

Cartesian Input/Output Library  
1.3.6

作成 : Doxygen 1.8.4

Wed Sep 11 2013 14:54:15



# Contents

<b>1</b>	<b>ネームスペース索引</b>	<b>1</b>
1.1	ネームスペース一覧	1
<b>2</b>	<b>階層索引</b>	<b>3</b>
2.1	クラス階層	3
<b>3</b>	<b>構成索引</b>	<b>5</b>
3.1	構成	5
<b>4</b>	<b>ファイル索引</b>	<b>7</b>
4.1	ファイル一覧	7
<b>5</b>	<b>ネームスペース</b>	<b>9</b>
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

<b>6 クラス</b>	<b>21</b>
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 getData Type	32
6.2.3.12 getData TypeString	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	<a href="#">instanceArray</a>	34
6.2.3.20	<a href="#">instanceArray</a>	35
6.2.3.21	<a href="#">instanceArray</a>	35
6.2.3.22	<a href="#">instanceArray</a>	35
6.2.3.23	<a href="#">instanceArray</a>	36
6.2.3.24	<a href="#">instanceArray</a>	36
6.2.3.25	<a href="#">instanceArray</a>	36
6.2.3.26	<a href="#">instanceArray</a>	36
6.2.3.27	<a href="#">instanceArray</a>	37
6.2.3.28	<a href="#">instanceArray</a>	37
6.2.3.29	<a href="#">instanceArray</a>	37
6.2.3.30	<a href="#">instanceArray</a>	37
6.2.3.31	<a href="#">interp_coarse</a>	38
6.2.3.32	<a href="#">readBinary</a>	39
6.2.3.33	<a href="#">setHeadIndex</a>	39
6.2.3.34	<a href="#">writeBinary</a>	39
6.2.4	<a href="#">変数</a>	39
6.2.4.1	<a href="#">m_dtype</a>	39
6.2.4.2	<a href="#">m_gc</a>	40
6.2.4.3	<a href="#">m_gcl</a>	40
6.2.4.4	<a href="#">m_gcl</a>	40
6.2.4.5	<a href="#">m_headIndex</a>	40
6.2.4.6	<a href="#">m_ncomp</a>	40
6.2.4.7	<a href="#">m_ncompl</a>	40
6.2.4.8	<a href="#">m_shape</a>	40
6.2.4.9	<a href="#">m_sz</a>	40
6.2.4.10	<a href="#">m_Sz</a>	41
6.2.4.11	<a href="#">m_szl</a>	41
6.2.4.12	<a href="#">m_Szl</a>	41
6.2.4.13	<a href="#">m_tailIndex</a>	41
6.3	<a href="#">クラス cio_DFI</a>	41
6.3.1	<a href="#">説明</a>	46
6.3.2	<a href="#">コンストラクタとデストラクタ</a>	46
6.3.2.1	<a href="#">cio_DFI</a>	46
6.3.2.2	<a href="#">~cio_DFI</a>	47
6.3.3	<a href="#">関数</a>	47
6.3.3.1	<a href="#">AddUnit</a>	47
6.3.3.2	<a href="#">CheckReadRank</a>	47
6.3.3.3	<a href="#">CheckReadType</a>	48
6.3.3.4	<a href="#">cio_Create_dfiProcessInfo</a>	48

6.3.3.5	ConvDatatypeE2S . . . . .	49
6.3.3.6	ConvDatatypeS2E . . . . .	50
6.3.3.7	CreateReadStartEnd . . . . .	50
6.3.3.8	Generate_DFI_Name . . . . .	52
6.3.3.9	Generate_Directory_Path . . . . .	53
6.3.3.10	Generate_FieldFileName . . . . .	54
6.3.3.11	get_cio_Datasize . . . . .	55
6.3.3.12	GetArrayShape . . . . .	56
6.3.3.13	GetArrayShapeString . . . . .	56
6.3.3.14	GetcioDomain . . . . .	57
6.3.3.15	GetcioFileInfo . . . . .	57
6.3.3.16	GetcioFilePath . . . . .	57
6.3.3.17	GetcioMPI . . . . .	57
6.3.3.18	GetcioProcess . . . . .	58
6.3.3.19	GetcioTimeSlice . . . . .	58
6.3.3.20	GetcioUnit . . . . .	58
6.3.3.21	GetComponentVariable . . . . .	58
6.3.3.22	GetDataType . . . . .	59
6.3.3.23	GetDataTimeString . . . . .	59
6.3.3.24	GetDFIGlobalDivision . . . . .	59
6.3.3.25	GetDFIGlobalVoxel . . . . .	60
6.3.3.26	getMinMax . . . . .	60
6.3.3.27	GetNumComponent . . . . .	60
6.3.3.28	getVectorMinMax . . . . .	60
6.3.3.29	getVersionInfo . . . . .	61
6.3.3.30	MakeDirectory . . . . .	61
6.3.3.31	MakeDirectoryPath . . . . .	62
6.3.3.32	MakeDirectorySub . . . . .	62
6.3.3.33	normalizeBaseTime . . . . .	62
6.3.3.34	normalizeDelteT . . . . .	63
6.3.3.35	normalizeIntervalTime . . . . .	63
6.3.3.36	normalizeLastTime . . . . .	63
6.3.3.37	normalizeStartTime . . . . .	63
6.3.3.38	normalizeTime . . . . .	64
6.3.3.39	read_averaged . . . . .	64
6.3.3.40	read_Datarecord . . . . .	64
6.3.3.41	read_HeaderRecord . . . . .	65
6.3.3.42	ReadData . . . . .	65
6.3.3.43	ReadData . . . . .	66
6.3.3.44	ReadData . . . . .	66

6.3.3.45	ReadData	67
6.3.3.46	ReadData	67
6.3.3.47	ReadFieldData	69
6.3.3.48	ReadInit	71
6.3.3.49	setComponentVariable	74
6.3.3.50	setIntervalStep	74
6.3.3.51	setIntervalTime	75
6.3.3.52	SetTimeSliceFlag	75
6.3.3.53	write_averaged	76
6.3.3.54	write_DataRecord	76
6.3.3.55	write_HeaderRecord	76
6.3.3.56	WriteData	77
6.3.3.57	WriteData	77
6.3.3.58	WriteData	79
6.3.3.59	WriteFieldData	80
6.3.3.60	WriteIndexDfiFile	81
6.3.3.61	WriteInit	82
6.3.3.62	WriteInit	83
6.3.3.63	WriteProcDfiFile	85
6.3.3.64	WriteProcDfiFile	85
6.3.4	変数	87
6.3.4.1	DFI_Domain	87
6.3.4.2	DFI_Finfo	87
6.3.4.3	DFI_Fpath	87
6.3.4.4	DFI_MPI	87
6.3.4.5	DFI_Process	87
6.3.4.6	DFI_TimeSlice	88
6.3.4.7	DFI_Unit	88
6.3.4.8	m_comm	88
6.3.4.9	m_directoryPath	88
6.3.4.10	m_indexDfiName	88
6.3.4.11	m_intervalMgr	88
6.3.4.12	m_RankID	88
6.3.4.13	m_read_type	89
6.3.4.14	m_readRankList	89
6.4	クラス cio_DFI_BOV	89
6.4.1	説明	91
6.4.2	コンストラクタとデストラクタ	91
6.4.2.1	cio_DFI_BOV	91
6.4.2.2	cio_DFI_BOV	91

6.4.2.3	~cio_DFI_BOV	91
6.4.3	関数	92
6.4.3.1	read_averaged	92
6.4.3.2	read_Datarecord	93
6.4.3.3	read_HeaderRecord	94
6.4.3.4	write_averaged	94
6.4.3.5	write_DataRecord	95
6.4.3.6	write_HeaderRecord	95
6.5	クラス cio_DFI_SPH	96
6.5.1	説明	98
6.5.2	列挙型	98
6.5.2.1	DataDims	98
6.5.2.2	RealType	98
6.5.3	コンストラクタとデストラクタ	98
6.5.3.1	cio_DFI_SPH	98
6.5.3.2	cio_DFI_SPH	99
6.5.3.3	~cio_DFI_SPH	100
6.5.4	関数	100
6.5.4.1	read_averaged	100
6.5.4.2	read_Datarecord	101
6.5.4.3	read_HeaderRecord	102
6.5.4.4	write_averaged	105
6.5.4.5	write_DataRecord	106
6.5.4.6	write_HeaderRecord	106
6.6	クラス cio_Domain	108
6.6.1	説明	109
6.6.2	コンストラクタとデストラクタ	109
6.6.2.1	cio_Domain	109
6.6.2.2	cio_Domain	109
6.6.2.3	~cio_Domain	110
6.6.3	関数	110
6.6.3.1	Read	110
6.6.3.2	Write	111
6.6.4	変数	112
6.6.4.1	ActiveSubdomainFile	112
6.6.4.2	GlobalDivision	112
6.6.4.3	GlobalOrigin	112
6.6.4.4	GlobalRegion	113
6.6.4.5	GlobalVoxel	113
6.7	クラス cio_FileInfo	113



6.7.1	説明	114
6.7.2	コンストラクタとデストラクタ	114
6.7.2.1	cio_FileInfo	114
6.7.2.2	cio_FileInfo	114
6.7.2.3	~cio_FileInfo	115
6.7.3	関数	115
6.7.3.1	getComponentVariable	115
6.7.3.2	Read	116
6.7.3.3	setComponentVariable	119
6.7.3.4	Write	119
6.7.4	変数	120
6.7.4.1	ArrayShape	120
6.7.4.2	Component	121
6.7.4.3	ComponentVariable	121
6.7.4.4	DataType	121
6.7.4.5	DirectoryPath	121
6.7.4.6	Endian	121
6.7.4.7	FileFormat	121
6.7.4.8	GuideCell	122
6.7.4.9	Prefix	122
6.7.4.10	TimeSliceDirFlag	122
6.8	クラス cio_FilePath	122
6.8.1	説明	122
6.8.2	コンストラクタとデストラクタ	123
6.8.2.1	cio_FilePath	123
6.8.2.2	cio_FilePath	123
6.8.2.3	~cio_FilePath	123
6.8.3	関数	123
6.8.3.1	Read	123
6.8.3.2	Write	124
6.8.4	変数	124
6.8.4.1	ProcDFIFile	124
6.9	クラス cio_Interval_Mngr	125
6.9.1	説明	126
6.9.2	列挙型	126
6.9.2.1	type_IO_spec	126
6.9.3	コンストラクタとデストラクタ	127
6.9.3.1	cio_Interval_Mngr	127
6.9.3.2	~cio_Interval_Mngr	127
6.9.4	関数	127

6.9.4.1	<a href="#">calcNextStep</a>	127
6.9.4.2	<a href="#">calcNextTime</a>	128
6.9.4.3	<a href="#">dmod</a>	128
6.9.4.4	<a href="#">getIntervalStep</a>	128
6.9.4.5	<a href="#">getIntervalTime</a>	128
6.9.4.6	<a href="#">getMode</a>	128
6.9.4.7	<a href="#">getStartStep</a>	129
6.9.4.8	<a href="#">getStartTime</a>	129
6.9.4.9	<a href="#">initTrigger</a>	129
6.9.4.10	<a href="#">isLastStep</a>	130
6.9.4.11	<a href="#">isLastTime</a>	130
6.9.4.12	<a href="#">isStarted</a>	130
6.9.4.13	<a href="#">isTriggered</a>	130
6.9.4.14	<a href="#">normalizeBaseTime</a>	132
6.9.4.15	<a href="#">normalizeDelteT</a>	132
6.9.4.16	<a href="#">normalizeIntervalTime</a>	132
6.9.4.17	<a href="#">normalizeLastTime</a>	132
6.9.4.18	<a href="#">normalizeStartTime</a>	132
6.9.4.19	<a href="#">normalizeTime</a>	133
6.9.4.20	<a href="#">setInterval</a>	133
6.9.4.21	<a href="#">setLast</a>	133
6.9.4.22	<a href="#">setMode</a>	134
6.9.4.23	<a href="#">setStart</a>	134
6.9.5	<a href="#">変数</a>	134
6.9.5.1	<a href="#">m_base_step</a>	134
6.9.5.2	<a href="#">m_base_time</a>	134
6.9.5.3	<a href="#">m_dt</a>	134
6.9.5.4	<a href="#">m_intvl_step</a>	135
6.9.5.5	<a href="#">m_intvl_time</a>	135
6.9.5.6	<a href="#">m_last_step</a>	135
6.9.5.7	<a href="#">m_last_time</a>	135
6.9.5.8	<a href="#">m_mode</a>	135
6.9.5.9	<a href="#">m_start_step</a>	135
6.9.5.10	<a href="#">m_start_time</a>	135
6.10	<a href="#">クラス cio_MPI</a>	136
6.10.1	<a href="#">説明</a>	136
6.10.2	<a href="#">コンストラクタとデストラクタ</a>	136
6.10.2.1	<a href="#">cio_MPI</a>	136
6.10.2.2	<a href="#">cio_MPI</a>	136
6.10.2.3	<a href="#">~cio_MPI</a>	137

