3.1 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 }
```

6.7.3.2 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 }
```

6.7.3.3 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 }
```

6.7.3.4 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 }
```

6.7.4 変数

6.7.4.1 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::WriteData(), と cio_DFI::WriteInit().

6.7.4.2 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_DataRecord(), cio_DFI_SPH::write_DataRecord(), cio_DFI_SPH::write_HeaderRecord(), cio_DFI::WriteData(), と cio_DFI::WriteInit().

6.7.4.3 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().

6.7.4.4 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_SPH::write_DataRecord(), cio_DFI_SPH::write_HeaderRecord(), cio_DFI::WriteData(), と cio_DFI::WriteInit().

6.7.4.5 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::WriteData(), と cio_DFI::WriteInit().

6.7.4.6 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(), と cio_DFI::WriteInit().

6.7.4.7 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(), と cio_DFI::WriteInit().

6.7.4.8 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_SPH::write_HeaderRecord(), cio_DFI::WriteData(), cio_DFI::WriteFieldData(), と cio_DFI::WriteInit().

6.7.4.9 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::WriteIndexDfiFile(), と cio_DFI::WriteInit().

6.7.4.10 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(), 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

6.8 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**

6.8.1 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。

6.8.2 コンストラクタとデストラクタ

6.8.2.1 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 }
```

6.8.2.2 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 }
```

6.8.2.3 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 }
```

6.8.3 関数

6.8.3.1 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 }
```

6.8.3.2 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 }
```

6.8.4 変数

6.8.4.1 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

6.9 クラス cio_Interval_Mngr

`#include <cio_Interval_Mngr.h>`

Public 型

- enum type_IO_spec { noset =-1, By_step = 1, By_time = 2 }

  モード

Public メソッド

- cio_Interval_Mngr ()

  コンストラクタ
- ~cio_Interval_Mngr ()

  デストラクタ
- void setMode (const type_IO_spec mode)

  モードをセット
- type_IO_spec getMode ()

  モードを取得
- bool setInterval (const double interval)

  インターバル値のセット
- int getIntervalStep ()

  インターバル *(ステップ)* の取得
- double getIntervalTime ()

  インターバル *(時刻)* の取得
- bool setLast (const double last)

  最終ステップ *(時刻)* のセット
- bool setStart (const double start)

  セッション開始ステップ *(時刻)* のセット
- int getStartStep ()

  セッション開始ステップの取得
- double getStartTime ()

  セッション開始時刻の取得
- bool initTrigger (const int step, const double time, const double dt, bool d_flag=false)

  トリガーの初期化
- bool isTriggered (const int step, const double time, bool forced_out=false, bool d_flag=false)
- bool isStarted (const int step, const double time)

  セッションが開始しているかをチェック
- bool normalizeTime (const double scale)
- void normalizeBaseTime (const double scale)
- void normalizeIntervalTime (const double scale)
- void normalizeStartTime (const double scale)
- void normalizeLastTime (const double scale)
- void normalizeDelteT (const double scale)

Protected メソッド

- int calcNextStep (const int step)
  次の出力ステップを計算
- double calcNextTime (const double time)
  次の出力時刻を計算
- bool isLastStep (const int step)
  最終ステップかどうか
- bool isLastTime (const double time)
  最終時刻かどうか

Static Protected メソッド

- static double dmod (double a, double b)
  実数の余り *(fortran* **の** *mod* **と同じ)**

Protected 変数

- type_IO_spec m_mode
  モード *(noset:***未定義、** *By_step:***ステップ間隔指定、** *By_time:***時刻間隔指定)**
- int m_base_step
  インターバルの基点となるステップ *(By_step* **のとき有効)**
- int m_intvl_step
  ステップ間隔 *(By_step* **のとき有効)**
- int m_start_step
  セッションを開始するステップ *(By_step* **のとき有効)**
- int m_last_step
  最終ステップ *(By_step* **のとき有効)**
- double m_base_time
  インターバルの基点となる時刻 *(By_time* **のとき有効)**
- double m_intvl_time
  時刻間隔 *(By_time* **のとき有効)**
- double m_start_time
  セッションを開始する時刻 *(By_time* **のとき有効)**
- double m_last_time
  最終時刻 *(By_time* **のとき有効)**
- double m_dt
  計算の時間間隔 Δt

6.9.1 説明

cio_Interval_Mngr.h の 23 行で定義されています。

6.9.2 列挙型

6.9.2.1 enum cio_Interval_Mngr::type_IO_spec

モード

列挙型の値

noset

By_step

By_time

cio_Interval_Mngr.h の 28 行で定義されています。

```
29   {
30     noset  =-1,
31     By_step= 1,
32     By_time= 2,
33   };
```

6.9.3 コンストラクタとデストラクタ

6.9.3.1 cio_Interval_Mngr::cio_Interval_Mngr ( ) [inline]

コンストラクタ

cio_Interval_Mngr.h の 54 行で定義されています。

参照先 m_base_step, m_base_time, m_dt, m_intvl_step, m_intvl_time, m_last_step, m_last_time, m_mode, m_start_step, m_start_time, と noset.

```
55   {
56     m_mode = noset;
57     m_base_step  = 0;
58     m_intvl_step = 0;
59     m_start_step = 0;
60     m_last_step  = -1;
61     m_base_time  = 0.0;
62     m_intvl_time = 0.0;
63     m_start_time = 0.0;
64     m_last_time  = -1.0;
65     m_dt         = 0.0;
66   }
```

6.9.3.2 cio_Interval_Mngr::~cio_Interval_Mngr ( ) [inline]

デストラクタ

cio_Interval_Mngr.h の 69 行で定義されています。

```
70   {
71   }
```

6.9.4 関数

6.9.4.1 int cio_Interval_Mngr::calcNextStep (const int *step*) [inline], [protected]

次の出力ステップを計算

cio_Interval_Mngr.h の 336 行で定義されています。

参照先 m_base_step, m_intvl_step, と m_start_step.

参照元 initTrigger(), と isTriggered().

```
337   {
338     int s_step = (m_start_step > step) ? m_start_step : step;
339     int inc    = ((s_step-m_base_step)%m_intvl_step==0) ? 0 : 1;
340     int next_step = int((s_step-m_base_step)/m_intvl_step+inc) * m_intvl_step +
      m_base_step;
341     return next_step;
342   }
```

6.9.4.2 double cio_Interval_Mngr::calcNextTime (const double *time*) [inline], [protected]

次の出力時刻を計算

cio_Interval_Mngr.h の 345 行で定義されています。

参照先 dmod(), m_base_time, m_intvl_time, と m_start_time.

参照元 initTrigger(), と isTriggered().

```
346   {
347     //int s_time = (m_start_time > time) ? m_start_time : time;
348     double s_time = (m_start_time > time) ? m_start_time : time;
349     int inc    = (dmod(s_time-m_base_time, m_intvl_time)==0.0) ? 0 : 1;
350     double next_time = int((s_time-m_base_time)/m_intvl_time+inc) * m_intvl_time +
      m_base_time;
351     return next_time;
352   }
```

6.9.4.3 static double cio_Interval_Mngr::dmod (double *a*, double *b*) [inline], [static], [protected]

実数の余り (fortran の mod と同じ)

cio_Interval_Mngr.h の 355 行で定義されています。

参照元 calcNextTime().

```
356   {
357     return a - int(a/b) * b;
358   }
```

6.9.4.4 int cio_Interval_Mngr::getIntervalStep ( ) [inline]

インターバル (ステップ) の取得

cio_Interval_Mngr.h の 104 行で定義されています。

参照先 m_intvl_step.

```
105   {
106     return m_intvl_step;
107   }
```

6.9.4.5 double cio_Interval_Mngr::getIntervalTime ( ) [inline]

インターバル (時刻) の取得

cio_Interval_Mngr.h の 110 行で定義されています。

参照先 m_intvl_time.

```
111   {
112     return m_intvl_time;
113   }
```

6.9.4.6 type_IO_spec cio_Interval_Mngr::getMode ( ) [inline]

モードを取得

cio_Interval_Mngr.h の 80 行で定義されています。

参照先 m_mode.

```
81   {
82     return m_mode;
83   }
```

6.9.4.7 int cio_Interval_Mngr::getStartStep ( ) [inline]

セッション開始ステップの取得

cio_Interval_Mngr.h の 152 行で定義されています。

参照先 m_start_step.

```
153   {
154     return m_start_step;
155   }
```

6.9.4.8 double cio_Interval_Mngr::getStartTime ( ) [inline]

セッション開始時刻の取得

cio_Interval_Mngr.h の 158 行で定義されています。

参照先 m_start_time.

```
159   {
160     return m_start_time;
161   }
```

6.9.4.9 bool cio_Interval_Mngr::initTrigger (const int *step,* const double *time,* const double *dt,* bool *d_flag* = false) [inline]

トリガーの初期化

cio_Interval_Mngr.h の 164 行で定義されています。

参照先 By_step, By_time, calcNextStep(), calcNextTime(), m_base_step, m_base_time, m_dt, m_intvl_step, m_intvl_time, m_last_step, m_last_time, m_mode, m_start_step, と m_start_time.

参照元 cio_DFI::setIntervalStep(), と cio_DFI::setIntervalTime().

```
165   {
166     m_dt = dt;
167     if( m_mode == By_step )
168     {
169       m_base_step = step;
170       if( d_flag )
171       {
172         int next_step = calcNextStep(step);
173         if( next_step == m_base_step ) next_step += m_intvl_step;
174         printf("cio_Interval_Mngr : mode=By_step, base_step=%d, interval=%d, start=%d, first_step=%d"
175                , m_base_step, m_intvl_step, m_start_step, next_step);
176         if( m_last_step > 0 )
177           printf(", last_step=%d\n", m_last_step);
178         else
179           printf("\n");
180       }
181     }
182     else if( m_mode == By_time )
183     {
184       m_base_time = time;
185       if( d_flag )
186       {
187         double next_time = calcNextTime(time);
188         if( next_time == m_base_time ) next_time += m_intvl_time;
189         printf("cio_Interval_Mngr : mode=By_time, base_time=%e, delta_t=%e, interval=%e, start=%e,
      first_time=%e"
190                , m_base_time, m_dt, m_intvl_time, m_start_time, next_time);
191         if( m_last_time > 0.0 )
192           printf(", last_time=%e\n", m_last_time);
193         else
194           printf("\n");
195       }
196     }
197     else
198     {
199       return false;
```

```
200     }
201     return true;
202   }
```

6.9.4.10 bool cio_Interval_Mngr::isLastStep (const int *step*) [inline],[protected]

最終ステップかどうか

cio_Interval_Mngr.h の 361 行で定義されています。

参照先 m_last_step.

参照元 isTriggered().

```
362   {
363     if( m_last_step <= 0 ) return false;
364     if( m_last_step == step ) return true;
365     return false;
366   }
```

6.9.4.11 bool cio_Interval_Mngr::isLastTime (const double *time*) [inline],[protected]

最終時刻かどうか

cio_Interval_Mngr.h の 369 行で定義されています。

参照先 m_dt, と m_last_time.

参照元 isTriggered().

```
370   {
371     if( m_last_time <= 0 ) return false;
372     if( time <= m_last_time && m_last_time < time + m_dt ) return true;
373     return false;
374   }
```

6.9.4.12 bool cio_Interval_Mngr::isStarted (const int *step*, const double *time*) [inline]

セッションが開始しているかをチェック

cio_Interval_Mngr.h の 286 行で定義されています。

参照先 By_step, By_time, m_mode, m_start_step, と m_start_time.

参照元 isTriggered().

```
287   {
288     if( m_mode == By_step )
289     {
290       if( m_start_step <= step ) return true;
291     }
292     else if( m_mode == By_time )
293     {
294       if( m_start_time <= time ) return true;
295     }
296     return false;
297   }
```

6.9.4.13 bool cio_Interval_Mngr::isTriggered (const int *step*, const double *time*, bool *forced_out* = false, bool *d_flag* = false) [inline]

インターバルのチェック 出力ステップ (時刻) であっても、同じステップ (時刻) に既に isTriggered がコールされている場合は false を返す (重複出力対応)

cio_Interval_Mngr.h の 206 行で定義されています。

参照先 By_step, By_time, calcNextStep(), calcNextTime(), isLastStep(), isLastTime(), isStarted(), m_dt, m_intvl_step, m_intvl_time, m_mode, と noset.

参照元 cio_DFI::WriteData().