6.10.3	関数	137
6.10.3.1	Read	137
6.10.3.2	Write	138
6.10.4	変数	138
6.10.4.1	NumberOfGroup	138
6.10.4.2	NumberOfRank	138
6.11	クラス cio_Process	138
6.11.1	説明	140
6.11.2	型定義	140
6.11.2.1	headT	140
6.11.3	コンストラクタとデストラクタ	140
6.11.3.1	cio_Process	140
6.11.3.2	~cio_Process	140
6.11.4	関数	140
6.11.4.1	CheckReadRank	140
6.11.4.2	CheckStartEnd	141
6.11.4.3	CreateHeadMap	142
6.11.4.4	CreateHeadMap	143
6.11.4.5	CreateRankList	143
6.11.4.6	CreateRankList	144
6.11.4.7	CreateRankMap	145
6.11.4.8	CreateRankMap	146
6.11.4.9	CreateSubDomainInfo	147
6.11.4.10	isMatchEndianSbdmMagick	147
6.11.4.11	Read	148
6.11.4.12	ReadActiveSubdomainFile	149
6.11.4.13	Write	150
6.11.5	変数	150
6.11.5.1	m_rankMap	150
6.11.5.2	RankList	151
6.12	クラス cio_Rank	151
6.12.1	説明	151
6.12.2	コンストラクタとデストラクタ	151
6.12.2.1	cio_Rank	151
6.12.2.2	~cio_Rank	152
6.12.3	関数	152
6.12.3.1	Read	152
6.12.3.2	Write	153
6.12.4	変数	154
6.12.4.1	HeadIndex	154

6.12.4.2	HostName	154
6.12.4.3	RankID	154
6.12.4.4	TailIndex	154
6.12.4.5	VoxelSize	154
6.13	クラス cio_Slice	154
6.13.1	説明	155
6.13.2	コンストラクタとデストラクタ	155
6.13.2.1	cio_Slice	155
6.13.2.2	~cio_Slice	156
6.13.3	関数	156
6.13.3.1	Read	156
6.13.3.2	Write	157
6.13.4	変数	158
6.13.4.1	AveragedStep	158
6.13.4.2	AveragedTime	158
6.13.4.3	avr_mode	159
6.13.4.4	Max	159
6.13.4.5	Min	159
6.13.4.6	step	159
6.13.4.7	time	159
6.13.4.8	VectorMax	159
6.13.4.9	VectorMin	159
6.14	クラス cio_TextParser	160
6.14.1	説明	160
6.14.2	コンストラクタとデストラクタ	160
6.14.2.1	cio_TextParser	160
6.14.2.2	~cio_TextParser	161
6.14.3	関数	161
6.14.3.1	chkLabel	161
6.14.3.2	chkNode	161
6.14.3.3	countLabels	162
6.14.3.4	GetNodeStr	163
6.14.3.5	getTPinstance	164
6.14.3.6	GetValue	164
6.14.3.7	GetValue	165
6.14.3.8	GetValue	166
6.14.3.9	GetVector	167
6.14.3.10	GetVector	168
6.14.3.11	GetVector	169
6.14.3.12	readTPfile	169

6.14.3.13	<code>remove</code>	170
6.14.4	変数	170
6.14.4.1	<code>tp</code>	170
6.15	クラス <code>cio_TimeSlice</code>	170
6.15.1	説明	171
6.15.2	コンストラクタとデストラクタ	171
6.15.2.1	<code>cio_TimeSlice</code>	171
6.15.2.2	<code>~cio_TimeSlice</code>	171
6.15.3	関数	171
6.15.3.1	<code>AddSlice</code>	171
6.15.3.2	<code>getMinMax</code>	172
6.15.3.3	<code>getVectorMinMax</code>	173
6.15.3.4	<code>Read</code>	174
6.15.3.5	<code>Write</code>	175
6.15.4	変数	175
6.15.4.1	<code>SliceList</code>	175
6.16	クラス テンプレート <code>cio_TypeArray&lt; T &gt;</code>	176
6.16.1	説明	177
6.16.2	コンストラクタとデストラクタ	177
6.16.2.1	<code>cio_TypeArray</code>	177
6.16.2.2	<code>cio_TypeArray</code>	178
6.16.2.3	<code>~cio_TypeArray</code>	178
6.16.2.4	<code>cio_TypeArray</code>	178
6.16.3	関数	178
6.16.3.1	<code>_val</code>	178
6.16.3.2	<code>_val</code>	179
6.16.3.3	<code>copyArray</code>	179
6.16.3.4	<code>copyArray</code>	179
6.16.3.5	<code>getData</code>	180
6.16.3.6	<code>hval</code>	181
6.16.3.7	<code>hval</code>	181
6.16.3.8	<code>readBinary</code>	181
6.16.3.9	<code>val</code>	181
6.16.3.10	<code>val</code>	182
6.16.3.11	<code>writeBinary</code>	182
6.16.4	変数	182
6.16.4.1	<code>m_data</code>	182
6.16.4.2	<code>m_outptr</code>	182
6.17	クラス <code>cio_Unit</code>	182
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	183
6.17.3	関数	183
6.17.3.1	GetBaseName	183
6.17.3.2	GetBaseValue	184
6.17.3.3	GetDiffName	184
6.17.3.4	GetDiffValue	185
6.17.3.5	GetUnit	185
6.17.3.6	GetUnitElem	186
6.17.3.7	Read	186
6.17.3.8	Write	187
6.17.4	変数	188
6.17.4.1	UnitList	188
6.18	クラス cio_UnitElem	188
6.18.1	説明	189
6.18.2	コンストラクタとデストラクタ	189
6.18.2.1	cio_UnitElem	189
6.18.2.2	cio_UnitElem	189
6.18.2.3	~cio_UnitElem	189
6.18.3	関数	189
6.18.3.1	Read	189
6.18.3.2	Write	190
6.18.4	変数	191
6.18.4.1	BaseName	191
6.18.4.2	BaseValue	191
6.18.4.3	DiffName	191
6.18.4.4	DiffValue	191
6.18.4.5	Name	191
6.18.4.6	Unit	191
<b>7</b>	<b>ファイル</b>	<b>193</b>
7.1	cio_ActiveSubDomain.C	193
7.1.1	説明	193
7.2	cio_ActiveSubDomain.h	193
7.3	cio_Array.h	194
7.3.1	関数	195
7.3.1.1	cio_interp_ijkn_r4_	195
7.3.1.2	cio_interp_ijkn_r8_	195
7.3.1.3	cio_interp_nijk_r4_	195

7.3.1.4	<code>cio_interp_nijk_r8_</code>	195
7.4	<code>cio_Array_inline.h</code>	195
7.4.1	マクロ定義	196
7.4.1.1	<code>CIO_INLINE</code>	196
7.4.1.2	<code>CIO_MEMFUN</code>	196
7.5	<code>cio_Define.h</code>	196
7.5.1	説明	199
7.5.2	マクロ定義	199
7.5.2.1	<code>_CIO_IDX_IJ</code>	199
7.5.2.2	<code>_CIO_IDX_IJK</code>	199
7.5.2.3	<code>_CIO_IDX_IJKN</code>	201
7.5.2.4	<code>_CIO_IDX_NIJ</code>	201
7.5.2.5	<code>_CIO_IDX_NIJK</code>	202
7.5.2.6	<code>_CIO_TAB_STR</code>	202
7.5.2.7	<code>_CIO_WRITE_TAB</code>	202
7.5.2.8	<code>D_CIO_BIG</code>	203
7.5.2.9	<code>D_CIO_EXT_BOV</code>	203
7.5.2.10	<code>D_CIO_EXT_SPH</code>	203
7.5.2.11	<code>D_CIO_FLOAT32</code>	203
7.5.2.12	<code>D_CIO_FLOAT64</code>	203
7.5.2.13	<code>D_CIO_IJNK</code>	203
7.5.2.14	<code>D_CIO_INT16</code>	203
7.5.2.15	<code>D_CIO_INT32</code>	203
7.5.2.16	<code>D_CIO_INT64</code>	204
7.5.2.17	<code>D_CIO_INT8</code>	204
7.5.2.18	<code>D_CIO_LITTLE</code>	204
7.5.2.19	<code>D_CIO_NIJK</code>	204
7.5.2.20	<code>D_CIO_OFF</code>	204
7.5.2.21	<code>D_CIO_ON</code>	204
7.5.2.22	<code>D_CIO_UINT16</code>	204
7.5.2.23	<code>D_CIO_UINT32</code>	204
7.5.2.24	<code>D_CIO_UINT64</code>	204
7.5.2.25	<code>D_CIO_UINT8</code>	204
7.6	<code>cio_DFI.C</code>	205
7.6.1	説明	205
7.7	<code>cio_DFI.h</code>	205
7.7.1	説明	206
7.8	<code>cio_DFI_BOV.C</code>	206
7.8.1	説明	206
7.9	<code>cio_DFI_BOV.h</code>	207

7.9.1 説明	207
7.10 cio_DFI_inline.h	207
7.10.1 マクロ定義	208
7.10.1.1 CIO_INLINE	208
7.11 cio_DFI_Read.C	208
7.11.1 説明	208
7.12 cio_DFI_SPH.C	208
7.12.1 説明	208
7.13 cio_DFI_SPH.h	209
7.13.1 説明	209
7.14 cio_DFI_Write.C	209
7.14.1 説明	210
7.15 cio_Domain.C	210
7.15.1 説明	210
7.16 cio_Domain.h	210
7.16.1 説明	211
7.17 cio_endianUtil.h	211
7.17.1 説明	212
7.17.2 マクロ定義	212
7.17.2.1 BSWAP16	212
7.17.2.2 BSWAP32	212
7.17.2.3 BSWAP64	212
7.17.2.4 BSWAP_X_16	213
7.17.2.5 BSWAP_X_32	213
7.17.2.6 BSWAP_X_64	213
7.17.2.7 BSWAPVEC	213
7.17.2.8 CIO_INLINE	213
7.17.2.9 DBSWAPVEC	213
7.17.2.10 SBSWAPVEC	214
7.18 cio_FileInfo.C	214
7.18.1 説明	214
7.19 cio_FileInfo.h	214
7.19.1 説明	215
7.20 cio_FilePath.C	215
7.20.1 説明	215
7.21 cio_FilePath.h	215
7.21.1 説明	216
7.22 cio_interp_ijkn.h	216
7.23 cio_interp_nijk.h	216
7.24 cio_Interval_Mngr.h	216



7.25	<code>cio_MPI.C</code>	216
7.25.1	説明	216
7.26	<code>cio_MPI.h</code>	217
7.26.1	説明	217
7.27	<code>cio_PathUtil.h</code>	217
7.27.1	マクロ定義	218
7.27.1.1	<code>MAXPATHLEN</code>	218
7.28	<code>cio_Process.C</code>	218
7.28.1	説明	218
7.29	<code>cio_Process.h</code>	219
7.29.1	説明	219
7.30	<code>cio_TextParser.C</code>	219
7.30.1	説明	219
7.31	<code>cio_TextParser.h</code>	220
7.31.1	説明	220
7.32	<code>cio_TimeSlice.C</code>	220
7.32.1	説明	221
7.33	<code>cio_TimeSlice.h</code>	221
7.33.1	説明	221
7.34	<code>cio_TypeArray.h</code>	221
7.35	<code>cio_Unit.C</code>	222
7.35.1	説明	223
7.36	<code>cio_Unit.h</code>	223
7.36.1	説明	223
7.37	<code>cio_Version.h</code>	223
7.37.1	説明	224
7.37.2	マクロ定義	224
7.37.2.1	<code>CIO_REVISION</code>	224
7.37.2.2	<code>CIO_VERSION_NO</code>	224
7.38	<code>mpi_stubs.h</code>	224
7.38.1	マクロ定義	224
7.38.1.1	<code>MPI_CHAR</code>	224
7.38.1.2	<code>MPI_COMM_WORLD</code>	225
7.38.1.3	<code>MPI_INT</code>	225
7.38.1.4	<code>MPI_SUCCESS</code>	225
7.38.2	型定義	225
7.38.2.1	<code>MPI_Comm</code>	225
7.38.2.2	<code>MPI_Datatype</code>	225
7.38.3	関数	225
7.38.3.1	<code>MPI_Allgather</code>	225

7.38.3.2	MPI_Comm_rank	225
7.38.3.3	MPI_Comm_size	225
7.38.3.4	MPI_Gather	226
7.38.3.5	MPI_Init	226

索引	227
----	-----

## 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 . . . . .	89
cio_DFI_SPH . . . . .	96
cio_Domain . . . . .	108
cio_FileInfo . . . . .	113
cio_FilePath . . . . .	122
cio_Interval_Mngr . . . . .	125
cio_MPI . . . . .	136
cio_Process . . . . .	138
cio_Rank . . . . .	151
cio_Slice . . . . .	154
cio_TextParser . . . . .	160
cio_TimeSlice . . . . .	170
cio_Unit . . . . .	182
cio_UnitElem . . . . .	188