```
207   {
208     // 強制出力フラグ
209     if( forced_out )
210     {
211       return true;
212     }
213
214     // noset のときは必ず true(上位プログラムで判定しているものとする)
215     if( m_mode == noset )
216     {
217       return true;
218     }
219
220     // セッションが開始しているか
221     if( !isStarted(step, time) )
222     {
223       return false;
224     }
225
226     // ステップ指定のとき
227     if( m_mode == By_step )
228     {
229       // interval が正常か
230       if( m_intvl_step <= 0 )
231       {
232         return false;
233       }
234       // 次の出力ステップかどうか
235       if( step == calcNextStep( step ) )
236       {
237         if( d_flag )
238         {
239           printf("cio_Interval_Mngr::isTriggerd : step=%d\n", step);
240         }
241         return true;
242       }
243       // 最終ステップかどうか
244       if( isLastStep( step ) )
245       {
246         if( d_flag )
247         {
248           printf("cio_Interval_Mngr::isTriggerd : last step=%d\n", step);
249         }
250         return true;
251       }
252     }
253     // 時刻指定のとき
254     else if( m_mode == By_time )
255     {
256       // interval が正常か
257       if( m_intvl_time <= 0.0 )
258       {
259         return false;
260       }
261       // 次の出力時刻
262       double next_time = calcNextTime( time );
263       // 次の出力時刻かどうか
264       if( time <= next_time && next_time < time + m_dt )
265       {
266         if( d_flag )
267         {
268           printf("cio_Interval_Mngr::isTriggerd : time=%e\n", time);
269         }
270         return true;
271       }
272       // 最終時刻かどうか
273       if( isLastTime( time ) )
274       {
275         if( d_flag )
276         {
277           printf("cio_Interval_Mngr::isTriggerd : last time=%e\n", time);
278         }
279         return true;
280       }
281     }
282     return false;
283   }
```

6.9.4.14 void cio_Interval_Mngr::normalizeBaseTime (const double *scale*) [inline]

cio_Interval_Mngr.h の 313 行で定義されています。

参照先 m_base_time.

参照元 cio_DFI::normalizeBaseTime(), と normalizeTime().

```
314   {
315     m_base_time /= scale;
316   }
```

6.9.4.15 void cio_Interval_Mngr::normalizeDelteT (const double *scale*) [inline]

cio_Interval_Mngr.h の 329 行で定義されています。

参照先 m_dt.

参照元 cio_DFI::normalizeDelteT(), と normalizeTime().

```
330   {
331     m_dt /= scale;
332   }
```

6.9.4.16 void cio_Interval_Mngr::normalizeIntervalTime (const double *scale*) [inline]

cio_Interval_Mngr.h の 317 行で定義されています。

参照先 m_intvl_time.

参照元 cio_DFI::normalizeIntervalTime(), と normalizeTime().

```
318   {
319     m_intvl_time /= scale;
320   }
```

6.9.4.17 void cio_Interval_Mngr::normalizeLastTime (const double *scale*) [inline]

cio_Interval_Mngr.h の 325 行で定義されています。

参照先 m_last_time.

参照元 cio_DFI::normalizeLastTime(), と normalizeTime().

```
326   {
327     m_last_time /= scale;
328   }
```

6.9.4.18 void cio_Interval_Mngr::normalizeStartTime (const double *scale*) [inline]

cio_Interval_Mngr.h の 321 行で定義されています。

参照先 m_start_time.

参照元 cio_DFI::normalizeStartTime(), と normalizeTime().

```
322   {
323     m_start_time /= scale;
324   }
```

6.9.4.19 bool cio_Interval_Mngr::normalizeTime (const double *scale*) [inline]

cio_Interval_Mngr.h の 300 行で定義されています。

参照先 By_time, m_mode, normalizeBaseTime(), normalizeDelteT(), normalizeIntervalTime(), normalizeLastTime(), と normalizeStartTime().

参照元 cio_DFI::normalizeTime().

```
301   {
302     if( m_mode != By_time )
303     {
304       return false;
305     }
306     normalizeBaseTime(scale);
307     normalizeIntervalTime(scale);
308     normalizeStartTime(scale);
309     normalizeLastTime(scale);
310     normalizeDelteT(scale);
311     return true;
312   }
```

6.9.4.20 bool cio_Interval_Mngr::setInterval (const double *interval*) [inline]

インターバル値のセット

cio_Interval_Mngr.h の 86 行で定義されています。

参照先 By_step, By_time, m_intvl_step, m_intvl_time, と m_mode.

参照元 cio_DFI::setIntervalStep(), と cio_DFI::setIntervalTime().

```
87   {
88     if( m_mode == By_step )
89     {
90       m_intvl_step = (int)interval;
91     }
92     else if( m_mode == By_time )
93     {
94       m_intvl_time = interval;
95     }
96     else
97     {
98       return false;
99     }
100     return true;
101   }
```

6.9.4.21 bool cio_Interval_Mngr::setLast (const double *last*) [inline]

最終ステップ (時刻) のセット

cio_Interval_Mngr.h の 116 行で定義されています。

参照先 By_step, By_time, m_last_step, m_last_time, と m_mode.

参照元 cio_DFI::setIntervalStep(), と cio_DFI::setIntervalTime().

```
117   {
118     if( m_mode == By_step )
119     {
120       m_last_step = (int)last;
121     }
122     else if( m_mode == By_time )
123     {
124       m_last_time = last;
125     }
126     else
127     {
128       return false;
129     }
130     return true;
131   }
```

6.9.4.22 void cio_Interval_Mngr::setMode (const type_IO_spec *mode*) [inline]

モードをセット

cio_Interval_Mngr.h の 74 行で定義されています。

参照先 m_mode.

参照元 cio_DFI::setIntervalStep(), と cio_DFI::setIntervalTime().

```
75   {
76     m_mode = mode;
77   }
```

6.9.4.23 bool cio_Interval_Mngr::setStart (const double *start*) [inline]

セッション開始ステップ (時刻) のセット

cio_Interval_Mngr.h の 134 行で定義されています。

参照先 By_step, By_time, m_mode, m_start_step, と m_start_time.

参照元 cio_DFI::setIntervalStep(), と cio_DFI::setIntervalTime().

```
135   {
136     if( m_mode == By_step )
137     {
138       m_start_step = (int)start;
139     }
140     else if( m_mode == By_time )
141     {
142       m_start_time = start;
143     }
144     else
145     {
146       return false;
147     }
148     return true;
149   }
```

6.9.5 変数

6.9.5.1 int cio_Interval_Mngr::m_base_step [protected]

インターバルの基点となるステップ (By_step のとき有効)

cio_Interval_Mngr.h の 39 行で定義されています。

参照元 calcNextStep(), cio_Interval_Mngr(), と initTrigger().

6.9.5.2 double cio_Interval_Mngr::m_base_time [protected]

インターバルの基点となる時刻 (By_time のとき有効)

cio_Interval_Mngr.h の 44 行で定義されています。

参照元 calcNextTime(), cio_Interval_Mngr(), initTrigger(), と normalizeBaseTime().

6.9.5.3 double cio_Interval_Mngr::m_dt [protected]

計算の時間間隔 Δ t

cio_Interval_Mngr.h の 49 行で定義されています。

参照元 cio_Interval_Mngr(), initTrigger(), isLastTime(), isTriggered(), と normalizeDelteT().

6.9.5.4 int cio_Interval_Mngr::m_intvl_step [protected]

ステップ間隔 (By_step のとき有効)

cio_Interval_Mngr.h の 40 行で定義されています。

参照元 calcNextStep(), cio_Interval_Mngr(), getIntervalStep(), initTrigger(), isTriggered(), と setInterval().

6.9.5.5 double cio_Interval_Mngr::m_intvl_time [protected]

時刻間隔 (By_time のとき有効)

cio_Interval_Mngr.h の 45 行で定義されています。

参照元 calcNextTime(), cio_Interval_Mngr(), getIntervalTime(), initTrigger(), isTriggered(), normalizeIntervalTime(), と setInterval().

6.9.5.6 int cio_Interval_Mngr::m_last_step [protected]

最終ステップ (By_step のとき有効)

cio_Interval_Mngr.h の 42 行で定義されています。

参照元 cio_Interval_Mngr(), initTrigger(), isLastStep(), と setLast().

6.9.5.7 double cio_Interval_Mngr::m_last_time [protected]

最終時刻 (By_time のとき有効)

cio_Interval_Mngr.h の 47 行で定義されています。

参照元 cio_Interval_Mngr(), initTrigger(), isLastTime(), normalizeLastTime(), と setLast().

6.9.5.8 type_IO_spec cio_Interval_Mngr::m_mode [protected]

モード (noset:未定義、By_step:ステップ間隔指定、By_time:時刻間隔指定)

cio_Interval_Mngr.h の 37 行で定義されています。

参照元 cio_Interval_Mngr(), getMode(), initTrigger(), isStarted(), isTriggered(), normalizeTime(), setInterval(), setLast(), setMode(), と setStart().

6.9.5.9 int cio_Interval_Mngr::m_start_step [protected]

セッションを開始するステップ (By_step のとき有効)

cio_Interval_Mngr.h の 41 行で定義されています。

参照元 calcNextStep(), cio_Interval_Mngr(), getStartStep(), initTrigger(), isStarted(), と setStart().

6.9.5.10 double cio_Interval_Mngr::m_start_time [protected]

セッションを開始する時刻 (By_time のとき有効)

cio_Interval_Mngr.h の 46 行で定義されています。

参照元 calcNextTime(), cio_Interval_Mngr(), getStartTime(), initTrigger(), isStarted(), normalizeStartTime(), と setStart().

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_Interval_Mngr.h

6.10 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

6.10.1 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。

6.10.2 コンストラクタとデストラクタ

6.10.2.1 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 }
```

6.10.2.2 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 }
```

6.10.2.3 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 }
```

6.10.3 関数

6.10.3.1 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 }
```

6.10.3.2 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 }
```

6.10.4 変数

6.10.4.1 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().

6.10.4.2 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::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

6.11 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

6.11.1 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。

6.11.2 型定義

6.11.2.1 typedef std::map<int,int> cio_Process::headT

cio_Process.h の 63 行で定義されています。

6.11.3 コンストラクタとデストラクタ

6.11.3.1 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 }
```

6.11.3.2 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 }
```

6.11.4 関数

6.11.4.1 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 }
```

6.11.4.2 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 }
```

6.11.4.3 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 }
```

6.11.4.4 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 }
```

6.11.4.5 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 }
```

6.11.4.6 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 }
```

6.11.4.7 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 }
```

6.11.4.8 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 }
```

6.11.4.9 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 }
```

6.11.4.10 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 }
```

6.11.4.11 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 }
```

6.11.4.12 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 }
```

6.11.4.13 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 }
```

6.11.5 変数

6.11.5.1 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().

6.11.5.2 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_DataRecord(), cio_DFI_SPH::write_DataRecord(), 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

6.12 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

6.12.1 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。

6.12.2 コンストラクタとデストラクタ

6.12.2.1 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 }
```

6.12.2.2 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 }
```

6.12.3 関数

6.12.3.1 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 }
```

6.12.3.2 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 }
```

6.12.4 変数

6.12.4.1 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().

6.12.4.2 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().

6.12.4.3 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().

6.12.4.4 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().

6.12.4.5 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

6.13 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

6.13.1 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。

6.13.2 コンストラクタとデストラクタ

6.13.2.1 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 }
```

6.13.2.2 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 }
```

6.13.3 関数

6.13.3.1 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 }
```

6.13.3.2 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 }
```

6.13.4 変数

6.13.4.1 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().

6.13.4.2 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().

6.13.4.3 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().

6.13.4.4 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().

6.13.4.5 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().

6.13.4.6 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().

6.13.4.7 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().

6.13.4.8 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().

6.13.4.9 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

6.14 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

6.14.1 説明

cio_TextParser.h の 30 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_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 {};
```

6.14.2.2 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 {};
```

6.14.3 関数

6.14.3.1 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 }
```

6.14.3.2 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 }
```

6.14.3.3 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 }
```

6.14.3.4 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 }
```

6.14.3.5 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 }
```

6.14.3.6 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 }
```

6.14.3.7 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 }
```

6.14.3.8 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 }
```

6.14.3.9 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 }
```

6.14.3.10 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 }
```

6.14.3.11 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 }
```

6.14.3.12 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 }
```

6.14.3.13 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   }
```

6.14.4 変数

6.14.4.1 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

6.15 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

6.15.1 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。

6.15.2 コンストラクタとデストラクタ

6.15.2.1 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 }
```

6.15.2.2 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 }
```

6.15.3 関数

6.15.3.1 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 }
```

6.15.3.2 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 }
```

6.15.3.3 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 }
```

6.15.3.4 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 }
```

6.15.3.5 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 }
```

6.15.4 変数

6.15.4.1 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(), と Write().

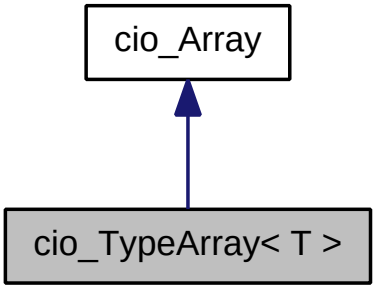
このクラスの説明は次のファイルから生成されました:

- cio_TimeSlice.h
- cio_TimeSlice.C

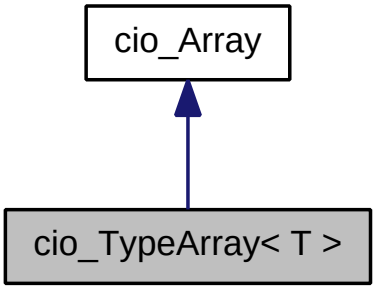
6.16 クラス テンプレート cio_TypeArray< T >

`#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 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***)*

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

6.16.1 説明

template<class T>class cio_TypeArray< T >

cio_TypeArray.h の 7 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 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 20 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.

```
27     : cio_Array(dtype,shape,ix,jx,kx,gc,ncomp)
28   {
29
30     m_outptr=false;
31
32     size_t nw=1;
33     for( int i=0;i<3;i++ )
34     {
35       nw *= (m_sz[i]+2*m_gc);
36     }
37     nw *= m_ncomp;
38     m_data = new T[nw];
39     memset(m_data,0,sizeof(T)*nw);
40   }
```

6.16.2.2 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 43 行で定義されています。

参照先 cio_TypeArray< T >::m_data, と cio_TypeArray< T >::m_outptr.

```
51     : cio_Array(dtype,shape,ix,jx,kx,gc,ncomp)
52   {
53
54     m_outptr=true;
55
56     m_data = data;
57   }
```

6.16.2.3 template<class T> virtual cio_TypeArray< T >::~cio_TypeArray ( ) [inline], [virtual]

デストラクタ

cio_TypeArray.h の 60 行で定義されています。

参照先 cio_TypeArray< T >::m_data, と cio_TypeArray< T >::m_outptr.

```
61   {
62     if( m_data )
63     {
64       if( m_data && !m_outptr )
65       {
```

```
66         delete [] m_data;
67       }
68     }
69   }
```

6.16.2.4 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 119 行で定義されています。

参照先 cio_TypeArray< T >::m_data.

```
119                   : cio_Array()
120   {
121     m_data  = NULL;
122   }
```

6.16.3 関数

6.16.3.1 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 357 行で定義されています。

```
358 {
359   return _val(i,j,k,n);
360 }
```

6.16.3.2 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 346 行で定義されています。

```
347 {
348   return m_data[ m_Sz[2] * m_Sz[1] * m_Sz[0] * n
349                        + m_Sz[1] * m_Sz[0] * k
350                                  + m_Sz[0] * j
351                                            + i ];
352 }
```

6.16.3.3 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 365 行で定義されています。

参照先 cio_Array::getGcInt(), cio_Array::getHeadIndex(), と cio_Array::getTailIndex().

```
366 {
367   cio_TypeArray<T> *src = this;
368
369   // コピーの範囲
370   int       gcS    = src->getGcInt();
371   const int *headS = src->getHeadIndex();
372   const int *tailS = src->getTailIndex();
373   int       gcD    = dst->getGcInt();
374   const int *headD = dst->getHeadIndex();
375   const int *tailD = dst->getTailIndex();
376   if( ignoreGc )
```

```
377   {
378     gcS = gcD = 0;
379   }
380   int sta[3],end[3];
381   for( int i=0;i<3;i++ )
382   {
383     sta[i] = (headS[i]-gcS>=headD[i]-gcD) ? headS[i]-gcS : headD[i]-gcD;
384     end[i] = (tailS[i]+gcS<=tailD[i]+gcD) ? tailS[i]+gcS : tailD[i]+gcD;
385   }
386
387   return copyArray(sta,end,dst);
388 }
```

6.16.3.4 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 393 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().

```
394 {
395   cio_TypeArray<T> *src = this;
396
397   //mod.s
398   cio_TypeArray<T> *dst = dynamic_cast<cio_TypeArray<T>*>(dstptr);
399   if( !dst )
400   {
401     return 1;
402   }
403
404   // データタイプのチェック
405   if( src->getDataType() != dst->getDataType() )
406   {
407     return 2;
408   }
409   CIO::E_CIO_DTYPE dtype = src->getDataType();
410
411   // 配列形状
412   if( src->getArrayShape() != dst->getArrayShape() )
413   {
414     return 3;
415   }
416   CIO::E_CIO_ARRAYSHAPE shape = src->getArrayShape();
417
418   // 成分数
419   if( src->getNcomp() != src->getNcomp() )
420   {
421     return 4;
422   }
423   int ncomp = src->getNcomp();
424
425   // コピーの範囲
426   int       gcS    = src->getGcInt();
427   const int *headS = src->getHeadIndex();
428   const int *tailS = src->getTailIndex();
429   int       gcD    = dst->getGcInt();
430   const int *headD = dst->getHeadIndex();
431   const int *tailD = dst->getTailIndex();
432   int sta[3],end[3];
433   for( int i=0;i<3;i++ )
434   {
435     sta[i] = (headS[i]-gcS>=headD[i]-gcD) ? headS[i]-gcS : headD[i]-gcD;
436     end[i] = (tailS[i]+gcS<=tailD[i]+gcD) ? tailS[i]+gcS : tailD[i]+gcD;
437   }
438   for( int i=0;i<3;i++ )
439   {
440     sta[i] = (_sta[i]>=sta[i]) ? _sta[i] : sta[i];
441     end[i] = (_end[i]<=end[i]) ? _end[i] : end[i];
442   }
443
444   // コピー
445   if( m_shape == CIO::E_CIO_IJKN )
446   {
447     for( int n=0;n<ncomp;n++ ){
448     for( int k=sta[2];k<=end[2];k++ ){
449     for( int j=sta[1];j<=end[1];j++ ){
450     for( int i=sta[0];i<=end[0];i++ ){
```

```
451       dst->hval(i,j,k,n) = src->hval(i,j,k,n);
452     }}}}
453   }
454   else
455   {
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     for( int n=0;n<ncomp;n++ ){
460       dst->hval(n,i,j,k) = src->hval(n,i,j,k);
461     }}}}
462   }
463
464   return 0;
465 }
```

6.16.3.5 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 72 行で定義されています。

参照先 cio_TypeArray< T >::m_data.

参照元 cio_Array::getData().

```
73   {
74     T *ptr = m_data;
75     if( extract )
76     {
77       m_data = NULL;
78     }
79     return ptr;
80   }
```

6.16.3.6 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 338 行で定義されています。

参照元 cio_TypeArray< T >::copyArray().

```
339 {
340   return hval(i,j,k,n);
341 }
```

6.16.3.7 template<class T > cio_TypeArray< T >::hval (int *i*, int *j*, int *k*, int *l* = 0)

cio_Array_inline.h の 327 行で定義されています。

```
328 {
329   return m_data[ m_Sz[2] * m_Sz[1] * m_Sz[0] * size_t(n-m_headIndex[3]+m_gcl[3])
330                + m_Sz[1] * m_Sz[0] * size_t(k-m_headIndex[2]+m_gcl[2])
331                          + m_Sz[0] * size_t(j-m_headIndex[1]+m_gcl[1])
332                                    + size_t(i-m_headIndex[0]+m_gcl[0]) ];
333 }
```

6.16.3.8 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 548 行で定義されています。

参照先 BSWAPVEC, DBSWAPVEC, と SBSWAPVEC.

```
549 {
550   if( !fp ) return size_t(0);
551   size_t ndata = getArrayLength();
552   size_t nread = fread(m_data,sizeof(T),ndata,fp);
553   if( !bMatchEndian )
554   {
555     size_t bsz = sizeof(T);
556     if( bsz == 2 )
557     {
558       SBSWAPVEC(m_data,nread);
559     }
560     else if( bsz == 4 )
561     {
562       BSWAPVEC(m_data,nread);
563     }
564     else if( bsz == 8 )
565     {
566       DBSWAPVEC(m_data,nread);
567     }
568   }
569   return nread;
570 }
```

6.16.3.9 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 319 行で定義されています。

```
320 {
321   return val(i,j,k,n);
322 }
```

6.16.3.10 template<class T > cio_TypeArray< T >::val (int *i*, int *j*, int *k*, int *l* = 0)

cio_Array_inline.h の 308 行で定義されています。

```
309 {
310   return m_data[ m_Sz[2] * m_Sz[1] * m_Sz[0] * size_t(n+m_gcl[3])
311                          + m_Sz[1] * m_Sz[0] * size_t(k+m_gcl[2])
312                                    + m_Sz[0] * size_t(j+m_gcl[1])
313                                              + size_t(i+m_gcl[0]) ];
314 }
```

6.16.3.11 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 574 行で定義されています。

```
575 {
576   if( !fp ) return size_t(0);
577   return fwrite(m_data,sizeof(T),getArrayLength(),fp);
578 }
```

6.16.4 変数

6.16.4.1 template<class T> T* cio_TypeArray< T >::m_data [protected]

実データ配列

cio_TypeArray.h の 136 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().

6.16.4.2 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 133 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

6.17 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

6.17.1 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。

6.17.2 コンストラクタとデストラクタ

6.17.2.1 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 }
```

6.17.2.2 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 }
```

6.17.3 関数

6.17.3.1 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 }
```

6.17.3.2 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 }
```

6.17.3.3 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 }
```

6.17.3.4 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 }
```

6.17.4 変数

6.17.4.1 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

6.18 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）**

6.18.1 説明

cio_Unit.h の 18 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 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 }
```

6.18.2.2 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 }
```

6.18.2.3 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 }
```

6.18.3 関数

6.18.3.1 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 }
```

6.18.3.2 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 }
```

6.18.4 変数

6.18.4.1 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().

6.18.4.2 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().

6.18.4.3 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().

6.18.4.4 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().

6.18.4.5 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

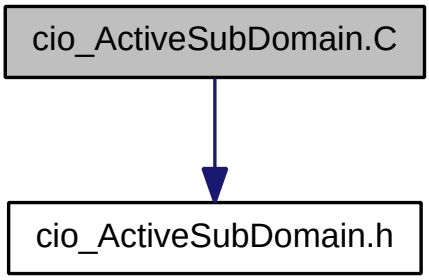
ファイル

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

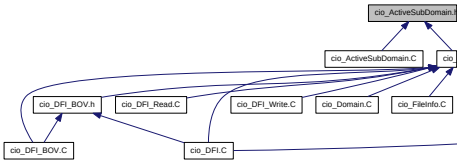
作者

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。



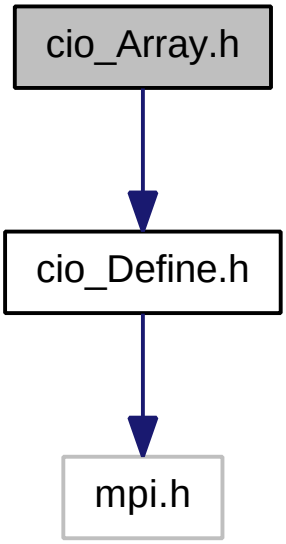
構成

- class cio_ActiveSubDomain

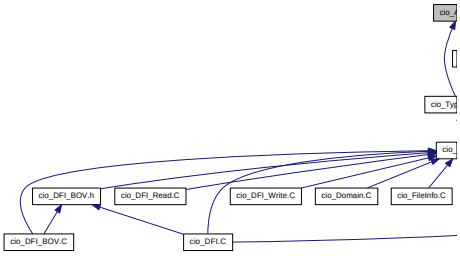
7.3 cio_Array.h

```
#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().

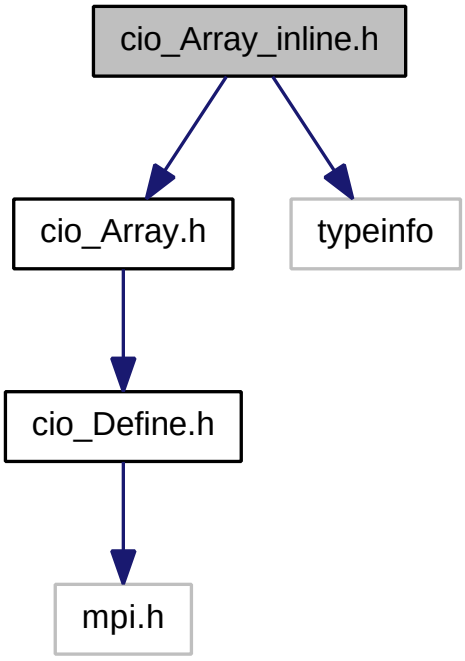
7.3.1.4 void cio_interp_nijk_r8_ (const int * *szS*, const int * *gcS*, const int * *szD*, const int * *gcD*, const int * *ncomp*, double * *src*, double * *dst*)

参照元 cio_Array::interp_coarse().

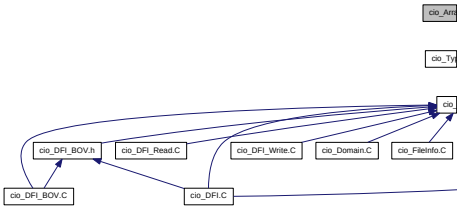
7.4 cio_Array_inline.h