## 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。

<a href="#">cio_ActiveSubDomain</a>	21
<a href="#">cio_Array</a>	25
<a href="#">cio_DFI</a>	41
<a href="#">cio_DFI_BOV</a>	89
<a href="#">cio_DFI_SPH</a>	96
<a href="#">cio_Domain</a>	108
<a href="#">cio_FileInfo</a>	113
<a href="#">cio_FilePath</a>	122
<a href="#">cio_Interval_Mngr</a>	125
<a href="#">cio_MPI</a>	136
<a href="#">cio_Process</a>	138
<a href="#">cio_Rank</a>	151
<a href="#">cio_Slice</a>	154
<a href="#">cio_TextParser</a>	160
<a href="#">cio_TimeSlice</a>	170
<a href="#">cio_TypeArray&lt; T &gt;</a>	176
<a href="#">cio_Unit</a>	182
<a href="#">cio_UnitElem</a>	188





## 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。

<a href="#">cio_ActiveSubDomain.C</a>	
Cio_ActiveSubDomain class 関数	193
<a href="#">cio_ActiveSubDomain.h</a>	193
<a href="#">cio_Array.h</a>	194
<a href="#">cio_Array_inline.h</a>	195
<a href="#">cio_Define.h</a>	
CIO の定義マクロ記述ヘッダーファイル	196
<a href="#">cio_DFI.C</a>	
Cio_DFI Class	205
<a href="#">cio_DFI.h</a>	
Cio_DFI Class Header	205
<a href="#">cio_DFI_BOV.C</a>	
Cio_DFI_BOV Class	206
<a href="#">cio_DFI_BOV.h</a>	
Cio_DFI_BOV Class Header	207
<a href="#">cio_DFI_inline.h</a>	207
<a href="#">cio_DFI_Read.C</a>	
Cio_DFI Class	208
<a href="#">cio_DFI_SPH.C</a>	
Cio_DFI_SPH Class	208
<a href="#">cio_DFI_SPH.h</a>	
Cio_DFI_SPH Class Header	209
<a href="#">cio_DFI_Write.C</a>	
Cio_DFI Class	209
<a href="#">cio_Domain.C</a>	
Cio_Domain Class	210
<a href="#">cio_Domain.h</a>	
Cio_Domain Class Header	210
<a href="#">cio_endianUtil.h</a>	
エンディアンユーティリティマクロ・関数ファイル	211
<a href="#">cio_FileInfo.C</a>	
Cio_FileInfo Class	214
<a href="#">cio_FileInfo.h</a>	
Cio_FileInfo Class Header	214
<a href="#">cio_FilePath.C</a>	
Cio_FilePath Class	215
<a href="#">cio_FilePath.h</a>	
Cio_FilePath Class Header	215

<a href="#">cio_interp_ijkn.h</a>	216
<a href="#">cio_interp_nijk.h</a>	216
<a href="#">cio_Interval_Mngr.h</a>	216
<a href="#">cio_MPI.C</a>	
Cio_MPI Class	216
<a href="#">cio_MPI.h</a>	
Cio_MPI Class Header	217
<a href="#">cio_PathUtil.h</a>	217
<a href="#">cio_Process.C</a>	
Cio_Rank & <a href="#">cio_Process</a> Class	218
<a href="#">cio_Process.h</a>	
Cio_RANK & <a href="#">cio_Process</a> Class Header	219
<a href="#">cio_TextParser.C</a>	
TextParser Control class	219
<a href="#">cio_TextParser.h</a>	
TextParser Control class Header	220
<a href="#">cio_TimeSlice.C</a>	
Cio_Slice Class	220
<a href="#">cio_TimeSlice.h</a>	
Cio_Slice & <a href="#">cio_TimeSliceClass</a> Header	221
<a href="#">cio_TypeArray.h</a>	221
<a href="#">cio_Unit.C</a>	
Cio_Unit Class	222
<a href="#">cio_Unit.h</a>	
Cio_UnitElem & <a href="#">cio_Unit</a> Class Header	223
<a href="#">cio_Version.h</a>	223
<a href="#">mpi_stubs.h</a>	224

## 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_GUIDECCELL` = 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_OPENERERROR` = 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_REC7 = 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_HEADER = 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_REC5 = 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 `vfvpPath_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 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_GLOBALREGION** DFI GlobalRegion 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_GLOBALVOXEL** DFI GlobalVoxel 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_GLOBALDIVISION** DFI GlobalDivision 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_DIRECTORYPATH** DFI DirectoryPath 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_TIMESLICEDIRECTORY** DFI TimeSliceDirectoryPath 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_PREFIX** DFI Prefix 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_FILEFORMAT** DFI FileFormat 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_GUIDECCELL** DFI GuideCell 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_DATATYPE** DFI DataType 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_ENDIAN** DFI Endian 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_ARRAYSHAPE** DFI ArrayShape 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_COMPONENT** DFI Component 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_FILEPATH\_PROCESS** DFI FilePath/Process 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_NO\_RANK** DFI Rank 要素なし  
**E\_CIO\_ERROR\_READ\_DFI\_ID** DFI ID 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_HOSTNAME** DFI HoatName 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_VOXELSIZE** DFI VoxelSize 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_HEADINDEX** DFI HeadIndex 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_TAILINDEX** DFI TailIndex 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_NO\_SLICE** DFI TimeSlice 要素なし  
**E\_CIO\_ERROR\_READ\_DFI\_STEP** DFI Step 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_TIME** DFI Time 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_NO\_MINMAX** DFI MinMax 要素なし  
**E\_CIO\_ERROR\_READ\_DFI\_MIN** DFI Min 読み込みエラー  
**E\_CIO\_ERROR\_READ\_DFI\_MAX** DFI Max 読み込みエラー  
**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 読み込みエラー  
**E\_CIO\_ERROR\_READ\_FILEPATH** FilePath 読み込みエラー  
**E\_CIO\_ERROR\_READ\_UNIT** UNIT 読み込みエラー  
**E\_CIO\_ERROR\_READ\_TIMESLICE** TimeSlice 読み込みエラー  
**E\_CIO\_ERROR\_READ\_PROCFILE\_OPENERROR** Proc ファイルオープンエラー  
**E\_CIO\_ERROR\_READ\_DOMAIN** Domain 読み込みエラー  
**E\_CIO\_ERROR\_READ\_MPI** MPI 読み込みエラー  
**E\_CIO\_ERROR\_READ\_PROCESS** Process 読み込みエラー  
**E\_CIO\_ERROR\_READ\_FIELDDATA\_FILE** フィールドデータファイル読み込みエラー  
**E\_CIO\_ERROR\_READ\_SPH\_FILE** SPH ファイル読み込みエラー

**E\_CIO\_ERROR\_READ\_SPH\_REC1** SPH ファイルレコード 1 読み込みエラー  
**E\_CIO\_ERROR\_READ\_SPH\_REC2** SPH ファイルレコード 2 読み込みエラー  
**E\_CIO\_ERROR\_READ\_SPH\_REC3** SPH ファイルレコード 3 読み込みエラー  
**E\_CIO\_ERROR\_READ\_SPH\_REC4** SPH ファイルレコード 4 読み込みエラー  
**E\_CIO\_ERROR\_READ\_SPH\_REC5** SPH ファイルレコード 5 読み込みエラー  
**E\_CIO\_ERROR\_READ\_SPH\_REC6** SPH ファイルレコード 6 読み込みエラー  
**E\_CIO\_ERROR\_READ\_SPH\_REC7** SPH ファイルレコード 7 読み込みエラー  
**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Format が合致しない ( Endian 形式がBig,Little 以外 )  
**E\_CIO\_ERROR\_READ\_BOV\_FILE** BOV ファイル読み込みエラー  
**E\_CIO\_ERROR\_READ\_FIELD\_HEADER\_RECORD** フィールドヘッダーレコード読み込み失敗  
**E\_CIO\_ERROR\_READ\_FIELD\_DATA\_RECORD** フィールドデータレコード読み込み失敗  
**E\_CIO\_ERROR\_READ\_FIELD\_AVERAGED\_RECORD** フィールドAverage 読み込み失敗  
**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\_OPENERERROR** 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\_OPENERERROR** 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_GUIDECCELL = 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_OPENERERROR = 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_OPENERERROR = 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 ///< Data type 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_OPENERERROR = 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_OPENEROR = 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

読み込みタイプコード

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 ( strcmp( 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

<i>in</i>	<i>pos</i>	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

<i>in</i>	<i>dom</i>	比較対象の活性サブドメイン情報
-----------	------------	-----------------

戻り値

<i>true</i>	違う位置情報を持つ
<i>false</i>	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

<i>in</i>	<i>dom</i>	比較対象の活性サブドメイン情報
-----------	------------	-----------------

戻り値

<i>true</i>	同じ位置情報を持つ
<i>false</i>	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

<i>in</i>	<i>pos</i>	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().

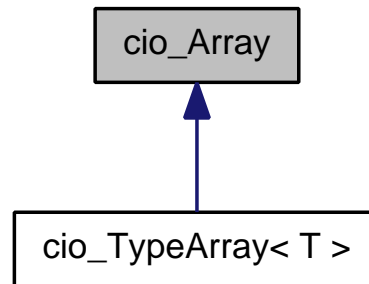
このクラスの説明は次のファイルから生成されました:

- [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 getData Type () const  
データタイプの取得
- char \* getData TypeString () 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\_gcl, m\_gcl, m\_ncomp, m\_ncompl, m\_shape, m\_sz, m\_Sz, m\_szl, m\_Szl, 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\_gcl.

参照元 cio\_TypeArray< T >::copyArray().

```

208 {
209     return m_gcl;
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::getNcomplnt ( ) 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_ncompl`.

```
220 {
221     return m_ncompl;
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 467 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().

```

468 {
469     err = 1;
470
471     // データタイプ
472     // 実数型のみ対応
473     CIO::E_CIO_DTYPE dtype = src->getDataType();
474     if( dtype != CIO::E_CIO_FLOAT32 && dtype != CIO::E_CIO_FLOAT64 )
475     {
476         err = -1;
477         return NULL;
478     }
479
480     // 配列形状
481     CIO::E_CIO_ARRAYSHAPE shape = src->getArrayShape();
482
483     // 成分数
484     // 成分数は 1 か 3 のみ対応
485     int ncomp = src->getNcomp();
486     if( ncomp != 1 && ncomp != 3 )
487     {
488         err = -1;
489         return NULL;
490     }
491
492     // その他の情報の取得
493     int gcS = src->getGc();
494     void *ptrS = src->getData();
495     const int *szS = src->getArraySizeInt();
496     const int *headS = src->getHeadIndex();
497     const int *tailS = src->getTailIndex();
498
499     // 密配列のインスタンス
500     int gcD = gcS*2;
501     int szD[3] = {szS[0]*2, szS[1]*2, szS[2]*2};
502     cio_Array *dst = cio_Array::instanceArray( dtype, shape, szD, gcD, ncomp );
503     void *ptrD = dst->getData();
504
505     // head インデックスのセット
506     int headD[3];
507     for( int i=0; i<3; i++ )
508     {
509         headD[i] = headS[i]*2;
510         if( !head0start )
511         {
512             headD[i] -= 1;
513         }
514     }
515     dst->setHeadIndex( headD );
516
517     // f90 コードのコール (配列形状、実数型毎)
518     if( shape == CIO::E_CIO_IJKN )
519     {
520         if( dtype == CIO::E_CIO_FLOAT32 )
521         {
522             cio_interp_ijkn_r4_(szS, &gcS, szD, &gcD, &ncomp, (float*)ptrS, (float*)ptrD);
523         }
524         else
525         {
526             cio_interp_ijkn_r8_(szS, &gcS, szD, &gcD, &ncomp, (double*)ptrS, (double*)ptrD);
527         }
528     }
529     else
530     {
531         if( dtype == CIO::E_CIO_FLOAT32 )
532         {
533             cio_interp_nijk_r4_(szS, &gcS, szD, &gcD, &ncomp, (float*)ptrS, (float*)ptrD);
534         }
535         else
536         {
537             cio_interp_nijk_r8_(szS, &gcS, szD, &gcD, &ncomp, (double*)ptrS, (double*)ptrD);
538         }
539     }
540

```



```

541     return dst;
542 }

```

**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 `getGclnt()`.

#### 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 `getNcomplnt()`.

#### 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(), getSize(), 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 \_getSize(), cio\_Array(), getArrayLength(), と instanceArray().

#### 6.2.4.11 int cio\_Array::m\_sz[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 getSizeInt().

#### 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 \_getSizeInt(), 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().