```
#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 12 行で定義されています。

7.4.1.2 #define CIO_MEMFUN(*rettype*) CIO_INLINE rettype

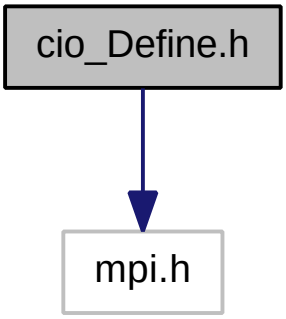
cio_Array_inline.h の 17 行で定義されています。

7.5 cio_Define.h

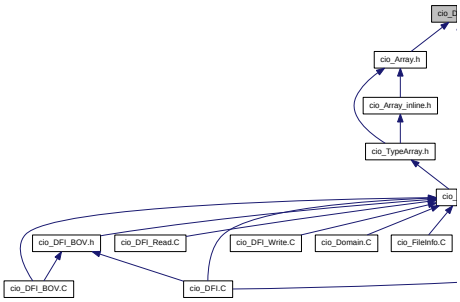
CIO の定義マクロ記述ヘッダーファイル

```
#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_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 }
- 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_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 230 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 216 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 262 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 245 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 280 行で定義されています。

7.5.2.6 #define _CIO_TAB_STR " "

cio_Define.h の 48 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 288 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 46 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().

7.5.2.10 #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().

7.5.2.11 #define D_CIO_FLOAT32 "Float32"

cio_Define.h の 39 行で定義されています。

参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.12 #define D_CIO_FLOAT64 "Float64"

cio_Define.h の 40 行で定義されています。

参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.13 #define D_CIO_IJNK "ijkn"

cio_Define.h の 42 行で定義されています。

参照元 cio_DFI::GetArrayShapeString().

7.5.2.14 #define D_CIO_INT16 "Int16"

cio_Define.h の 32 行で定義されています。

参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.15 #define D_CIO_INT32 "Int32"

cio_Define.h の 33 行で定義されています。

参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.16 #define D_CIO_INT64 "Int64"

cio_Define.h の 34 行で定義されています。
参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.17 #define D_CIO_INT8 "Int8"

cio_Define.h の 31 行で定義されています。
参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.18 #define D_CIO_LITTLE "little"

cio_Define.h の 45 行で定義されています。

7.5.2.19 #define D_CIO_NIJK "nijk"

cio_Define.h の 43 行で定義されています。
参照元 cio_DFI::GetArrayShapeString().

7.5.2.20 #define D_CIO_OFF "off"

cio_Define.h の 29 行で定義されています。

7.5.2.21 #define D_CIO_ON "on"

cio_Define.h の 28 行で定義されています。

7.5.2.22 #define D_CIO_UINT16 "UInt16"

cio_Define.h の 36 行で定義されています。
参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.23 #define D_CIO_UINT32 "UInt32"

cio_Define.h の 37 行で定義されています。
参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.24 #define D_CIO_UINT64 "UInt64"

cio_Define.h の 38 行で定義されています。
参照元 cio_DFI::ConvDatatypeE2S().

7.5.2.25 #define D_CIO_UINT8 "UInt8"

cio_Define.h の 35 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 <unistd.h>
#include "cio_DFI.h"
#include "cio_DFI_SPH.h"
#include "cio_DFI_BOV.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。

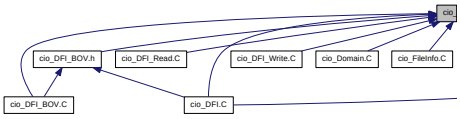
7.7 cio_DFI.h

cio_DFI Class Header

```
#include <stdlib.h>
#include <errno.h>
#include <sys/stat.h>
#include <typeinfo>
#include <set>
#include <map>
#include <string>
#include "cio_Define.h"
#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 "cio_Interval_Mngr.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_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

7.8.1 説明

cio_DFI_BOV Class

作者

kero

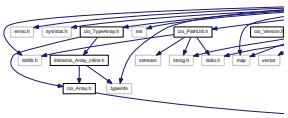
cio_DFI_BOV.C で定義されています。

7.9 cio_DFI_BOV.h

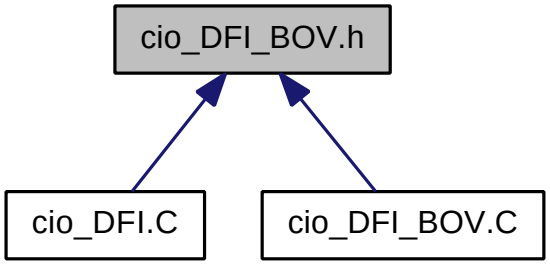
cio_DFI_BOV Class Header

```
#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

7.9.1 説明

cio_DFI_BOV Class Header

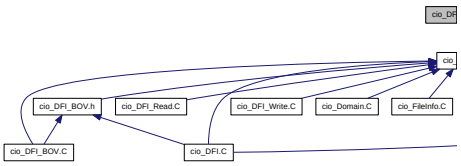
作者

kero

cio_DFI_BOV.h で定義されています。

7.10 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

7.10.1 マクロ定義

7.10.1.1 #define CIO_INLINE inline

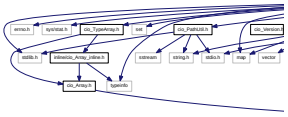
cio_DFI_inline.h の 23 行で定義されています。

7.11 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



7.11.1 説明

cio_DFI Class

作者

kero

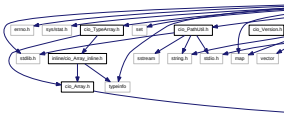
cio_DFI_Read.C で定義されています。

7.12 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



7.12.1 説明

cio_DFI_SPH Class

作者

kero

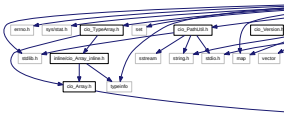
cio_DFI_SPH.C で定義されています。

7.13 cio_DFI_SPH.h

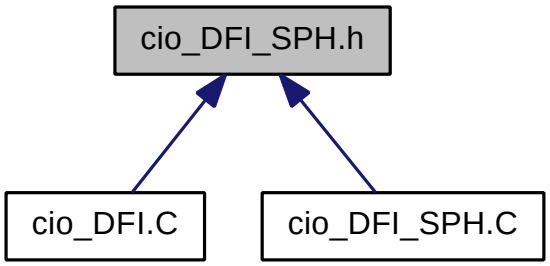
cio_DFI_SPH Class Header

```
#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

7.13.1 説明

cio_DFI_SPH Class Header

作者

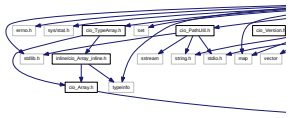
kero

cio_DFI_SPH.h で定義されています。

7.14 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



7.14.1 説明

cio_DFI Class

作者

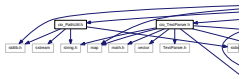
kero

cio_DFI_Write.C で定義されています。

7.15 cio_Domain.C

cio_Domain Class

```
#include <unistd.h>
#include "cio_DFI.h"
```
cio_Domain.C のインクルード依存関係図



7.15.1 説明

cio_Domain Class

作者

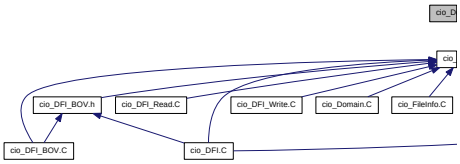
kero

cio_Domain.C で定義されています。

7.16 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

7.16.1 説明

cio_Domain Class Header

作者

kero

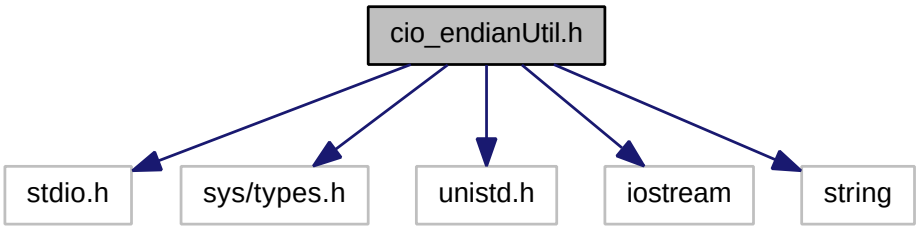
cio_Domain.h で定義されています。

7.17 cio_endianUtil.h

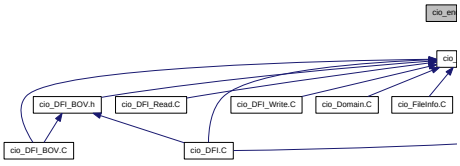
エンディアンユーティリティマクロ・関数ファイル

```
#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)

7.17.1 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。

7.17.2 マクロ定義

7.17.2.1 #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。

7.17.2.2 #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_HeaderRecord().

7.17.2.3 #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().

7.17.2.4 #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。

7.17.2.5 #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。

7.17.2.6 #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。

7.17.2.7 #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(), と cio_TypeArray< T >::readBinary().

7.17.2.8 #define CIO_INLINE inline

cio_endianUtil.h の 28 行で定義されています。

7.17.2.9 #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().

7.17.2.10 #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。

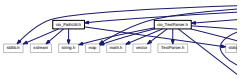
参照元 cio_TypeArray< T >::readBinary().

7.18 cio_FileInfo.C

cio_FileInfo Class

```
#include <unistd.h>
#include "cio_DFI.h"
```

cio_FileInfo.C のインクルード依存関係図



7.18.1 説明

cio_FileInfo Class

作者

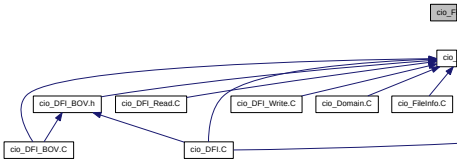
kero

cio_FileInfo.C で定義されています。

7.19 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

7.19.1 説明

cio_FileInfo Class Header

作者

kero

cio_FileInfo.h で定義されています。

7.20 cio_FilePath.C

cio_FilePath Class

```
#include <unistd.h>
#include "cio_DFI.h"
```

cio_FilePath.C のインクルード依存関係図

7.20.1 説明

cio_FilePath Class

作者

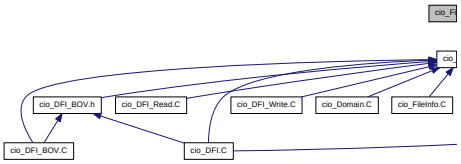
kero

cio_FilePath.C で定義されています。

7.21 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

7.21.1 説明

cio_FilePath Class Header

作者

kero

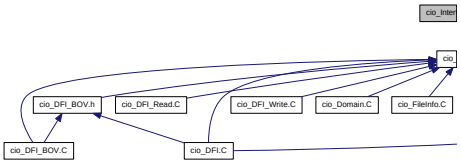
cio_FilePath.h で定義されています。

7.22 cio_interp_ijkn.h

7.23 cio_interp_nijk.h

7.24 cio_Interval_Mngr.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_Interval_Mngr

7.25 cio_MPI.C

cio_MPI Class

```
#include <unistd.h>
#include "cio_DFI.h"
```

cio_MPI.C のインクルード依存関係図

7.25.1 説明

cio_MPI Class

作者

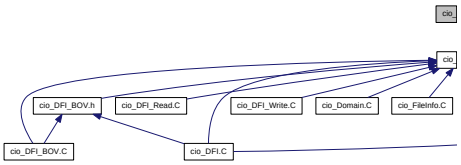
kero

cio_MPI.C で定義されています。

7.26 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

7.26.1 説明

cio_MPI Class Header

作者

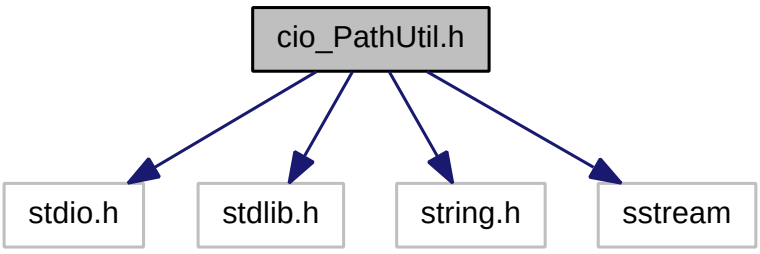
kero

cio_MPI.h で定義されています。

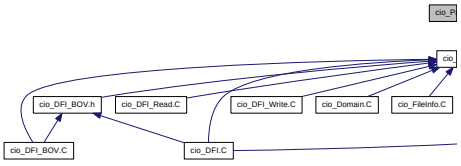
7.27 cio_PathUtil.h

```
#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)

7.27.1 マクロ定義

7.27.1.1 #define MAXPATHLEN 512

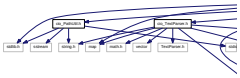
cio_PathUtil.h の 17 行で定義されています。

7.28 cio_Process.C

cio_Rank & cio_Process Class

```
#include <unistd.h>
#include "cio_DFI.h"
```

cio_Process.C のインクルード依存関係図



7.28.1 説明

cio_Rank & cio_Process Class

作者

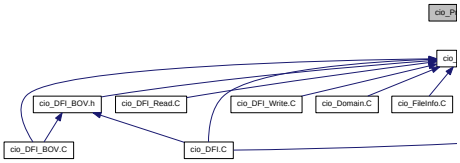
kero

cio_Process.C で定義されています。

7.29 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

7.29.1 説明

cio_RANK & cio_Process Class Header

作者

kero

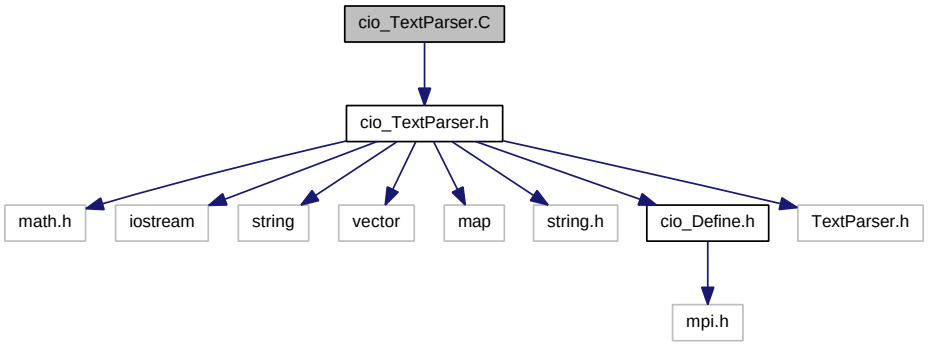
cio_Process.h で定義されています。

7.30 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



7.30.1 説明

TextParser Control class.