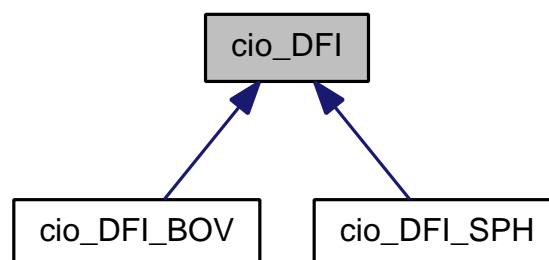
このクラスの説明は次のファイルから生成されました:

- [cio\\_Array.h](#)
- [cio\\_Array\\_inline.h](#)

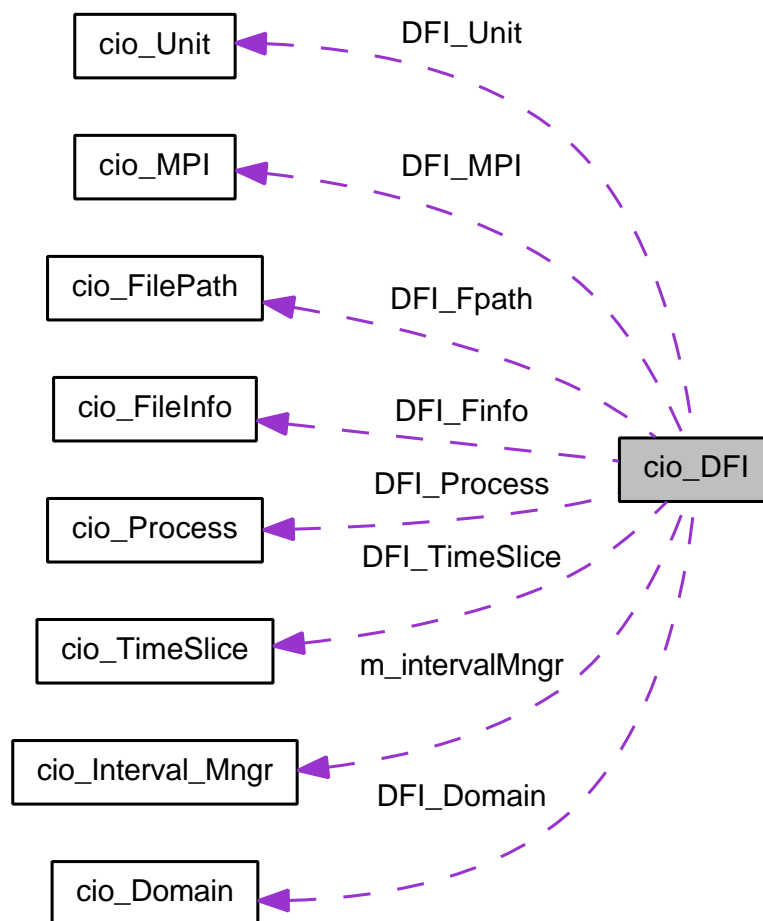
## 6.3 クラス cio\_DFI

```
#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
- `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 GetDataTimeString ()`  
*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 std::string BaseName, const double BaseValue, const std::string DiffName="", const double DiffValue=0.0)  
 Uuit をセットする
  - `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)

- 読み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 `normalizeDeltaT` (const double scale)  
インターバルの *DeltaT* をスケールで無次元化する
- 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 std::string `getVersionInfo` ()

## Protected メソッド

- 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み込み*
- 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み込み*
- 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み込み*
- 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])  
*読み込み判定判定*
- 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み込み範囲を求める*
- `CIO::E_CIO_ERRORCODE WriteIndexDfiFile` (const std::string dfi\_name)  
*index DFI ファイル出力*
- 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*
- 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*

## 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*
- static int `MakeDirectorySub` (std::string path)  
*ディレクトリパスの作成 (system 関数 mkdir で作成)*

## Protected 変数

- [MPI\\_Comm m\\_comm](#)  
MPI コミュニケーター
- [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](#)  
読み込みタイプ
- [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](#)  
読み込みランクリスト
- [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 std::string *BaseName*, const double *BaseValue*, const std::string *DiffName* = " ", const double *DiffValue* = 0.0 )

Unit をセットする

引数

in	<i>Name</i>	追加する単位系 ("Length","Velocity",...)
in	<i>Unit</i>	単位ラベル ("M","CM","MM","M/S",...)
in	<i>BaseName</i>	代表値名 ("L0","V0",...)
in	<i>BaseValue</i>	代表値値
in	<i>DiffName</i>	差があるときの名前 ("DiffPrs","DiffTemp",...)
in	<i>DiffValue</i>	差の値

UnitElem の生成

UnitList へのセット

cio\_DFI.C の 847 行で定義されています。

参照先 DFI\_Unit, と cio\_Unit::UnitList.

```
853 {
854
855   cio_UnitElem unit =   cio_UnitElem(Name,Unit,BaseName,BaseValue,DiffName,DiffValue);
856
857   DFI_Unit.UnitList.insert (map<std::string,cio_UnitElem>::value_type (Name,unit));
858
859
860
861 }
```

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* )

読み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、読み込みランクリスト readRankList を生成する

引数

in	<i>dfi_domain</i>	DFI の domain 情報
in	<i>head</i>	ソルバーのHeadIndex
in	<i>tail</i>	ソルバーのTailIndex
in	<i>readflag</i>	読み込み方法
out	<i>readRankList</i>	読み込みランクリスト

戻り値

error code

cio\_DFI.C の 914 行で定義されています。

参照先 cio\_Process::CheckReadRank(), と DFI\_Process.

```

919 {
920
921     return DFI_Process.CheckReadRank(dfi_domain, head, tail, readflag, readRankList);
922
923 }

```

**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]

読み込み判定判定

引数

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

戻り値

読み込みタイプコード

cio\_DFI.C の 584 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().

```

588 {
589
590     bool isSameDiv=true;
591
592     //分割数チェック
593     for(int i=0; i<3; i++ ) {
594         if( DFI_GlobalDivision[i] != G_Div[i] ) {
595             isSameDiv = false;
596         }
597     }
598
599     if( isSameDiv ) {
600         if( G_voxel[0] == DFI_GlobalVoxel[0]  &&
601             G_voxel[1] == DFI_GlobalVoxel[1]  &&
602             G_voxel[2] == DFI_GlobalVoxel[2] ) return CIO::E_CIO_SAMEDIV_SAMERES;
603
604         if( G_voxel[0] == DFI_GlobalVoxel[0]*2 &&
605             G_voxel[1] == DFI_GlobalVoxel[1]*2 &&
606             G_voxel[2] == DFI_GlobalVoxel[2]*2 ) return
CIO::E_CIO_SAMEDIV_REFINEMENT;
607     } else {
608         if( G_voxel[0] == DFI_GlobalVoxel[0]  &&
609             G_voxel[1] == DFI_GlobalVoxel[1]  &&
610             G_voxel[2] == DFI_GlobalVoxel[2] ) return CIO::E_CIO_DIFFDIV_SAMERES;
611
612         if( G_voxel[0] == DFI_GlobalVoxel[0]*2 &&
613             G_voxel[1] == DFI_GlobalVoxel[1]*2 &&
614             G_voxel[2] == DFI_GlobalVoxel[2]*2 ) return
CIO::E_CIO_DIFFDIV_REFINEMENT;
615     }
616
617     return CIO::E_CIO_READTYPE_UNKNOWN;
618 }

```

**6.3.3.4 void cio\_DFI::cio\_Create\_dfiProcessInfo ( const MPI\_Comm *comm*, cio\_Process & *G\_Process* )**  
[protected]

Create Process.

## 引数

in	<i>comm</i>	MPI コミュニケータ
out	<i>G_Process</i>	Process class

cio\_DFI.C の 531 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().

```

533 {
534
535     cio_Rank G_Rank;
536
537     int RankID;
538     MPI_Comm_rank( comm, &RankID );
539
540     int nrank;
541     MPI_Comm_size( comm, &nrank );
542
543     if( nrank > 1 ) {
544         int *headtail = NULL;
545         if( RankID == 0 ) {
546             headtail = new int[6*nrank];
547         }
548
549         int sbuff[6];
550         for(int i=0; i<3; i++) {
551             sbuff[i] = DFI_Process.RankList[RankID].HeadIndex[i];
552             sbuff[i+3] = DFI_Process.RankList[RankID].TailIndex[i];
553         }
554
555         MPI_Gather(sbuff, 6, MPI_INT, headtail, 6, MPI_INT, 0, comm);
556
557         if( RankID == 0 ) {
558             for(int i=0; i<nrank; i++) {
559                 G_Rank.RankID=i;
560                 for(int j=0; j<3; j++) {
561                     G_Rank.HeadIndex[j]=headtail[i*6+j];
562                     G_Rank.TailIndex[j]=headtail[i*6+j+3];
563                     G_Rank.VoxelSize[j]=G_Rank.TailIndex[j]-G_Rank.HeadIndex[j]+1;
564                 }
565                 G_Process.RankList.push_back(G_Rank);
566             }
567         }
568
569         if ( RankID == 0 ) delete [] headtail;
570
571     } else {
572         G_Rank.RankID=0;
573         for(int i=0; i<3; i++) {
574             G_Rank.HeadIndex[i]=DFI_Process.RankList[0].HeadIndex[i];
575             G_Rank.TailIndex[i]=DFI_Process.RankList[0].TailIndex[i];
576             G_Rank.VoxelSize[i]=G_Rank.TailIndex[i]-G_Rank.HeadIndex[i]+1;
577         }
578         G_Process.RankList.push_back(G_Rank);
579     }
580 }

```

### 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	<i>Dtype</i>	データタイプ
----	--------------	--------

## 戻り値

データタイプ (string)

cio\_DFI.C の 482 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 GetDataTimeString(), と cio\_FileInfo::Write().

```

483 {
484     if ( Dtype == CIO::E_CIO_INT8 ) return D_CIO_INT8;
485     else if( Dtype == CIO::E_CIO_INT16 ) return D_CIO_INT16;
486     else if( Dtype == CIO::E_CIO_INT32 ) return D_CIO_INT32;
487     else if( Dtype == CIO::E_CIO_INT64 ) return D_CIO_INT64;
488     else if( Dtype == CIO::E_CIO_UINT8 ) return D_CIO_UINT8;
489     else if( Dtype == CIO::E_CIO_UINT16 ) return D_CIO_UINT16;
490     else if( Dtype == CIO::E_CIO_UINT32 ) return D_CIO_UINT32;
491     else if( Dtype == CIO::E_CIO_UINT64 ) return D_CIO_UINT64;
492     else if( Dtype == CIO::E_CIO_FLOAT32 ) return D_CIO_FLOAT32;
493     else if( Dtype == CIO::E_CIO_FLOAT64 ) return D_CIO_FLOAT64;
494     else return "dummy";
495 }
496 }
```

### 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 463 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().

```

464 {
465     if ( !strcasecmp(datatype.c_str(), "Int8" ) ) return CIO::E_CIO_INT8;
466     else if( !strcasecmp(datatype.c_str(), "Int16" ) ) return CIO::E_CIO_INT16;
467     else if( !strcasecmp(datatype.c_str(), "Int32" ) ) return CIO::E_CIO_INT32;
468     else if( !strcasecmp(datatype.c_str(), "Int64" ) ) return CIO::E_CIO_INT64;
469     else if( !strcasecmp(datatype.c_str(), "UInt8" ) ) return CIO::E_CIO_UINT8;
470     else if( !strcasecmp(datatype.c_str(), "UInt16" ) ) return CIO::E_CIO_UINT16;
471     else if( !strcasecmp(datatype.c_str(), "UInt32" ) ) return CIO::E_CIO_UINT32;
472     else if( !strcasecmp(datatype.c_str(), "UInt64" ) ) return CIO::E_CIO_UINT64;
473     else if( !strcasecmp(datatype.c_str(), "Float32" ) ) return CIO::E_CIO_FLOAT32;
474     else if( !strcasecmp(datatype.c_str(), "Float64" ) ) return CIO::E_CIO_FLOAT64;
475     return CIO::E_CIO_DTYPE_UNKNOWN;
476 }
477 }
478 }
```

### 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み込み範囲を求める

## 引数

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

cio\_DFI.C の 622 行で定義されています。

参照先 DFI\_Domain, と cio\_Domain::GlobalVoxel.

参照元 ReadData().

```

634 {
635
636   int src_head[3],src_tail[3],src_gc;
637   if( !isSame ) {
638     // 粗密のとき密に変換、ガイドセルは倍にする
639     src_gc = DFI_gc*2;
640     for(int i=0; i<3; i++) {
641       src_head[i]=DFI_head[i]*2-1;
642       src_tail[i]=DFI_tail[i]*2;
643     }
644   } else {
645     // 粗密でない時各値をコピー
646     src_gc = DFI_gc;
647     for(int i=0; i<3; i++) {
648       src_head[i]=DFI_head[i];
649       src_tail[i]=DFI_tail[i];
650     }
651   }
652
653   //スタート、エンドをセット
654   for(int i=0; i<3; i++) {
655     copy_sta[i] = max(head[i],src_head[i]);
656     copy_end[i] = min(tail[i],src_tail[i]);
657
658     //仮想セルが読み込みの実セル内のときの処理 (スタート)
659     if( copy_sta[i] == 1 ) {
660       copy_sta[i] -= min(gc,src_gc);
661     } else if( head[i]>src_head[i] ) {
662       copy_sta[i] = max(head[i]-gc,src_head[i]);
663     }
664
665     //仮想セルが読み込みの実セル内のときの処理
666     if( ( isSame && copy_end[i] == DFI_Domain.GlobalVoxel[i] ) ||
667         ( !isSame && copy_end[i] == DFI_Domain.GlobalVoxel[i]*2 ) ) {
668       copy_end[i] += min(gc,src_gc);
669     } else if( tail[i]<src_tail[i] ) {
670       copy_end[i] = min(tail[i]+gc,src_tail[i]);
671     }
672
673     //read satrt/end のセット
674     if( !isSame ) {
675       if( copy_sta[i]>0 ) read_sta[i] = (copy_sta[i]+1)/2;
676       else read_sta[i] = copy_sta[i]/2;
677
678       if( copy_end[i]>0 ) read_end[i] = (copy_end[i]+1)/2;
679       else read_end[i] = copy_end[i]/2;
680     } else {
681       read_sta[i] = copy_sta[i];
682       read_end[i] = copy_end[i];
683     }
684   }
685 }
686 }
687 }
```

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	<i>prefix</i>	ファイル接頭文字
----	---------------	----------

戻り値

DFI ファイル名

cio\_DFI.C の 822 行で定義されています。

参照先 CIO::cioPath\_ConnectPath(), CIO::cioPath\_DirName(), と CIO::cioPath\_FileName().

```

823 {
824
825     // directory path
826     std::string dirName = CIO::cioPath_DirName(prefix);
827
828     // file extension
829     std::string dfilename = CIO::cioPath_FileName(prefix, ".dfi");
830
831     // filename
832     std::string fname = CIO::cioPath_ConnectPath( dirName, dfilename );
833
834     #if 0 // for debug
835     printf("prefix    =%s\n", prefix.c_str() );
836     printf("  dirName =%s\n", dirName.c_str() );
837     printf("  dfilename =%s\n", dfilename.c_str() );
838     printf("   fname  =%s\n", fname.c_str() );
839     printf("\n");
840     #endif
841
842     return fname;
843 }
```

### 6.3.3.9 std::string cio\_DFI::Generate\_Directory\_Path( ) [protected]

dfi のパスとDirectoryPath を連結する関数

戻り値

パス名

cio\_DFI.C の 793 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().

```

794 {
795
796     // dfi のパスと DirectoryPath を連結する関数
797     // ただし、絶対パスのときは dfi のパスは無視
798     // CIO::cioPath_isAbsolute が true のとき絶対パス
799     // DirectoryPath + TimeSliceDir
800     std::string path = m_directoryPath;
801     if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON )
802     {
803         //path = CIO::cioPath_ConnectPath(path, m_timeSliceDir);
804         path = CIO::cioPath_ConnectPath(path, "");
805     }
806
807     // absolute path
808     if( CIO::cioPath_isAbsolute(path) )
809     {
810         return path;
811     }
812
813     // relative path
814     std::string dfidir = CIO::cioPath_DirName(m_indexDfiName);
815     path = CIO::cioPath_ConnectPath(dfidir, path);
816     return path;
817
818 }
```

6.3.3.10 `std::string cio_DFI::Generate_FieldFileName ( int RankID, int step, const bool mio )` [protected]

フィールドデータ ( SPH,BOV ) ファイル名の作成



## 引数

in	<i>RankID</i>	ランク番号
in	<i>step</i>	読み込みステップ番号
in	<i>mio</i>	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 691 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().

```

694 {
695
696     if( DFI_Finfo.DirectoryPath.empty() ) return NULL;
697     if( DFI_Finfo.Prefix.empty() ) return NULL;
698
699     std::string fmt;
700     if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_SPH ) {
701         fmt=D_CIO_EXT_SPH;
702     } else if( DFI_Finfo.FileFormat == CIO::E_CIO_FMT_BOV ) {
703         fmt=D_CIO_EXT_BOV;
704     }
705
706     int len = DFI_Finfo.DirectoryPath.size() + DFI_Finfo.Prefix.size() + fmt.size() + 25;
707     // id(6) + step(10) + 1(\0) + "_"(2) + "."(1)+"id"(2)
708     if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON ) len += 11;
709
710     char* tmp = new char[len];
711     memset(tmp, 0, sizeof(char)*len);
712
713     if( mio ) {
714         if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON ) {
715             sprintf(tmp, "%s/%010d/%s_%010d_id%06d.%s", DFI_Finfo.DirectoryPath.c_str(), step,
716                 DFI_Finfo.Prefix.c_str(),
717                 step, RankID, fmt.c_str());
718         } else {
719             sprintf(tmp, "%s/%s_%010d_id%06d.%s", DFI_Finfo.DirectoryPath.c_str(),
720                 DFI_Finfo.Prefix.c_str(),
721                 step, RankID, fmt.c_str());
722         }
723     } else {
724         if( DFI_Finfo.TimeSliceDirFlag == CIO::E_CIO_ON ) {
725             sprintf(tmp, "%s/%010d/%s_%010d.%s", DFI_Finfo.DirectoryPath.c_str(), step,
726                 DFI_Finfo.Prefix.c_str(),
727                 step, fmt.c_str());
728         }
729     }
730
731     std::string fname(tmp);
732     if( tmp ) delete [] tmp;
733
734     return fname;
735 }

```

**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	<i>Dtype</i>	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 499 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().

```

500 {
501
502     if ( Dtype == CIO::E_CIO_INT8 ) return sizeof(char);
503     else if( Dtype == CIO::E_CIO_INT16 ) return sizeof(short);
504     else if( Dtype == CIO::E_CIO_INT32 ) return sizeof(int);
505     else if( Dtype == CIO::E_CIO_INT64 ) return sizeof(long long);
506     else if( Dtype == CIO::E_CIO_UINT8 ) return sizeof(unsigned char);
507     else if( Dtype == CIO::E_CIO_UINT16 ) return sizeof(unsigned short);
508     else if( Dtype == CIO::E_CIO_UINT32 ) return sizeof(unsigned int);
509     else if( Dtype == CIO::E_CIO_UINT64 ) return sizeof(unsigned long long);
510     else if( Dtype == CIO::E_CIO_FLOAT32 ) return sizeof(float);
511     else if( Dtype == CIO::E_CIO_FLOAT64 ) return sizeof(double);
512     else return 0;
513
514 }
```

### 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 435 行で定義されています。

参照先 cio\_FileInfo::ArrayShape, と DFI\_Finfo.

```

436 {
437     return (CIO::E_CIO_ARRAYSHAPE)DFI_Finfo.ArrayShape;
438 }
```

### 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 426 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.

```

427 {
428     if( DFI_Finfo.ArrayShape == CIO::E_CIO_IJKN ) return D_CIO_IJNK;
429     if( DFI_Finfo.ArrayShape == CIO::E_CIO_NIJK ) return D_CIO_NIJK;
430     return " ";
431 }
```

**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 241 行で定義されています。

参照先 DFI\_Domain.

```
242 {  
243     return &DFI_Domain;  
244 }
```

**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 217 行で定義されています。

参照先 DFI\_Finfo.

```
218 {  
219     return &DFI_Finfo;  
220 }
```

**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 225 行で定義されています。

参照先 DFI\_Fpath.

```
226 {  
227     return &DFI_Fpath;  
228 }
```

**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 249 行で定義されています。

参照先 DFI\_MPI.

```
250 {  
251     return &DFI_MPI;  
252 }
```

#### 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 266 行で定義されています。

参照先 `DFI_Process`.

```
267 {  
268     return &DFI_Process;  
269 }
```

#### 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 257 行で定義されています。

参照先 `DFI_TimeSlice`.

```
258 {  
259     return &DFI_TimeSlice;  
260 }
```

#### 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 233 行で定義されています。

参照先 `DFI_Unit`.

```
234 {  
235     return &DFI_Unit;  
236 }
```

#### 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 881 行で定義されています。

参照先 DFI\_Finfo, と cio\_FileInfo::getComponentVariable().

```
882 {
883
884     return DFI_Finfo.getComponentVariable(pcomp);
885
886 }
```

### 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 449 行で定義されています。

参照先 cio\_FileInfo::DataType, と DFI\_Finfo.

```
450 {
451     return (CIO::E_CIO_DTYPE) DFI_Finfo.DataType;
452 }
```

### 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 442 行で定義されています。

参照先 ConvDatatypeE2S(), cio\_FileInfo::DataType, と DFI\_Finfo.

```
443 {
444     return ConvDatatypeE2S((CIO::E_CIO_DTYPE) DFI_Finfo.DataType);
445 }
```

### 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 525 行で定義されています。

参照先 DFI\_Domain, と cio\_Domain::GlobalDivision.

```
526 {
527     return DFI_Domain.GlobalDivision;
528 }
```

**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 518 行で定義されています。

参照先 DFI\_Domain, と cio\_Domain::GlobalVoxel.

```
519 {
520     return DFI_Domain.GlobalVoxel;
521 }
```

**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	<i>step</i>	取得するステップ
in	<i>compNo</i>	成分No(0 ~ n)
out	<i>min_value</i>	取得した min
out	<i>max_value</i>	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 901 行で定義されています。

参照先 DFI\_TimeSlice, と cio\_TimeSlice::getMinMax().

```
905 {
906
907     return DFI_TimeSlice.getMinMax(step, compNo, min_value, max_value);
908
909 }
```

**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 456 行で定義されています。

参照先 cio\_FileInfo::Component, と DFI\_Finfo.

```
457 {
458     return DFI_Finfo.Component;
459 }
```

**6.3.3.28** `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	<i>step</i>	取得するステップ
out	<i>vec_min</i>	取得した minmax の合成値
out	<i>vec_max</i>	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 890 行で定義されています。

参照先 DFI\_TimeSlice, と cio\_TimeSlice::getVectorMinMax().

```

893 {
894
895     return DFI_TimeSlice.getVectorMinMax(step, vec_min, vec_max);
896
897 }
```

## 6.3.3.29 static std::string cio\_DFI::getVersionInfo ( ) [inline],[static]

## バージョンを出力する

cio\_DFI.h の 879 行で定義されています。

参照先 CIO\_VERSION\_NO.

```

880 {
881     std::string str(CIO_VERSION_NO);
882     return str;
883 }
```

6.3.3.30 int cio\_DFI::MakeDirectory ( const std::string *path* ) [protected]

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

## 引数

in	<i>path</i>	パス
----	-------------	----

## 戻り値

error code

cio\_DFI.C の 740 行で定義されています。

参照先 MakeDirectorySub().

参照元 MakeDirectoryPath(), と WriteData().

```

741 {
742     int ret = MakeDirectorySub(path);
743     if( ret != 0 )
744     {
745         // 既存以外のエラー
746         if ( EEXIST != errno )
747         {
748             printf( "\tError(errno)=[%s]\n", strerror(errno) );
749             return 0;
750         }
751     }
752
753     // failed
754     return 1;
755 }
```

### 6.3.3.31 int cio\_DFI::MakeDirectoryPath ( ) [protected]

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 759 行で定義されています。

参照先 Generate\_Directory\_Path(), と MakeDirectory().

```

760 {
761     // DirectoryPath with TimeSlice
762     std::string path = Generate_Directory_Path();
763
764     return MakeDirectory(path);
765 }
```

### 6.3.3.32 int cio\_DFI::MakeDirectorySub ( std::string path ) [static], [protected]

ディレクトリパスの作成 (system 関数 mkdir で作成)

引数

in	<i>path</i>	パス
----	-------------	----

戻り値

error code

cio\_DFI.C の 768 行で定義されています。

参照先 CIO::cioPath\_DirName().

参照元 MakeDirectory().

```

769 {
770
771     umask(022);
772
773     int ret = mkdir(path.c_str(), 0777);
774     if( ret != 0 )
775     {
776         if( errno == EEXIST ) return 0;
777
778         std::string parent = CIO::cioPath_DirName(path);
779         int ret2 = MakeDirectorySub( parent );
780         if( ret2 != 0 )
781         {
782             return ret2;
783         }
784         ret = MakeDirectorySub( path );
785     }
786
787     return ret;
788
789 }
```

### 6.3.3.33 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

<i>in</i>	<i>scale</i>	スケール
-----------	--------------	------

cio\_DFI.C の 981 行で定義されています。

参照先 m\_intervalMngr, と cio\_Interval\_Mngr::normalizeBaseTime().