作者

kero

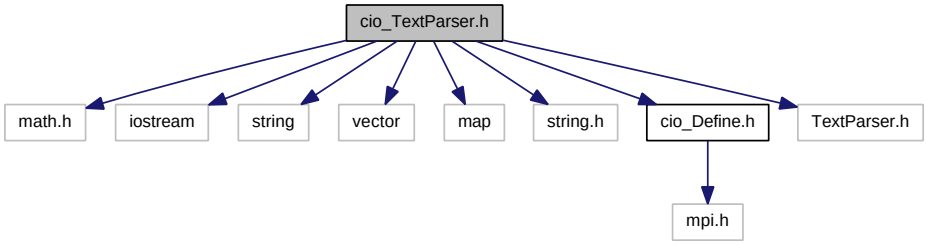
cio_TextParser.C で定義されています。

7.31 cio_TextParser.h

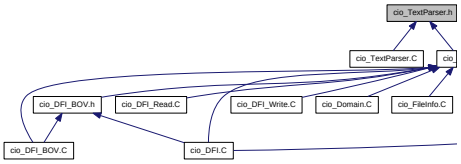
TextParser Control class Header.

```
#include <math.h>
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include "string.h"
#include "cio_Define.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

7.31.1 説明

TextParser Control class Header.

作者

kero

cio_TextParser.h で定義されています。

7.32 cio_TimeSlice.C

cio_Slice Class

```
#include <unistd.h>
#include "cio_DFI.h"
```

cio_TimeSlice.C のインクルード依存関係図

7.32.1 説明

cio_Slice Class

作者

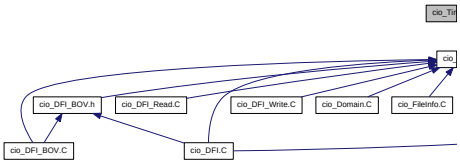
kero

cio_TimeSlice.C で定義されています。

7.33 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

7.33.1 説明

cio_Slice & cio_TimeSliceClass Header

作者

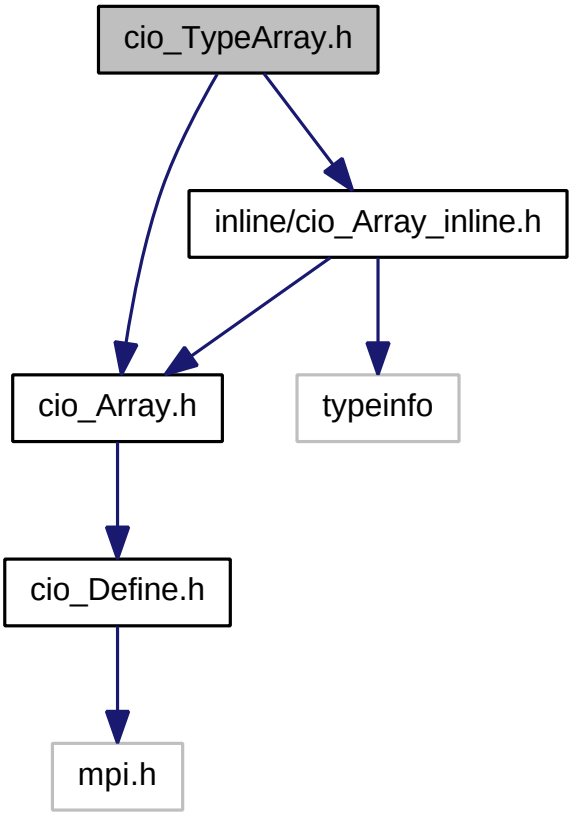
kero

cio_TimeSlice.h で定義されています。

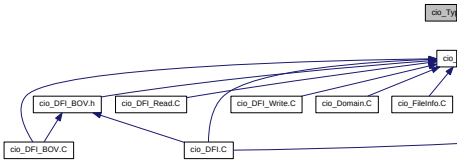
7.34 cio_TypeArray.h

```
#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

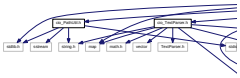
- class cio_TypeArray< T >

7.35 cio_Unit.C

cio_Unit Class

```
#include <unistd.h>
#include "cio_DFI.h"
```

cio_Unit.C のインクルード依存関係図



7.35.1 説明

cio_Unit Class

作者

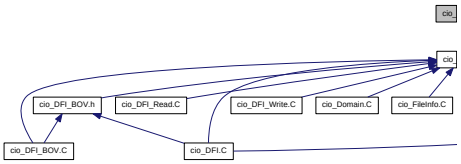
kero

cio_Unit.C で定義されています。

7.36 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

7.36.1 説明

cio_UnitElem & cio_Unit Class Header

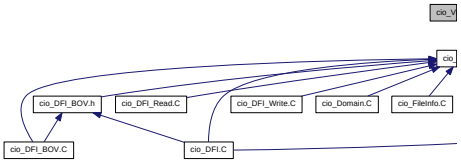
作者

kero

cio_Unit.h で定義されています。

7.37 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 "1.3.7"
- #define CIO_REVISION "20130627_2300"

7.37.1 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。

7.37.2 マクロ定義

7.37.2.1 #define CIO_REVISION "20130627_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。

7.37.2.2 #define CIO_VERSION_NO "1.3.7"

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().

7.38 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)

7.38.1 マクロ定義

7.38.1.1 #define MPI_CHAR 2

mpi_stubs.h の 8 行で定義されています。

参照元 cio_DFI::WriteProcDfiFile().

7.38.1.2 #define MPI_COMM_WORLD 0

mpi_stubs.h の 6 行で定義されています。

参照元 cio_DFI::WriteProcDfiFile().

7.38.1.3 #define MPI_INT 1

mpi_stubs.h の 7 行で定義されています。

参照元 cio_DFI::cio_Create_dfiProcessInfo().

7.38.1.4 #define MPI_SUCCESS true

mpi_stubs.h の 10 行で定義されています。

7.38.2 型定義

7.38.2.1 typedef int MPI_Comm

mpi_stubs.h の 4 行で定義されています。

7.38.2.2 typedef int MPI_Datatype

mpi_stubs.h の 5 行で定義されています。

7.38.3 関数

7.38.3.1 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 26 行で定義されています。

```
29 {
30   return 0;
31 }
```

7.38.3.2 int MPI_Comm_rank (MPI_Comm *comm*, int * *rank*) [inline]

mpi_stubs.h の 14 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().

```
15 {
16   *rank = 0;
17   return 0;
18 }
```

7.38.3.3 int MPI_Comm_size (MPI_Comm *comm*, int * *size*) [inline]

mpi_stubs.h の 20 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().

```
21 {
22   *size = 1;
23   return 0;
24 }
```

7.38.3.4 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 33 行で定義されています。

参照元 cio_DFI::cio_Create_dfiProcessInfo(), と cio_DFI::WriteProcDfiFile().

```
36 {
37   return 0;
38 }
```

7.38.3.5 bool MPI_Init (int * *argc*, char *** *argv*) [inline]

mpi_stubs.h の 12 行で定義されています。

```
12 { return true; }
```

Index

~cio_ActiveSubDomain
    cio_ActiveSubDomain, 23
~cio_Array
    cio_Array, 28
~cio_DFI
    cio_DFI, 47
~cio_DFI_BOV
    cio_DFI_BOV, 92
~cio_DFI_SPH
    cio_DFI_SPH, 101
~cio_Domain
    cio_Domain, 111
~cio_FileInfo
    cio_FileInfo, 116
~cio_FilePath
    cio_FilePath, 124
~cio_Interval_Mngr
    cio_Interval_Mngr, 128
~cio_MPI
    cio_MPI, 138
~cio_Process
    cio_Process, 141
~cio_Rank
    cio_Rank, 153
~cio_Slice
    cio_Slice, 157
~cio_TextParser
    cio_TextParser, 162
~cio_TimeSlice
    cio_TimeSlice, 172
~cio_TypeArray
    cio_TypeArray, 178
~cio_Unit
    cio_Unit, 183
~cio_UnitElem
    cio_UnitElem, 188
_CIO_IDX_IJ
    cio_Define.h, 197
_CIO_IDX_IJK
    cio_Define.h, 197
_CIO_IDX_IJKN
    cio_Define.h, 199
_CIO_IDX_NIJ
    cio_Define.h, 199
_CIO_IDX_NIJK
    cio_Define.h, 200
_CIO_TAB_STR
    cio_Define.h, 200
_CIO_WRITE_TAB
    cio_Define.h, 200
_DATA_UNKNOWN
    cio_DFI_SPH, 99
_DOUBLE
    cio_DFI_SPH, 99
_FLOAT
    cio_DFI_SPH, 99
_REAL_UNKNOWN
    cio_DFI_SPH, 99
_SCALAR
    cio_DFI_SPH, 99
_VECTOR
    cio_DFI_SPH, 99
_getArraySize
    cio_Array, 29
_getArraySizeInt
    cio_Array, 29
_val
    cio_TypeArray, 179

ActiveSubdomainFile
    cio_Domain, 113
AddSlice
    cio_TimeSlice, 172
AddUnit
    cio_DFI, 47
ArrayShape
    cio_FileInfo, 121
AveragedStep
    cio_Slice, 159
AveragedTime
    cio_Slice, 159
avr_mode
    cio_Slice, 160

BSWAP16
    cio_endianUtil.h, 210
BSWAP32
    cio_endianUtil.h, 210
BSWAP64
    cio_endianUtil.h, 210
BSWAP_X_16
    cio_endianUtil.h, 210
BSWAP_X_32
    cio_endianUtil.h, 211
BSWAP_X_64
    cio_endianUtil.h, 211
BSWAPVEC
    cio_endianUtil.h, 211
BsetDiff