```

982 {
983   m_intervalMngr.normalizeBaseTime(scale);
984 }
```

#### 6.3.3.34 void cio\_DFI::normalizeDeltaT ( const double *scale* )

インターバルの DeltaT をスケールで無次元化する

引数

<i>in</i>	<i>scale</i>	スケール
-----------	--------------	------

cio\_DFI.C の 1009 行で定義されています。

参照先 m\_intervalMngr, と cio\_Interval\_Mngr::normalizeDeltaT().

```

1010 {
1011   m_intervalMngr.normalizeDeltaT(scale);
1012 }
```

#### 6.3.3.35 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

<i>in</i>	<i>scale</i>	スケール
-----------	--------------	------

cio\_DFI.C の 988 行で定義されています。

参照先 m\_intervalMngr, と cio\_Interval\_Mngr::normalizeIntervalTime().

```

989 {
990   m_intervalMngr.normalizeIntervalTime(scale);
991 }
```

#### 6.3.3.36 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

<i>in</i>	<i>scale</i>	スケール
-----------	--------------	------

cio\_DFI.C の 1002 行で定義されています。

参照先 m\_intervalMngr, と cio\_Interval\_Mngr::normalizeLastTime().

```

1003 {
1004   m_intervalMngr.normalizeLastTime(scale);
1005 }
```

#### 6.3.3.37 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

<i>in</i>	<i>scale</i>	スケール
-----------	--------------	------

cio\_DFI.C の 995 行で定義されています。

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

```

996 {
997     m_intervalMngr.normalizeStartTime(scale);
998 }
```

### 6.3.3.38 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

<i>in</i>	<i>scale</i>	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 974 行で定義されています。

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

```

975 {
976     return m_intervalMngr.normalizeTime(scale);
977 }
```

### 6.3.3.39 virtual CIO::E\_CIO\_ERRORCODE cio\_DFI::read\_averaged ( FILE \* *fp*, bool *matchEndian*, unsigned *step*, unsigned & *avr\_step*, double & *avr\_time* ) [protected], [pure virtual]

sph ファイルのAverage データレコードの読み込み

引数

<i>in</i>	<i>fp</i>	ファイルポインタ
<i>in</i>	<i>matchEndian</i>	true:Endian 一致
<i>in</i>	<i>step</i>	読み込み step 番号
<i>out</i>	<i>avr_step</i>	平均ステップ
<i>out</i>	<i>avr_time</i>	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()`.

### 6.3.3.40 virtual CIO::E\_CIO\_ERRORCODE cio\_DFI::read\_Datarecord ( FILE \* *fp*, bool *matchEndian*, cio\_Array \* *buf*, int *head*[3], int *nz*, cio\_Array \*& *src* ) [protected], [pure virtual]

フィールドデータファイルのデータレコード読み込み

引数

<i>in</i>	<i>fp</i>	ファイルポインタ
<i>in</i>	<i>matchEndian</i>	true:Endian 一致
<i>in</i>	<i>buf</i>	読み込み用バッファ

in	<i>head</i>	読み込みバッファHeadIndex
in	<i>nz</i>	z 方向のボクセルサイズ ( 実セル + ガイドセル * 2 )
out	<i>src</i>	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\(\)](#).

6.3.3.41 `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 voysize[3], double & time )` [protected], [pure virtual]

フィールドデータファイルのヘッダーレコード読み

引数

in	<i>fp</i>	ファイルポインタ
in	<i>matchEndian</i>	true:Endian 一致
in	<i>step</i>	ステップ番号
in	<i>head</i>	dfi のHeadIndex
in	<i>tail</i>	dfi のTailIndex
in	<i>gc</i>	dfi のガイドセル数
out	<i>voysize</i>	voysize
out	<i>time</i>	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\(\)](#).

6.3.3.42 `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 }

```

**6.3.3.43** `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 }

```

**6.3.3.44** `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	<i>ret</i>	終了コード 1:正常、1 以外：エラー
in	<i>step</i>	入力ステップ番号
in	<i>gc</i>	仮想セル数
in	<i>Gvoxel</i>	グローバルボクセルサイズ
in	<i>Gdivision</i>	領域分割数
in	<i>head</i>	計算領域の開始位置
in	<i>tail</i>	計算領域の終了位置

out	<i>time</i>	読み込んだ時間
in	<i>mode</i>	平均ステップ & 時間読み込みフラグ    false : 読み込み true : 読み込まない
out	<i>step_avr</i>	平均ステップ
out	<i>time_avr</i>	平均時間

戻り値

読み込んだフィールドデータのポインタ

参照元 ReadData().

**6.3.3.45**    `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み込む

引数

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

戻り値

終了コード 1:正常 1 以外:エラー

**6.3.3.46**    `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	<i>val</i>	読み込み先の配列をポインタで渡す
in	<i>step</i>	読み込むステップ番号
in	<i>gc</i>	仮想セル数
in	<i>Gvoxel</i>	グローバルボクセルサイズ
in	<i>Gdivision</i>	領域分割数

in	<i>head</i>	計算領域の開始位置
in	<i>tail</i>	計算領域の終了位置
out	<i>time</i>	読み込んだ時間
in	<i>mode</i>	平均ステップ & 時間読み込みフラグ    false : 読み込み true : 読み込まない
out	<i>step_avr</i>	平均ステップ
out	<i>time_avr</i>	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

< 読み込み判定フラグ

読み込みフラグ取得

粗密フラグセット

読み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

読み込み領域 start end の取得

読み込み方法の取得

フィールドデータの読み込み

読み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
35   int Shead[3];
36   for(int i=0; i<3; i++) Shead[i] = head[i];
37   dst->setHeadIndex(Shead);
38
39   std::string dir = CIO::cioPath_DirName(m_indexDfiName);
40
41   bool mio = false;
42   bool isSame =true;
43   CIO::E_CIO_READTYPE readflag;
44
45   readflag = CheckReadType(Gvoxel, DFI_Domain.GlobalVoxel,
46                             Gdivision, DFI_Domain.GlobalDivision);
47
48   if( readflag == CIO::E_CIO_SAMEDIV_REFINEMENT || readflag ==
49       CIO::E_CIO_DIFFDIV_REFINEMENT ) isSame = false;
50
51   ret = DFI_Process.CheckReadRank(DFI_Domain, head, tail, readflag,
52                                   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
68      std::string fname;
69      if( CIO::cioPath_isAbsolute(DFI_Finfo.DirectoryPath) ){
70          fname = Generate_FieldFileName(ID, step, mio);
71      } else {
72          std::string tmp = Generate_FieldFileName(ID, step, mio);
73          fname = CIO::cioPath_ConnectPath( dir, tmp );
74      }
75
76      int copy_sta[3], copy_end[3], read_sta[3], read_end[3];
77
78      CreateReadStartEnd(isSame, head, tail, gc, DFI_Process.RankList[n].HeadIndex,
79                          DFI_Process.RankList[n].TailIndex,
80                          DFI_Finfo.GuideCell, readflag,
81                          copy_sta, copy_end, read_sta, read_end);
82
83      cio_Array* src = ReadFieldData(fname, step, time, read_sta, read_end,
84                                    DFI_Process.RankList[n].HeadIndex,
85                                    DFI_Process.RankList[n].TailIndex,
86                                    avr_mode, avr_step, avr_time, ret);
87
88      if( ret != CIO::E_CIO_SUCCESS ) {
89          delete src;
90          return ret;
91      }
92
93      if( m_RankID == 0 ) {
94          printf("\t[%s] has read :\tstep=%d  time=%e ]\n", fname.c_str(), step, time);
95      }
96
97      src->setHeadIndex(read_sta);
98
99      if( !isSame ) {
100          cio_Array *temp = src;
101          int err;
102          src = cio_Array::interp_coarse(temp, err, false);
103          delete temp;
104      }
105
106      src->copyArray(copy_sta, copy_end, dst);
107      delete src;
108  }
109
110  return CIO::E_CIO_SUCCESS;
111
112 }

```

**6.3.3.47** `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 )` [protected], [virtual]

read field data record(sph or bov)

引数

in	<i>fname</i>	FieldData ファイル名
in	<i>step</i>	読み込みステップ番号
out	<i>time</i>	読み込んだ時間
in	<i>sta</i>	読み込みスタート位置
in	<i>end</i>	読み込みエンド位置

in	<i>DFI_head</i>	dfi のHeadIndex
in	<i>DFI_tail</i>	dfi のTailIndex
in	<i>avr_mode</i>	平均ステップ & 時間読み込みフラグ    false : 読み込み

true : 読み込まない

引数

out	<i>avr_step</i>	平均ステップ
out	<i>avr_time</i>	平均時間
out	<i>ret</i>	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み込み

< 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
147     FILE* fp;
148     if( !(fp=fopen(fname.c_str(),"rb")) ) {
149         printf("Can't open file. (%s)\n",fname.c_str());
150         ret = CIO::E_CIO_ERROR_OPEN_FIELDDATA;
151         return NULL;
152     }
153
154
155     int idumy = 1;
156     char* cdumy = (char*)(&idumy);
157     CIO::E_CIO_ENDIANTYPE Endian=CIO::E_CIO_ENDIANTYPE_UNKNOWN;
158     if( cdumy[0] == 0x01 ) Endian = CIO::E_CIO_LITTLE;
159     if( cdumy[0] == 0x00 ) Endian = CIO::E_CIO_BIG;
160
161     bool matchEndian = true;
162     if( Endian != DFI_Finfo.Endian ) matchEndian = false;
163
164     //RealType real_type;
165     int voxsize[3];
166     ret = read_HeaderRecord(fp, matchEndian, step, DFI_head, DFI_tail,
167                           DFI_Finfo.GuideCell, voxsize, time);
168     if( ret != CIO::E_CIO_SUCCESS )
169     {
170         ret = CIO::E_CIO_ERROR_READ_FIELD_HEADER_RECORD;
171         printf("**** read error\n");
172         fclose(fp);
173         return NULL;
174     }
175
176
177     int sz[3];
178     for(int i=0; i<3; i++) sz[i]=voxsize[i]-2*DFI_Finfo.GuideCell;
179
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 読み込み
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 }

```

**6.3.3.48** cio\_DFI \* cio\_DFI::ReadInit ( const MPI\_Comm comm, const std::string dffile, 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 コミュニケーター
in	dffile	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 の読み込み

FilePath の読み込み

Unit の読み込み

TimeSlice の読み込み

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 の読み込み

MPI の読み込み

Process の読み込み

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\_OPENERERROR, CIO::E\_CIO\_ERROR\_READ\_MPI, CIO::E\_CIO\_ERROR\_READ\_PROCESS, CIO::E\_CIO\_ERROR\_READ\_PROCFILE\_OPENERERROR, 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_OPENERERROR;
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_OPENERERROR;
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
169 tpCntl.remove();
170
171 cio_DFI *dfi = NULL;
172 if( F_info.FileFormat == CIO::E_CIO_FMT_SPH ) {
173     dfi = new cio_DFI_SPH(F_info, F_path, unit, domain, mpi, TimeSlice, process);
174 } else if( F_info.FileFormat == CIO::E_CIO_FMT_BOV ) {
175     dfi = new cio_DFI_BOV(F_info, F_path, unit, domain, mpi, TimeSlice, process);
176 } else {
177     return NULL;
178 }
179
180 //読み込みタイプのチェック
181 dfi->m_read_type = dfi->CheckReadType(G_Voxel, dfi->DFI_Domain.GlobalVoxel,
182                                     G_Div, dfi->DFI_Domain.GlobalDivision);
183 if( dfi->m_read_type == CIO::E_CIO_READTYPE_UNKNOWN ) {
184     printf("\tDimension size error (%d %d %d)\n",
185           G_Voxel[0], G_Voxel[1], G_Voxel[2]);
186     ret = CIO::E_CIO_ERROR_INVALID_DIVNUM;
187     return NULL;
188 }
189
190 #if 0
191 if( dfi->m_start_type == E_CIO_SAMEDIV_SAMERES ) {
192     printf("***** SAMEDIV_SAMERES\n");
193 } else if( dfi->m_start_type == E_CIO_SAMEDIV_REFINEMENT ) {
194     printf("***** SAMEDIV_REFINEMENT\n");
195 } else if( dfi->m_start_type == E_CIO_DIFFDIV_SAMERES ) {
196     printf("***** DIFFDIV_SAMERES\n");
197 } else if( dfi->m_start_type == E_CIO_DIFFDIV_REFINEMENT ) {
198     printf("***** DIFFDIV_REFINEMENT\n");
199 }
200 #endif
201 dfi->m_comm = comm;
202 dfi->m_indexDfiName = DfiName;
203 dfi->m_RankID = RankID;
204
205 ret = CIO::E_CIO_SUCCESS;
206
207 return dfi;
208 }

```

### 6.3.3.49 void cio\_DFI::setComponentVariable ( int pcomp, std::string compName )

FileInfo の成分名を登録する

引数

in	<i>pcomp</i>	成分位置 0:u, 1:v, 2:w
in	<i>compName</i>	成分名 "u","v","w",,,

cio\_DFI.C の 872 行で定義されています。

参照先 DFI\_Finfo, と cio\_FileInfo::setComponentVariable().

```

873 {
874
875     DFI_Finfo.setComponentVariable(pcomp, compName);
876
877 }

```

### 6.3.3.50 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	<i>interval_step</i>	インターバルステップ
in	<i>base_step</i>	基準となるステップ (デフォルト 0 ステップ)
in	<i>start_step</i>	セッション開始ステップ (デフォルト 0 ステップ)
in	<i>last_step</i>	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 927 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().

```

931 {
932     //インターバルモードの登録 (ステップ)
933     m_intervalMngr.setMode(cio_Interval_Mngr::By_step);
934
935     //インターバルステップの登録
936     m_intervalMngr.setInterval(interval_step);
937
938     //インターバル開始ステップの登録
939     m_intervalMngr.setStart(start_step);
940
941     //インターバル最終ステップの登録
942     m_intervalMngr.setLast(last_step);
943
944     //トリガーの初期化
945     m_intervalMngr.initTrigger(base_step, 0.0, 0.0);
946 }
```

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

cio\_DFI.C の 950 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().

```

955 {
956     //インターバルモードの登録 (タイム)
957     m_intervalMngr.setMode(cio_Interval_Mngr::By_time);
958
959     //インターバルタイムの登録
960     m_intervalMngr.setInterval(interval_time);
961
962     //インターバル開始タイムの登録
963     m_intervalMngr.setStart(start_time);
964
965     //インターバル終了タイムの登録
966     m_intervalMngr.setLast(last_time);
967
968     //トリガーの初期化
969     m_intervalMngr.initTrigger(0, base_time, dt);
970 }
```

**6.3.3.52** 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 865 行で定義されています。

参照先 DFI\_Finfo, と cio\_FileInfo::TimeSliceDirFlag.

```
866 {
867   DFI_Finfo.TimeSliceDirFlag == ONOFF;
868 }
```

**6.3.3.53** 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	<i>fp</i>	ファイルポインタ
in	<i>step_avr</i>	平均ステップ番号
in	<i>time_avr</i>	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().

**6.3.3.54** 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	<i>fp</i>	ファイルポインタ
in	<i>val</i>	データポインタ
in	<i>gc</i>	ガイドセル
in	<i>RankID</i>	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().

**6.3.3.55** 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	<i>fp</i>	ファイルポインタ
in	<i>step</i>	ステップ番号
in	<i>time</i>	時刻
in	<i>RankID</i>	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().

**6.3.3.56** `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 }
```

**6.3.3.57** `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、 $\text{minmax}[0]=\min$   $\text{minmax}[1]=\max$  ベクトルのとき、 $\text{minmax}[0]=$ 成分 1 の  $\min$   $\text{minmax}[1]=$ 成分 1 の  $\max$  ...  $\text{minmax}[2n-2]=$ 成分  $n$  の  $\min$   $\text{minmax}[2n-1]=$ 成分  $n$  の  $\max$   $\text{minmax}[2n]=$ 合成値の  $\min$   $\text{minmax}[2n+1]=$ 合成値の  $\max$



## 引数

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

参照元 WriteData().

**6.3.3.58 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	<i>step</i>	出力ステップ番号
in	<i>gc</i>	仮想セル数
in	<i>time</i>	出力時刻
in	<i>val</i>	出力データポインタ
in	<i>minmax</i>	フィールドデータのMinMax
in	<i>avr_mode</i>	平均ステップ & 時間出力    false : 出力 true : 出力しない
in	<i>step_avr</i>	平均ステップ
in	<i>time_avr</i>	平均時間
in	<i>force</i>	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 //フィールドデータの出力
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 dfname = CIO::cioPath_FileName(m_indexDfiName,".dfi");
257 std::string fname = CIO::cioPath_ConnectPath( m_directoryPath, dfname );
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 }

```

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

```

### 6.3.3.60 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\_OPENERERROR, CIO::E\_CIO\_ERROR\_WRITE\_INDEXFILENAME\_EMPTY, CIO::E\_CIO\_ERROR\_WRITE\_PREFIX\_EMPTY, CIO::E\_CIO\_ERROR\_WRITE\_TIMESLICE, CIO::E\_CIO\_ERROR\_WRITE\_UNIT, CIO::E\_CIO\_SUCCESS, cio\_FileInfo::Prefix, cio\_FilePath::Write(), cio\_FileInfo::Write(), cio\_TimeSlice::Write(), と cio\_Unit::Write().

参照元 WriteData().

```

21 {
22
23     if ( dfi_name.empty() ) return CIO::E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY;
24     if ( DFI_Finfo.Prefix.empty() ) return CIO::E_CIO_ERROR_WRITE_PREFIX_EMPTY;
25
26     FILE* fp = NULL;
27
28     // File exist ?
29     bool flag = false;
30     if ( fp = fopen(dfi_name.c_str(), "r" ) )
31     {
32         flag = true;
33         fclose(fp);
34     }
35
36     if( !(fp = fopen(dfi_name.c_str(), "w")) )
37     {
38         fprintf(stderr, "Can't open file.(%s)\n", dfi_name.c_str());
39         return CIO::E_CIO_ERROR_WRITE_INDEXFILE_OPENERERROR;
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 }

```

**6.3.3.61** `cio_DFI * cio_DFI::Writelnit ( 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	<i>comm</i>	MPI コミュニケーター
in	<i>DfiName</i>	DFI ファイル名
in	<i>Path</i>	フィールドデータのディレクトリ
in	<i>prefix</i>	ベースファイル名
in	<i>format</i>	ファイルフォーマット
in	<i>GCell</i>	出力仮想セル数
in	<i>DataType</i>	データタイプ
in	<i>ArrayShape</i>	配列形状
in	<i>nComp</i>	成分数
in	<i>proc_fname</i>	proc.dfi ファイル名
in	<i>G_size</i>	グローバルボクセルサイズ
in	<i>pitch</i>	ピッチ
in	<i>G_origin</i>	原点座標値
in	<i>division</i>	領域分割数
in	<i>head</i>	計算領域の開始位置
in	<i>tail</i>	計算領域の終了位置
in	<i>hostname</i>	ホスト名
in	<i>TSliceOnOff</i>	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 273 行で定義されています。

291 {

```

292
293 // float 型を double 型に変換して double 版 WriteInit 関数を呼ぶ
294
295 double d_pch[3], d_org[3];
296 for(int i=0; i<3; i++) {
297     d_pch[i]=(double)pitch[i];
298     d_org[i]=(double)G_origin[i];
299 }
300
301 return WriteInit(comm,
302                 DfiName,
303                 Path,
304                 prefix,
305                 format,
306                 GCell,
307                 DataType,
308                 ArrayShape,
309                 nComp,
310                 proc_fname,
311                 G_size,
312                 d_pch,
313                 d_org,
314                 division,
315                 head,
316                 tail,
317                 hostname,
318                 TSliceOnOff);
319
320 }

```

**6.3.3.62** `cio_DFI * cio_DFI::Writelnit ( 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	<i>comm</i>	MPI コミュニケーター
in	<i>DfiName</i>	DFI ファイル名
in	<i>Path</i>	フィールドデータのディレクトリ
in	<i>prefix</i>	ベースファイル名
in	<i>format</i>	ファイルフォーマット
in	<i>GCell</i>	出力仮想セル数
in	<i>DataType</i>	データタイプ
in	<i>ArrayShape</i>	配列形状
in	<i>nComp</i>	成分数
in	<i>proc_fname</i>	proc.dfi ファイル名
in	<i>G_size</i>	グローバルボクセルサイズ
in	<i>pitch</i>	ピッチ
in	<i>G_origin</i>	原点座標値
in	<i>division</i>	領域分割数
in	<i>head</i>	計算領域の開始位置
in	<i>tail</i>	計算領域の終了位置
in	<i>hostname</i>	ホスト名
in	<i>TSliceOnOff</i>	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 324 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.

```

342 {
343
344     cio_DFI *dfi = NULL;
345
346     int RankID;
347     MPI_Comm_rank( comm, &RankID );
348
349     int nrank;
350     MPI_Comm_size( comm, &nrank );
351
352     cio_FileInfo out_F_info;
353     out_F_info.DirectoryPath = Path;
354     out_F_info.TimeSliceDirFlag = TSliceOnOff;
355     out_F_info.Prefix = prefix;
356     out_F_info.FileFormat = format;
357     out_F_info.GuideCell = GCell;
358     out_F_info.DataType = DataType;
359     out_F_info.ArrayShape = ArrayShape;
360     out_F_info.Component = nComp;
361
362     int idumy = 1;
363     char* cdumy = (char*)&idumy;
364     if( cdumy[0] == 0x01 ) out_F_info.Endian = CIO::E_CIO_LITTLE;
365     if( cdumy[0] == 0x00 ) out_F_info.Endian = CIO::E_CIO_BIG;
366
367     cio_FilePath out_F_path;
368     out_F_path.ProcDFIFile = proc_fname;
369
370     cio_Unit out_unit;
371
372     cio_MPI out_mpi;
373     out_mpi.NumberOfRank = nrank;
374     out_mpi.NumberOfGroup = 1;
375
376     cio_Domain out_domain;
377     cio_Process out_Process;
378     cio_Rank out_Rank;
379
380     for(int i=0; i<nrank; i++ ) {
381         out_Process.RankList.push_back(out_Rank);
382     }
383
384     out_Process.RankList[RankID].RankID=RankID;
385     out_Process.RankList[RankID].HostName=hostname;
386     for(int i=0; i<3; i++) {
387         out_Process.RankList[RankID].HeadIndex[i]=head[i];
388         out_Process.RankList[RankID].TailIndex[i]=tail[i];
389         out_Process.RankList[RankID].VoxelSize[i]=tail[i]-head[i]+1;
390     }
391
392     for(int i=0; i<3; i++) {
393         out_domain.GlobalVoxel[i] = G_size[i];
394         out_domain.GlobalDivision[i] = division[i];
395         out_domain.GlobalOrigin[i] = G_origin[i];
396         out_domain.GlobalRegion[i] = pitch[i]*G_size[i];
397     }
398
399     cio_TimeSlice out_TSlice;
400
401     char tmpname[512];
402     memset(tmpname,0x00,sizeof(char)*512);
403     if( gethostname(tmpname, 512) != 0 ) printf("*** error gethostname() \n");
404
405     if( out_F_info.FileFormat == CIO::E_CIO_FMT_SPH ) {
406         dfi = new cio_DFI_SPH(out_F_info, out_F_path, out_unit, out_domain, out_mpi,
407                               out_TSlice, out_Process);
408     } else if( out_F_info.FileFormat == CIO::E_CIO_FMT_BOV ) {
409
410         dfi = new cio_DFI_BOV(out_F_info, out_F_path, out_unit, out_domain, out_mpi,
411                               out_TSlice, out_Process);
412     } else return NULL;
413
414
415     dfi->m_indexDfiName = DfiName;
416     dfi->m_directoryPath = CIO::cioPath_DirName(DfiName);
417     dfi->m_comm = comm;
418     dfi->m_RankID = RankID;
419
420     return dfi;
421

```

422 }

**6.3.3.63 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     //origin 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 }
```

**6.3.3.64 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     //origin 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_OPENERERROR;
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     }

```



```

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(), GetData\_Type(), GetData\_TypeString(), 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(), と WriteIndexDfiFile().

#### 6.3.4.8 MPI\_Comm cio\_DFI::m\_comm [protected]

MPI コミュニケーター

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(), normalizeDeltT(), 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]

読み込みタイプ

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]

読み込みランクリスト

cio\_DFI.h の 66 行で定義されています。

参照元 ReadData().