cio_UnitElem, 189
By_step
cio_Interval_Mngr, 128
By_time
cio_Interval_Mngr, 128

CIO, 9
cioPath_ConnectPath, 16
cioPath_DirName, 16
cioPath_FileName, 17
cioPath_getDelimChar, 17
cioPath_getDelimString, 18
cioPath_hasDrive, 18
cioPath_isAbsolute, 18
E_CIO_ARRAYSHAPE, 10
E_CIO_ARRAYSHAPE_UNKNOWN, 10
E_CIO_BIG, 11
E_CIO_DIFFDIV_REFINEMENT, 15
E_CIO_DIFFDIV_SAMERES, 15
E_CIO_DTYPE, 11
E_CIO_DTYPE_UNKNOWN, 11
E_CIO_ENDIANTYPE, 11
E_CIO_ENDIANTYPE_UNKNOWN, 11
E_CIO_ERROR, 12
E_CIO_ERROR_INVALID_DIVNUM, 13
E_CIO_ERROR_MAKEDIRECTORY, 13
E_CIO_ERROR_MISMATCH_NP_SUBDOMAIN, 13
E_CIO_ERROR_NOMATCH_ENDIAN, 13
E_CIO_ERROR_OPEN_FIELDDATA, 13
E_CIO_ERROR_OPEN_SBDM, 13
E_CIO_ERROR_READ_BOV_FILE, 13
E_CIO_ERROR_READ_DFI_ARRAYSHAPE, 12
E_CIO_ERROR_READ_DFI_COMPONENT, 12
E_CIO_ERROR_READ_DFI_DATATYPE, 12
E_CIO_ERROR_READ_DFI_DIRECTORYPATH, 12
E_CIO_ERROR_READ_DFI_ENDIAN, 12
E_CIO_ERROR_READ_DFI_FILEFORMAT, 12
E_CIO_ERROR_READ_DFI_FILEPATH_PROCESS, 12
E_CIO_ERROR_READ_DFI_GLOBALDIVISION, 12
E_CIO_ERROR_READ_DFI_GLOBALORIGIN, 12
E_CIO_ERROR_READ_DFI_GLOBALREGION, 12
E_CIO_ERROR_READ_DFI_GLOBALVOXEL, 12
E_CIO_ERROR_READ_DFI_GUIDECELL, 12
E_CIO_ERROR_READ_DFI_HEADINDEX, 12
E_CIO_ERROR_READ_DFI_HOSTNAME, 12
E_CIO_ERROR_READ_DFI_ID, 12
E_CIO_ERROR_READ_DFI_MAX, 12
E_CIO_ERROR_READ_DFI_MIN, 12
E_CIO_ERROR_READ_DFI_NO_MINMAX, 12
E_CIO_ERROR_READ_DFI_NO_RANK, 12
E_CIO_ERROR_READ_DFI_NO_SLICE, 12
E_CIO_ERROR_READ_DFI_PREFIX, 12
E_CIO_ERROR_READ_DFI_STEP, 12
E_CIO_ERROR_READ_DFI_TAILINDEX, 12
E_CIO_ERROR_READ_DFI_TIME, 12
E_CIO_ERROR_READ_DFI_TIMESLICEDIRECTORY, 12
E_CIO_ERROR_READ_DFI_VOXELSIZE, 12
E_CIO_ERROR_READ_DOMAIN, 12
E_CIO_ERROR_READ_FIELD_AVERAGED_RECORD, 13
E_CIO_ERROR_READ_FIELD_DATA_RECORD, 13
E_CIO_ERROR_READ_FIELD_HEADER_RECORD, 13
E_CIO_ERROR_READ_FIELDDATA_FILE, 12
E_CIO_ERROR_READ_FILEINFO, 12
E_CIO_ERROR_READ_FILEPATH, 12
E_CIO_ERROR_READ_INDEXFILE_OPENERROR, 12
E_CIO_ERROR_READ_MPI, 12
E_CIO_ERROR_READ_PROCESS, 12
E_CIO_ERROR_READ_PROCFILE_OPENERROR, 12
E_CIO_ERROR_READ_SBDM_CONTENTS, 13
E_CIO_ERROR_READ_SBDM_DIV, 13
E_CIO_ERROR_READ_SBDM_FORMAT, 13
E_CIO_ERROR_READ_SBDM_HEADER, 13
E_CIO_ERROR_READ_SPH_FILE, 12
E_CIO_ERROR_READ_SPH_REC1, 12
E_CIO_ERROR_READ_SPH_REC2, 13
E_CIO_ERROR_READ_SPH_REC3, 13
E_CIO_ERROR_READ_SPH_REC4, 13
E_CIO_ERROR_READ_SPH_REC5, 13
E_CIO_ERROR_READ_SPH_REC6, 13
E_CIO_ERROR_READ_SPH_REC7, 13
E_CIO_ERROR_READ_TIMESLICE, 12
E_CIO_ERROR_READ_UNIT, 12
E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO, 13
E_CIO_ERROR_TEXTPARSER, 12
E_CIO_ERROR_UNMATCH_VOXELSIZE, 13
E_CIO_ERROR_WRITE_DOMAIN, 13
E_CIO_ERROR_WRITE_FIELD_AVERAGED_RECORD, 13
E_CIO_ERROR_WRITE_FIELD_DATA_RECORD, 13
E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD, 13
E_CIO_ERROR_WRITE_FILEINFO, 13
E_CIO_ERROR_WRITE_FILEPATH, 13
E_CIO_ERROR_WRITE_INDEXFILE_OPENERROR, 13
E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY, 13
E_CIO_ERROR_WRITE_MPI, 13
E_CIO_ERROR_WRITE_PREFIX_EMPTY, 13
E_CIO_ERROR_WRITE_PROCESS, 13
E_CIO_ERROR_WRITE_PROCFILE_OPENERROR, 13
E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY, 13
E_CIO_ERROR_WRITE_RANKID, 13

E_CIO_ERROR_WRITE_SPH_REC1, 13
E_CIO_ERROR_WRITE_SPH_REC2, 13
E_CIO_ERROR_WRITE_SPH_REC3, 13
E_CIO_ERROR_WRITE_SPH_REC4, 13
E_CIO_ERROR_WRITE_SPH_REC5, 13
E_CIO_ERROR_WRITE_SPH_REC6, 13
E_CIO_ERROR_WRITE_SPH_REC7, 13
E_CIO_ERROR_WRITE_TIMESLICE, 13
E_CIO_ERROR_WRITE_UNIT, 13
E_CIO_ERRORCODE, 11
E_CIO_FLOAT32, 11
E_CIO_FLOAT64, 11
E_CIO_FMT_BOV, 15
E_CIO_FMT_SPH, 15
E_CIO_FMT_UNKNOWN, 15
E_CIO_FORMAT, 15
E_CIO_IJKN, 10
E_CIO_INT16, 11
E_CIO_INT32, 11
E_CIO_INT64, 11
E_CIO_INT8, 11
E_CIO_LITTLE, 11
E_CIO_NIJK, 10
E_CIO_OFF, 15
E_CIO_ON, 15
E_CIO_ONOFF, 15
E_CIO_READTYPE, 15
E_CIO_READTYPE_UNKNOWN, 15
E_CIO_SAMEDIV_REFINEMENT, 15
E_CIO_SAMEDIV_SAMERES, 15
E_CIO_SUCCESS, 12
E_CIO_UINT16, 11
E_CIO_UINT32, 11
E_CIO_UINT64, 11
E_CIO_UINT8, 11
E_CIO_WARN_GETUNIT, 14
vfvPath_emitDrive, 18
CIO_INLINE
cio_Array_inline.h, 194
cio_DFI_inline.h, 206
cio_endianUtil.h, 211
CIO_MEMFUN
cio_Array_inline.h, 194
CIO_REVISION
cio_Version.h, 222
CIO_VERSION_NO
cio_Version.h, 222
calcNextStep
cio_Interval_Mngr, 128
calcNextTime
cio_Interval_Mngr, 128
CheckReadRank
cio_DFI, 47
cio_Process, 141
CheckReadType
cio_DFI, 48
CheckStartEnd
cio_Process, 142
chkLabel
cio_TextParser, 162
chkNode
cio_TextParser, 162
cio_ActiveSubDomain, 21
~cio_ActiveSubDomain, 23
cio_ActiveSubDomain, 21
cio_ActiveSubDomain, 21
clear, 23
GetPos, 23
m_pos, 25
operator==, 24
SetPos, 24
cio_ActiveSubDomain.C, 191
cio_ActiveSubDomain.h, 191
cio_Array, 25
~cio_Array, 28
_getArraySize, 29
_getArraySizeInt, 29
cio_Array, 28
cio_Array, 28
copyArray, 29
getArrayLength, 30
getArrayShape, 30
getArrayShapeString, 30
getArraySize, 30
getArraySizeInt, 31
getData, 31
getDataType, 32
getDataTypeString, 32
getGc, 33
getGcInt, 33
getHeadIndex, 33
getNcomp, 33
getNcompInt, 34
getTailIndex, 34
instanceArray, 34–37
interp_coarse, 37
m_Sz, 41
m_SzI, 41
m_dtype, 39
m_gc, 39
m_gcI, 40
m_gcl, 40
m_headIndex, 40
m_ncomp, 40
m_ncompI, 40
m_shape, 40
m_sz, 40
m_szI, 41
m_tailIndex, 41
readBinary, 39
setHeadIndex, 39
writeBinary, 39
cio_Array.h, 192
cio_interp_ijkn_r4_, 193
cio_interp_ijkn_r8_, 193
cio_interp_nijk_r4_, 193

cio_interp_nijk_r8_, 193
cio_Array_inline.h, 193
CIO_INLINE, 194
CIO_MEMFUN, 194
cio_Create_dfiProcessInfo
cio_DFI, 49
cio_DFI, 41
~cio_DFI, 47
AddUnit, 47
CheckReadRank, 47
CheckReadType, 48
cio_Create_dfiProcessInfo, 49
cio_DFI, 47
cio_DFI, 47
ConvDatatypeE2S, 49
ConvDatatypeS2E, 50
CreateReadStartEnd, 50
DFI_Domain, 88
DFI_Finfo, 88
DFI_Fpath, 88
DFI_MPI, 88
DFI_Process, 88
DFI_TimeSlice, 89
DFI_Unit, 89
Generate_DFI_Name, 52
Generate_Directory_Path, 52
Generate_FieldFileName, 53
get_cio_Datasize, 54
GetArrayShape, 54
GetArrayShapeString, 54
getComponentVariable, 57
GetDFIGlobalDivision, 58
GetDFIGlobalVoxel, 58
GetDataType, 57
GetDataTypeString, 57
getMinMax, 58
GetNumComponent, 59
GetUnit, 59
GetUnitElem, 59
getVectorMinMax, 60
getVersionInfo, 60
GetcioDomain, 55
GetcioFileInfo, 55
GetcioFilePath, 55
GetcioMPI, 56
GetcioProcess, 56
GetcioTimeSlice, 56
GetcioUnit, 56
m_RankID, 89
m_comm, 89
m_directoryPath, 89
m_indexDfiName, 89
m_intervalMngr, 89
m_read_type, 90
m_readRankList, 90
MakeDirectory, 60
MakeDirectoryPath, 61
MakeDirectorySub, 61
normalizeBaseTime, 62
normalizeDelteT, 62
normalizeIntervalTime, 62
normalizeLastTime, 62
normalizeStartTime, 63
normalizeTime, 63
read_Datarecord, 64
read_HeaderRecord, 64
read_averaged, 63
ReadData, 64, 65, 67
ReadFieldData, 69
ReadInit, 71
setComponentVariable, 74
setIntervalStep, 74
setIntervalTime, 75
SetTimeSliceFlag, 75
write_DataRecord, 77
write_HeaderRecord, 77
write_averaged, 77
WriteData, 78, 80
WriteFieldData, 81
WriteIndexDfiFile, 82
WriteInit, 83, 84
WriteProcDfiFile, 86
cio_DFI.C, 203
cio_DFI.h, 203
cio_DFI_BOV, 90
~cio_DFI_BOV, 92
cio_DFI_BOV, 92
cio_DFI_BOV, 92
read_Datarecord, 94
read_HeaderRecord, 95
read_averaged, 93
write_DataRecord, 96
write_HeaderRecord, 96
write_averaged, 95
cio_DFI_BOV.C, 204
cio_DFI_BOV.h, 205
cio_DFI_Read.C, 206
cio_DFI_SPH, 97
~cio_DFI_SPH, 101
_DATA_UNKNOWN, 99
_DOUBLE, 99
_FLOAT, 99
_REAL_UNKNOWN, 99
_SCALAR, 99
_VECTOR, 99
cio_DFI_SPH, 99
cio_DFI_SPH, 99
DataDims, 99
read_Datarecord, 102
read_HeaderRecord, 103
read_averaged, 101
RealType, 99
write_DataRecord, 107
write_HeaderRecord, 107
write_averaged, 106
cio_DFI_SPH.C, 206

cio_DFI_SPH.h, 207
cio_DFI_Write.C, 207
cio_DFI_inline.h, 205
CIO_INLINE, 206
cio_Define.h, 194
_CIO_IDX_IJ, 197
_CIO_IDX_IJK, 197
_CIO_IDX_IJKN, 199
_CIO_IDX_NIJ, 199
_CIO_IDX_NIJK, 200
_CIO_TAB_STR, 200
_CIO_WRITE_TAB, 200
D_CIO_BIG, 201
D_CIO_EXT_BOV, 201
D_CIO_EXT_SPH, 201
D_CIO_FLOAT32, 201
D_CIO_FLOAT64, 201
D_CIO_IJNK, 201
D_CIO_INT16, 201
D_CIO_INT32, 201
D_CIO_INT64, 201
D_CIO_INT8, 202
D_CIO_LITTLE, 202
D_CIO_NIJK, 202
D_CIO_OFF, 202
D_CIO_ON, 202
D_CIO_UINT16, 202
D_CIO_UINT32, 202
D_CIO_UINT64, 202
D_CIO_UINT8, 202
cio_Domain, 109
∼cio_Domain, 111
ActiveSubdomainFile, 113
cio_Domain, 110
cio_Domain, 110
GlobalDivision, 113
GlobalOrigin, 113
GlobalRegion, 113
GlobalVoxel, 114
Read, 111
Write, 112
cio_Domain.C, 208
cio_Domain.h, 208
cio_FileInfo, 114
∼cio_FileInfo, 116
ArrayShape, 121
cio_FileInfo, 115
cio_FileInfo, 115
Component, 122
ComponentVariable, 122
DataType, 122
DirectoryPath, 122
Endian, 122
FileFormat, 122
getComponentVariable, 116
GuideCell, 122
Prefix, 123
Read, 116
setComponentVariable, 120
TimeSliceDirFlag, 123
Write, 120
cio_FileInfo.C, 212
cio_FileInfo.h, 212
cio_FilePath, 123
∼cio_FilePath, 124
cio_FilePath, 124
cio_FilePath, 124
ProcDFIFile, 125
Read, 124
Write, 125
cio_FilePath.C, 213
cio_FilePath.h, 213
cio_Interval_Mngr, 126
∼cio_Interval_Mngr, 128
By_step, 128
By_time, 128
calcNextStep, 128
calcNextTime, 128
cio_Interval_Mngr, 128
cio_Interval_Mngr, 128
dmod, 129
getIntervalStep, 129
getIntervalTime, 129
getMode, 129
getStartStep, 129
getStartTime, 130
initTrigger, 130
isLastStep, 131
isLastTime, 131
isStarted, 131
isTriggered, 131
m_base_step, 135
m_base_time, 135
m_dt, 135
m_intvl_step, 135
m_intvl_time, 136
m_last_step, 136
m_last_time, 136
m_mode, 136
m_start_step, 136
m_start_time, 136
normalizeBaseTime, 132
normalizeDelteT, 133
normalizeIntervalTime, 133
normalizeLastTime, 133
normalizeStartTime, 133
normalizeTime, 133
noset, 128
setInterval, 134
setLast, 134
setMode, 134
setStart, 135
type_IO_spec, 127
cio_Interval_Mngr.h, 214
cio_MPI, 137
∼cio_MPI, 138

cio_MPI, 137
cio_MPI, 137
NumberOfGroup, 139
NumberOfRank, 139
Read, 138
Write, 138
cio_MPI.C, 214
cio_MPI.h, 215
cio_PathUtil.h, 215
MAXPATHLEN, 216
cio_Process, 139
~cio_Process, 141
CheckReadRank, 141
CheckStartEnd, 142
cio_Process, 141
cio_Process, 141
CreateHeadMap, 143, 144
CreateRankList, 144, 145
CreateRankMap, 146, 147
CreateSubDomainInfo, 148
headT, 141
isMatchEndianSbdmMagick, 148
m_rankMap, 151
RankList, 152
Read, 149
ReadActiveSubdomainFile, 150
Write, 151
cio_Process.C, 216
cio_Process.h, 217
cio_Rank, 152
~cio_Rank, 153
cio_Rank, 153
cio_Rank, 153
HeadIndex, 155
HostName, 155
RankID, 155
Read, 153
TailIndex, 155
VoxelSize, 155
Write, 154
cio_Slice, 156
~cio_Slice, 157
AveragedStep, 159
AveragedTime, 159
avr_mode, 160
cio_Slice, 156
cio_Slice, 156
Max, 160
Min, 160
Read, 157
step, 160
time, 160
VectorMax, 160
VectorMin, 160
Write, 159
cio_TextParser, 161
~cio_TextParser, 162
chkLabel, 162
chkNode, 162
cio_TextParser, 162
cio_TextParser, 162
countLabels, 164
GetNodeStr, 165
getTPinstance, 166
GetValue, 166–168
GetVector, 168–170
readTPfile, 170
remove, 171
tp, 171
cio_TextParser.C, 217
cio_TextParser.h, 218
cio_TimeSlice, 171
~cio_TimeSlice, 172
AddSlice, 172
cio_TimeSlice, 172
cio_TimeSlice, 172
getMinMax, 173
getVectorMinMax, 174
Read, 174
SliceList, 176
Write, 175
cio_TimeSlice.C, 218
cio_TimeSlice.h, 219
cio_TypeArray
~cio_TypeArray, 178
_val, 179
cio_TypeArray, 178, 179
cio_TypeArray, 178, 179
copyArray, 179, 180
getData, 181
hval, 181
m_data, 182
m_outptr, 182
readBinary, 181
val, 182
writeBinary, 182
cio_TypeArray< T >, 176
cio_TypeArray.h, 219
cio_Unit, 183
~cio_Unit, 183
cio_Unit, 183
cio_Unit, 183
GetUnit, 184
GetUnitElem, 184
Read, 185
UnitList, 186
Write, 186
cio_Unit.C, 220
cio_Unit.h, 221
cio_UnitElem, 187
~cio_UnitElem, 188
BsetDiff, 189
cio_UnitElem, 187
cio_UnitElem, 187
difference, 189
Name, 189

Read, 188
reference, 190
Unit, 190
Write, 189
cio_Version.h, 221
CIO_REVISION, 222
CIO_VERSION_NO, 222
cio_endianUtil.h, 209
BSWAP16, 210
BSWAP32, 210
BSWAP64, 210
BSWAP_X_16, 210
BSWAP_X_32, 211
BSWAP_X_64, 211
BSWAPVEC, 211
CIO_INLINE, 211
DBSWAPVEC, 211
SBSWAPVEC, 211
cio_interp_ijkn.h, 214
cio_interp_ijkn_r4_
cio_Array.h, 193
cio_interp_ijkn_r8_
cio_Array.h, 193
cio_interp_nijk.h, 214
cio_interp_nijk_r4_
cio_Array.h, 193
cio_interp_nijk_r8_
cio_Array.h, 193
cioPath_ConnectPath
CIO, 16
cioPath_DirName
CIO, 16
cioPath_FileName
CIO, 17
cioPath_getDelimChar
CIO, 17
cioPath_getDelimString
CIO, 18
cioPath_hasDrive
CIO, 18
cioPath_isAbsolute
CIO, 18
clear
cio_ActiveSubDomain, 23
Component
cio_FileInfo, 122
ComponentVariable
cio_FileInfo, 122
ConvDatatypeE2S
cio_DFI, 49
ConvDatatypeS2E
cio_DFI, 50
copyArray
cio_Array, 29
cio_TypeArray, 179, 180
countLabels
cio_TextParser, 164
CreateHeadMap
cio_Process, 143, 144
CreateRankList
cio_Process, 144, 145
CreateRankMap
cio_Process, 146, 147
CreateReadStartEnd
cio_DFI, 50
CreateSubDomainInfo
cio_Process, 148

D_CIO_BIG
cio_Define.h, 201
D_CIO_EXT_BOV
cio_Define.h, 201
D_CIO_EXT_SPH
cio_Define.h, 201
D_CIO_FLOAT32
cio_Define.h, 201
D_CIO_FLOAT64
cio_Define.h, 201
D_CIO_IJNK
cio_Define.h, 201
D_CIO_INT16
cio_Define.h, 201
D_CIO_INT32
cio_Define.h, 201
D_CIO_INT64
cio_Define.h, 201
D_CIO_INT8
cio_Define.h, 202
D_CIO_LITTLE
cio_Define.h, 202
D_CIO_NIJK
cio_Define.h, 202
D_CIO_OFF
cio_Define.h, 202
D_CIO_ON
cio_Define.h, 202
D_CIO_UINT16
cio_Define.h, 202
D_CIO_UINT32
cio_Define.h, 202
D_CIO_UINT64
cio_Define.h, 202
D_CIO_UINT8
cio_Define.h, 202
DBSWAPVEC
cio_endianUtil.h, 211
DFI_Domain
cio_DFI, 88
DFI_Finfo
cio_DFI, 88
DFI_Fpath
cio_DFI, 88
DFI_MPI
cio_DFI, 88
DFI_Process
cio_DFI, 88
DFI_TimeSlice

cio_DFI, 89
DFI_Unit
cio_DFI, 89
DataDims
cio_DFI_SPH, 99
DataType
cio_FileInfo, 122
difference
cio_UnitElem, 189
DirectoryPath
cio_FileInfo, 122
dmod
cio_Interval_Mngr, 129

E_CIO_ARRAYSHAPE
CIO, 10
E_CIO_ARRAYSHAPE_UNKNOWN
CIO, 10
E_CIO_BIG
CIO, 11
E_CIO_DIFFDIV_REFINEMENT
CIO, 15
E_CIO_DIFFDIV_SAMERES
CIO, 15
E_CIO_DTYPE
CIO, 11
E_CIO_DTYPE_UNKNOWN
CIO, 11
E_CIO_ENDIANTYPE
CIO, 11
E_CIO_ENDIANTYPE_UNKNOWN
CIO, 11
E_CIO_ERROR
CIO, 12
E_CIO_ERROR_INVALID_DIVNUM
CIO, 13
E_CIO_ERROR_MAKEDIRECTORY
CIO, 13
E_CIO_ERROR_MISMATCH_NP_SUBDOMAIN
CIO, 13
E_CIO_ERROR_NOMATCH_ENDIAN
CIO, 13
E_CIO_ERROR_OPEN_FIELDDATA
CIO, 13
E_CIO_ERROR_OPEN_SBDM
CIO, 13
E_CIO_ERROR_READ_BOV_FILE
CIO, 13
E_CIO_ERROR_READ_DFI_ARRAYSHAPE
CIO, 12
E_CIO_ERROR_READ_DFI_COMPONENT
CIO, 12
E_CIO_ERROR_READ_DFI_DATATYPE
CIO, 12
E_CIO_ERROR_READ_DFI_DIRECTORYPATH
CIO, 12
E_CIO_ERROR_READ_DFI_ENDIAN
CIO, 12
E_CIO_ERROR_READ_DFI_FILEFORMAT
CIO, 12
E_CIO_ERROR_READ_DFI_FILEPATH_PROCESS
CIO, 12
E_CIO_ERROR_READ_DFI_GLOBALDIVISION
CIO, 12
E_CIO_ERROR_READ_DFI_GLOBALORIGIN
CIO, 12
E_CIO_ERROR_READ_DFI_GLOBALREGION
CIO, 12
E_CIO_ERROR_READ_DFI_GLOBALVOXEL
CIO, 12
E_CIO_ERROR_READ_DFI_GUIDECELL
CIO, 12
E_CIO_ERROR_READ_DFI_HEADINDEX
CIO, 12
E_CIO_ERROR_READ_DFI_HOSTNAME
CIO, 12
E_CIO_ERROR_READ_DFI_ID
CIO, 12
E_CIO_ERROR_READ_DFI_MAX
CIO, 12
E_CIO_ERROR_READ_DFI_MIN
CIO, 12
E_CIO_ERROR_READ_DFI_NO_MINMAX
CIO, 12
E_CIO_ERROR_READ_DFI_NO_RANK
CIO, 12
E_CIO_ERROR_READ_DFI_NO_SLICE
CIO, 12
E_CIO_ERROR_READ_DFI_PREFIX
CIO, 12
E_CIO_ERROR_READ_DFI_STEP
CIO, 12
E_CIO_ERROR_READ_DFI_TAILINDEX
CIO, 12
E_CIO_ERROR_READ_DFI_TIME
CIO, 12
E_CIO_ERROR_READ_DFI_TIMESLICEDIRECTORY
CIO, 12
E_CIO_ERROR_READ_DFI_VOXELSIZE
CIO, 12
E_CIO_ERROR_READ_DOMAIN
CIO, 12
E_CIO_ERROR_READ_FIELD_AVERAGED_RECORD
CIO, 13
E_CIO_ERROR_READ_FIELD_DATA_RECORD
CIO, 13
E_CIO_ERROR_READ_FIELD_HEADER_RECORD
CIO, 13
E_CIO_ERROR_READ_FIELDDATA_FILE
CIO, 12
E_CIO_ERROR_READ_FILEINFO
CIO, 12
E_CIO_ERROR_READ_FILEPATH
CIO, 12
E_CIO_ERROR_READ_INDEXFILE_OPENERROR
CIO, 12

E_CIO_ERROR_READ_MPI
CIO, 12
E_CIO_ERROR_READ_PROCESS
CIO, 12
E_CIO_ERROR_READ_PROCFILE_OPENERROR
CIO, 12
E_CIO_ERROR_READ_SBDM_CONTENTS
CIO, 13
E_CIO_ERROR_READ_SBDM_DIV
CIO, 13
E_CIO_ERROR_READ_SBDM_FORMAT
CIO, 13
E_CIO_ERROR_READ_SBDM_HEADER
CIO, 13
E_CIO_ERROR_READ_SPH_FILE
CIO, 12
E_CIO_ERROR_READ_SPH_REC1
CIO, 12
E_CIO_ERROR_READ_SPH_REC2
CIO, 13
E_CIO_ERROR_READ_SPH_REC3
CIO, 13
E_CIO_ERROR_READ_SPH_REC4
CIO, 13
E_CIO_ERROR_READ_SPH_REC5
CIO, 13
E_CIO_ERROR_READ_SPH_REC6
CIO, 13
E_CIO_ERROR_READ_SPH_REC7
CIO, 13
E_CIO_ERROR_READ_TIMESLICE
CIO, 12
E_CIO_ERROR_READ_UNIT
CIO, 12
E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO
CIO, 13
E_CIO_ERROR_TEXTPARSER
CIO, 12
E_CIO_ERROR_UNMATCH_VOXELSIZE
CIO, 13
E_CIO_ERROR_WRITE_DOMAIN
CIO, 13
E_CIO_ERROR_WRITE_FIELD_AVERAGED_RECORD
CIO, 13
E_CIO_ERROR_WRITE_FIELD_DATA_RECORD
CIO, 13
E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD
CIO, 13
E_CIO_ERROR_WRITE_FILEINFO
CIO, 13
E_CIO_ERROR_WRITE_FILEPATH
CIO, 13
E_CIO_ERROR_WRITE_INDEXFILE_OPENERROR
CIO, 13
E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY
CIO, 13
E_CIO_ERROR_WRITE_MPI
CIO, 13
E_CIO_ERROR_WRITE_PREFIX_EMPTY
CIO, 13
E_CIO_ERROR_WRITE_PROCESS
CIO, 13
E_CIO_ERROR_WRITE_PROCFILE_OPENERROR
CIO, 13
E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY
CIO, 13
E_CIO_ERROR_WRITE_RANKID
CIO, 13
E_CIO_ERROR_WRITE_SPH_REC1
CIO, 13
E_CIO_ERROR_WRITE_SPH_REC2
CIO, 13
E_CIO_ERROR_WRITE_SPH_REC3
CIO, 13
E_CIO_ERROR_WRITE_SPH_REC4
CIO, 13
E_CIO_ERROR_WRITE_SPH_REC5
CIO, 13
E_CIO_ERROR_WRITE_SPH_REC6
CIO, 13
E_CIO_ERROR_WRITE_SPH_REC7
CIO, 13
E_CIO_ERROR_WRITE_TIMESLICE
CIO, 13
E_CIO_ERROR_WRITE_UNIT
CIO, 13
E_CIO_ERRORCODE
CIO, 11
E_CIO_FLOAT32
CIO, 11
E_CIO_FLOAT64
CIO, 11
E_CIO_FMT_BOV
CIO, 15
E_CIO_FMT_SPH
CIO, 15
E_CIO_FMT_UNKNOWN
CIO, 15
E_CIO_FORMAT
CIO, 15
E_CIO_IJKN
CIO, 10
E_CIO_INT16
CIO, 11
E_CIO_INT32
CIO, 11
E_CIO_INT64
CIO, 11
E_CIO_INT8
CIO, 11
E_CIO_LITTLE
CIO, 11
E_CIO_NIJK
CIO, 10
E_CIO_OFF

CIO, 15
E_CIO_ON
CIO, 15
E_CIO_ONOFF
CIO, 15
E_CIO_READTYPE
CIO, 15
E_CIO_READTYPE_UNKNOWN
CIO, 15
E_CIO_SAMEDIV_REFINEMENT
CIO, 15
E_CIO_SAMEDIV_SAMERES
CIO, 15
E_CIO_SUCCESS
CIO, 12
E_CIO_UINT16
CIO, 11
E_CIO_UINT32
CIO, 11
E_CIO_UINT64
CIO, 11
E_CIO_UINT8
CIO, 11
E_CIO_WARN_GETUNIT
CIO, 14
Endian
cio_FileInfo, 122

FileFormat
cio_FileInfo, 122

Generate_DFI_Name
cio_DFI, 52
Generate_Directory_Path
cio_DFI, 52
Generate_FieldFileName
cio_DFI, 53
get_cio_Datasize
cio_DFI, 54
getArrayLength
cio_Array, 30
GetArrayShape
cio_DFI, 54
getArrayShape
cio_Array, 30
GetArrayShapeString
cio_DFI, 54
getArrayShapeString
cio_Array, 30
getArraySize
cio_Array, 30
getArraySizeInt
cio_Array, 31
getComponentVariable
cio_DFI, 57
cio_FileInfo, 116
GetDFIGlobalDivision
cio_DFI, 58
GetDFIGlobalVoxel
cio_DFI, 58
getData
cio_Array, 31
cio_TypeArray, 181
GetDataType
cio_DFI, 57
getDataType
cio_Array, 32
GetDataTypeString
cio_DFI, 57
getDataTypeString
cio_Array, 32
getGc
cio_Array, 33
getGcInt
cio_Array, 33
getHeadIndex
cio_Array, 33
getIntervalStep
cio_Interval_Mngr, 129
getIntervalTime
cio_Interval_Mngr, 129
getMinMax
cio_DFI, 58
cio_TimeSlice, 173
getMode
cio_Interval_Mngr, 129
getNcomp
cio_Array, 33
getNcompInt
cio_Array, 34
GetNodeStr
cio_TextParser, 165
GetNumComponent
cio_DFI, 59
GetPos
cio_ActiveSubDomain, 23
getStartStep
cio_Interval_Mngr, 129
getStartTime
cio_Interval_Mngr, 130
getTPinstance
cio_TextParser, 166
getTailIndex
cio_Array, 34
GetUnit
cio_DFI, 59
cio_Unit, 184
GetUnitElem
cio_DFI, 59
cio_Unit, 184
GetValue
cio_TextParser, 166–168
GetVector
cio_TextParser, 168–170
getVectorMinMax
cio_DFI, 60
cio_TimeSlice, 174

getVersionInfo
cio_DFI, 60
GetcioDomain
cio_DFI, 55
GetcioFileInfo
cio_DFI, 55
GetcioFilePath
cio_DFI, 55
GetcioMPI
cio_DFI, 56
GetcioProcess
cio_DFI, 56
GetcioTimeSlice
cio_DFI, 56
GetcioUnit
cio_DFI, 56
GlobalDivision
cio_Domain, 113
GlobalOrigin
cio_Domain, 113
GlobalRegion
cio_Domain, 113
GlobalVoxel
cio_Domain, 114
GuideCell
cio_FileInfo, 122

HeadIndex
cio_Rank, 155
headT
cio_Process, 141
HostName
cio_Rank, 155
hval
cio_TypeArray, 181

initTrigger
cio_Interval_Mngr, 130
instanceArray
cio_Array, 34–37
interp_coarse
cio_Array, 37
isLastStep
cio_Interval_Mngr, 131
isLastTime
cio_Interval_Mngr, 131
isMatchEndianSbdmMagick
cio_Process, 148
isStarted
cio_Interval_Mngr, 131
isTriggered
cio_Interval_Mngr, 131

m_RankID
cio_DFI, 89
m_Sz
cio_Array, 41
m_SzI
cio_Array, 41
m_base_step
cio_Interval_Mngr, 135
m_base_time
cio_Interval_Mngr, 135
m_comm
cio_DFI, 89
m_data
cio_TypeArray, 182
m_directoryPath
cio_DFI, 89
m_dt
cio_Interval_Mngr, 135
m_dtype
cio_Array, 39
m_gc
cio_Array, 39
m_gcI
cio_Array, 40
m_gcl
cio_Array, 40
m_headIndex
cio_Array, 40
m_indexDfiName
cio_DFI, 89
m_intervalMngr
cio_DFI, 89
m_intvl_step
cio_Interval_Mngr, 135
m_intvl_time
cio_Interval_Mngr, 136
m_last_step
cio_Interval_Mngr, 136
m_last_time
cio_Interval_Mngr, 136
m_mode
cio_Interval_Mngr, 136
m_ncomp
cio_Array, 40
m_ncompI
cio_Array, 40
m_outptr
cio_TypeArray, 182
m_pos
cio_ActiveSubDomain, 25
m_rankMap
cio_Process, 151
m_read_type
cio_DFI, 90
m_readRankList
cio_DFI, 90
m_shape
cio_Array, 40
m_start_step
cio_Interval_Mngr, 136
m_start_time
cio_Interval_Mngr, 136
m_sz
cio_Array, 40

m_szI
cio_Array, 41
m_tailIndex
cio_Array, 41
MAXPATHLEN
cio_PathUtil.h, 216
MPI_Allgather
mpi_stubs.h, 223
MPI_CHAR
mpi_stubs.h, 222
MPI_COMM_WORLD
mpi_stubs.h, 222
MPI_Comm
mpi_stubs.h, 223
MPI_Comm_rank
mpi_stubs.h, 223
MPI_Comm_size
mpi_stubs.h, 223
MPI_Datatype
mpi_stubs.h, 223
MPI_Gather
mpi_stubs.h, 224
MPI_INT
mpi_stubs.h, 223
MPI_Init
mpi_stubs.h, 224
MPI_SUCCESS
mpi_stubs.h, 223
MakeDirectory
cio_DFI, 60
MakeDirectoryPath
cio_DFI, 61
MakeDirectorySub
cio_DFI, 61
Max
cio_Slice, 160
Min
cio_Slice, 160
mpi_stubs.h, 222
MPI_Allgather, 223
MPI_CHAR, 222
MPI_COMM_WORLD, 222
MPI_Comm, 223
MPI_Comm_rank, 223
MPI_Comm_size, 223
MPI_Datatype, 223
MPI_Gather, 224
MPI_INT, 223
MPI_Init, 224
MPI_SUCCESS, 223

Name
cio_UnitElem, 189
normalizeBaseTime
cio_DFI, 62
cio_Interval_Mngr, 132
normalizeDelteT
cio_DFI, 62
cio_Interval_Mngr, 133
normalizeIntervalTime
cio_DFI, 62
cio_Interval_Mngr, 133
normalizeLastTime
cio_DFI, 62
cio_Interval_Mngr, 133
normalizeStartTime
cio_DFI, 63
cio_Interval_Mngr, 133
normalizeTime
cio_DFI, 63
cio_Interval_Mngr, 133
noset
cio_Interval_Mngr, 128
NumberOfGroup
cio_MPI, 139
NumberOfRank
cio_MPI, 139

operator==
cio_ActiveSubDomain, 24

Prefix
cio_FileInfo, 123
ProcDFIFile
cio_FilePath, 125

RankID
cio_Rank, 155
RankList
cio_Process, 152
Read
cio_Domain, 111
cio_FileInfo, 116
cio_FilePath, 124
cio_MPI, 138
cio_Process, 149
cio_Rank, 153
cio_Slice, 157
cio_TimeSlice, 174
cio_Unit, 185
cio_UnitElem, 188
read_Datarecord
cio_DFI, 64
cio_DFI_BOV, 94
cio_DFI_SPH, 102
read_HeaderRecord
cio_DFI, 64
cio_DFI_BOV, 95
cio_DFI_SPH, 103
read_averaged
cio_DFI, 63
cio_DFI_BOV, 93
cio_DFI_SPH, 101
ReadActiveSubdomainFile
cio_Process, 150
readBinary
cio_Array, 39
cio_TypeArray, 181

ReadData
cio_DFI, 64, 65, 67
ReadFieldData
cio_DFI, 69
ReadInit
cio_DFI, 71
readTPfile
cio_TextParser, 170
RealType
cio_DFI_SPH, 99
reference
cio_UnitElem, 190
remove
cio_TextParser, 171

SBSWAPVEC
cio_endianUtil.h, 211
setComponentVariable
cio_DFI, 74
cio_FileInfo, 120
setHeadIndex
cio_Array, 39
setInterval
cio_Interval_Mngr, 134
setIntervalStep
cio_DFI, 74
setIntervalTime
cio_DFI, 75
setLast
cio_Interval_Mngr, 134
setMode
cio_Interval_Mngr, 134
SetPos
cio_ActiveSubDomain, 24
setStart
cio_Interval_Mngr, 135
SetTimeSliceFlag
cio_DFI, 75
SliceList
cio_TimeSlice, 176
step
cio_Slice, 160

TailIndex
cio_Rank, 155
time
cio_Slice, 160
TimeSliceDirFlag
cio_FileInfo, 123
tp
cio_TextParser, 171
type_IO_spec
cio_Interval_Mngr, 127

Unit
cio_UnitElem, 190
UnitList
cio_Unit, 186

val
cio_TypeArray, 182
VectorMax
cio_Slice, 160
VectorMin
cio_Slice, 160
vfvPath_emitDrive
CIO, 18
VoxelSize
cio_Rank, 155

Write
cio_Domain, 112
cio_FileInfo, 120
cio_FilePath, 125
cio_MPI, 138
cio_Process, 151
cio_Rank, 154
cio_Slice, 159
cio_TimeSlice, 175
cio_Unit, 186
cio_UnitElem, 189
write_DataRecord
cio_DFI, 77
cio_DFI_BOV, 96
cio_DFI_SPH, 107
write_HeaderRecord
cio_DFI, 77
cio_DFI_BOV, 96
cio_DFI_SPH, 107
write_averaged
cio_DFI, 77
cio_DFI_BOV, 95
cio_DFI_SPH, 106
writeBinary
cio_Array, 39
cio_TypeArray, 182
WriteData
cio_DFI, 78, 80
WriteFieldData
cio_DFI, 81
WriteIndexDfiFile
cio_DFI, 82
WriteInit
cio_DFI, 83, 84
WriteProcDfiFile
cio_DFI, 86