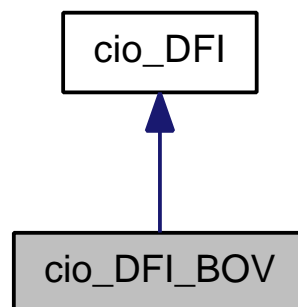
このクラスの説明は次のファイルから生成されました:

- [cio\\_DFI.h](#)
- [cio\\_DFI.C](#)
- [cio\\_DFI\\_Read.C](#)
- [cio\\_DFI\\_Write.C](#)
- [cio\\_DFI\\_inline.h](#)

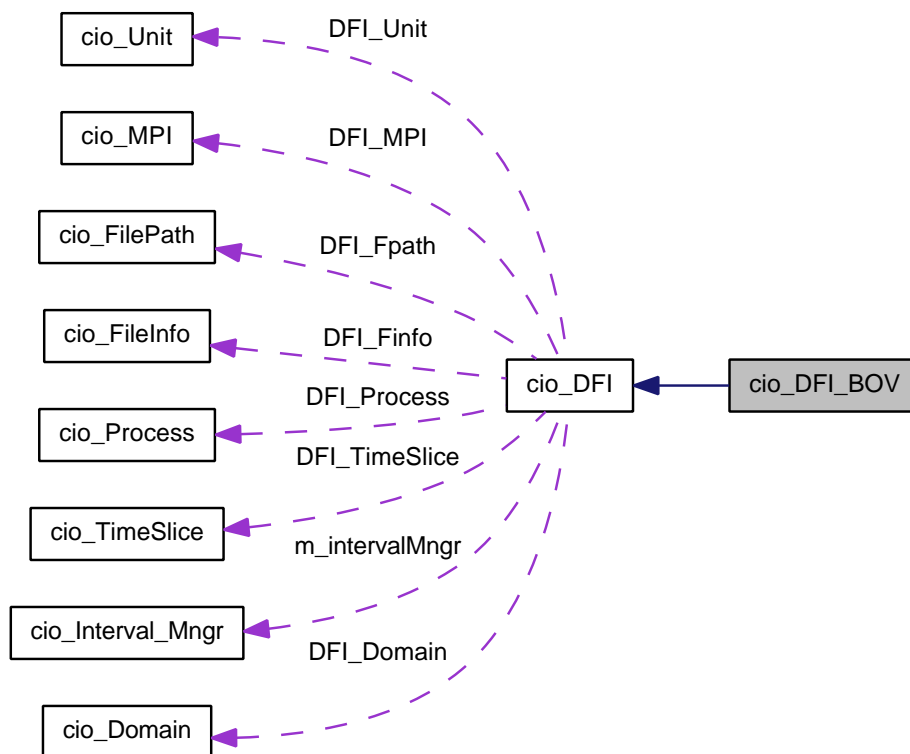
## 6.4 クラス cio\_DFI\_BOV

```
#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み込み*
- `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み込み*
- `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み込み*
- `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	<i>F_Info</i>	FileInfo
in	<i>F_Path</i>	FilePath
in	<i>unit</i>	Unit
in	<i>domain</i>	Domain
in	<i>mpi</i>	MPI
in	<i>TSlice</i>	TimeSlice
in	<i>process</i>	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み込み

## 引数

in	<i>fp</i>	ファイルポインタ
in	<i>matchEndian</i>	true:Endian 一致
in	<i>step</i>	読み込み step 番号
out	<i>avr_step</i>	平均ステップ
out	<i>avr_time</i>	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み

## 引数

in	<i>fp</i>	ファイルポインタ
in	<i>matchEndian</i>	true:Endian 一致
in	<i>buf</i>	読み込み用バッファ
in	<i>head</i>	読み込みバッファHeadIndex
in	<i>nz</i>	z 方向のボクセルサイズ ( 実セル + ガイドセル * 2 )
out	<i>src</i>	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 *voysize*[3], double & *time* )** [protected],[virtual]

bov ファイルのヘッダーレコード読み込み

引数

in	<i>fp</i>	ファイルポインタ
in	<i>matchEndian</i>	エンディアンチェックフラグ true:合致
in	<i>step</i>	ステップ番号
in	<i>head</i>	dfi のHeadIndex
in	<i>tail</i>	dfi のTailIndex
in	<i>gc</i>	dfi のガイドセル数
out	<i>voysize</i>	voysize
out	<i>time</i>	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     time=0.0;
46     for(int i=0; i<DFI_TimeSlice.SliceList.size(); i++) {
47         if( DFI_TimeSlice.SliceList[i].step == step ) {
48             time=(double)DFI_TimeSlice.SliceList[i].time;
49         }
50     }
51 }
52
53 for(int i=0; i<3; i++) voysize[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	<i>fp</i>	ファイルポインタ
in	<i>step_avr</i>	平均ステップ番号
in	<i>time_avr</i>	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	<i>fp</i>	ファイルポインタ
in	<i>val</i>	データポインタ
in	<i>gc</i>	仮想セル数
in	<i>RankID</i>	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 ヘッドファイルの出力

引数

in	<i>fp</i>	ファイルポインタ
in	<i>step</i>	ステップ番号
in	<i>time</i>	時刻
in	<i>RankID</i>	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 }
```

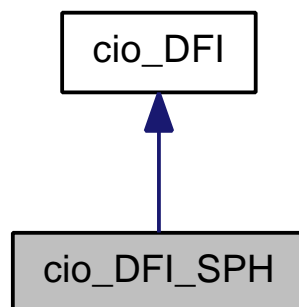
このクラスの説明は次のファイルから生成されました:

- [cio\\_DFI\\_BOV.h](#)
- [cio\\_DFI\\_BOV.C](#)

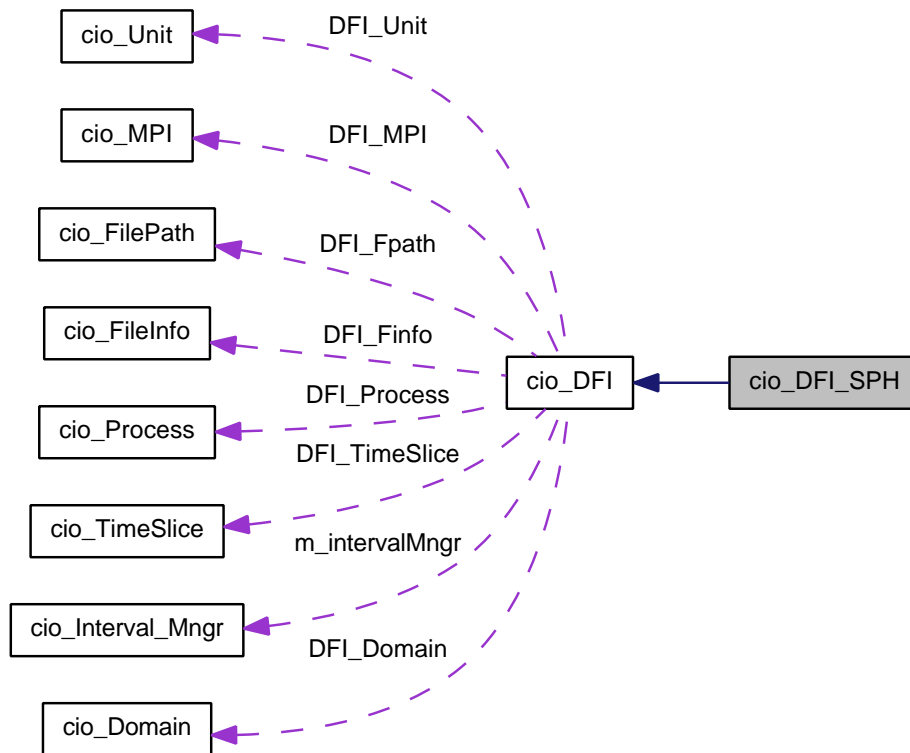
## 6.5 クラス [cio\\_DFI\\_SPH](#)

```
#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み込み*
- `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み込み*
- `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み込み*
- `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	<i>F_Info</i>	FileInfo
in	<i>F_Path</i>	FilePath
in	<i>unit</i>	Unit
in	<i>domain</i>	Domain
in	<i>mpi</i>	MPI
in	<i>TSlice</i>	TimeSlice
in	<i>process</i>	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み込み

## 引数

in	<i>fp</i>	ファイルポインタ
in	<i>matchEndian</i>	true:Endian 一致
in	<i>step</i>	読み込み step 番号
out	<i>avr_step</i>	平均ステップ
out	<i>avr_time</i>	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み込み

引数

in	fp	ファイルポインタ
in	matchEndian	true:Endian 一致
in	buf	読み込み用バッファ
in	head	読み込みバッファ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 の読み込み

```

```

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 インデックスをずらす
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 の読み込み
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み込み

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(voysize, sizeof(int), 3, fp) != 3 ){fclose(fp);return
CIO::E_CIO_ERROR_READ_SPH_REC2;}
91     if( !matchEndian ) {
92         BSWAP32(voysize[0]);
93         BSWAP32(voysize[1]);
94         BSWAP32(voysize[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     voysize[0]=(int)tmp[0];
105     voysize[1]=(int)tmp[1];
106     voysize[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	<i>fp</i>	ファイルポインタ
in	<i>val</i>	データポインタ
in	<i>gc</i>	ガイドセル
in	<i>RankID</i>	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	<i>fp</i>	ファイルポインタ
in	<i>step</i>	ステップ番号
in	<i>time</i>	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	<code>_GlobalOrigin</code>	起点座標
in	<code>_GlobalRegion</code>	各軸方向の長さ

in	<code>_GlobalVoxel</code>	ボクセル数
in	<code>_GlobalDivision</code>	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	<code>tpCntl</code>	<code>cio_TextParser</code> 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	<i>fp</i>	ファイルポインタ
in	<i>tab</i>	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\_MPL::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().



- 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	<i>_DirectoryPath</i>	ディレクトリパス
----	-----------------------	----------

in	<code>_TimeSliceDirFlag</code>	TimeSlice on or off
in	<code>_Prefix</code>	ファイル接頭文字
in	<code>_FileFormat</code>	ファイルフォーマット
in	<code>_GuideCell</code>	仮想セルの数
in	<code>_DataType</code>	配列のデータタイプ
in	<code>_Endian</code>	エンディアンタイプ
in	<code>_ArrayShape</code>	配列形状
in	<code>_Component</code>	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	<code>pcomp</code>	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\_GUIDECCELL, 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_GUIDECCELL;
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(),"nijkn" ) ) {
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( !strcasemp(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	<i>pcomp</i>	成分位置 0:u, 1:v, 2:w
in	<i>compName</i>	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	<i>fp</i>	ファイルポインタ
in	<i>tab</i>	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::Writeln().

#### 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::Writeln().

#### 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::Writeln().

#### 6.7.4.5 std::string cio\_FileInfo::DirectoryPath

フィールド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::Writeln().

#### 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::Writeln().

#### 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(index.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 ファイル名の読み込み

##### 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	<i>fp</i>	ファイルポインタ
in	<i>tab</i>	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 `normalizeDeltaT` (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  $t$

### 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\_DFl::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\_DFl::setIntervalStep(), と cio\_DFl::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\_DFl::setIntervalStep(), と cio\_DFl::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\_DFl::setIntervalStep(), と cio\_DFl::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\_DFl::setIntervalStep(), と cio\_DFl::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 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	<a href="#">_NumberOfRank</a>	プロセス数
----	-------------------------------	-------

in	<i>_NumberOfGroup</i>	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	<i>tpCntl</i>	cio_TextParser クラス
in	<i>domain</i>	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	<i>fp</i>	ファイルポインタ
in	<i>tab</i>	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)  
*読み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)  
*読み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 isMatchEndianSbdomMagick (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 )

読み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、読み込みランクリスト readRankList を生成する

引数

in	<i>dfi_domain</i>	DFI の domain 情報
in	<i>head</i>	ソルバーのHeadIndex
in	<i>tail</i>	ソルバーのTailIndex

in	<i>readflag</i>	読み込み方法
out	<i>readRankList</i>	読み込みランクリスト

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     headT mapHeadX, mapHeadY, mapHeadZ;
212
213     //DFI に Process/Rank[0] がない処理
214     if( RankList.empty() ) {
215         CIO::E_CIO_ERRORCODE ret = CreateRankList(dfi_domain, mapHeadX, mapHeadY, mapHeadZ);
216         if( ret != CIO::E_CIO_SUCCESS ) return ret;
217     }
218
219     //rankMap が未定義 ( DFI に Process/Rank[0] がある場合 )
220     if( m_rankMap == NULL ) {
221         m_rankMap = CreateRankMap(dfi_domain.GlobalDivision, mapHeadX, mapHeadY, mapHeadZ);
222     }
223
224     //mapHeadX, mapHeadY, mapHeadZ が未定義
225     if( mapHeadX.empty() || mapHeadY.empty() || mapHeadZ.empty() ) {
226         std::set<int> headx, heady, headz;
227         for(int i=0; i<RankList.size(); i++ ) {
228             headx.insert(RankList[i].HeadIndex[0]);
229             heady.insert(RankList[i].HeadIndex[1]);
230             headz.insert(RankList[i].HeadIndex[2]);
231         }
232         CreateHeadMap(headx, mapHeadX);
233         CreateHeadMap(heady, mapHeadY);
234         CreateHeadMap(headz, mapHeadZ);
235     }
236
237     return CheckStartEnd(dfi_domain, head, tail, readflag, mapHeadX, mapHeadY, mapHeadZ, ReadRankList);
238 }
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 )**

読み込みランクファイルリストの作成

引数

in	<i>dfi_domain</i>	DFI のDomain 情報
in	<i>head</i>	計算領域の開始位置
in	<i>tail</i>	計算領域の終了位置
in	<i>readflag</i>	粗密データ判定フラグ
in	<i>mapHeadX</i>	headX をキーにした位置情報マップ
in	<i>mapHeadY</i>	headY をキーにした位置情報マップ
in	<i>mapHeadZ</i>	headZ をキーにした位置情報マップ
out	<i>readRankList</i>	読み込みに必要なランク番号リスト

cio\_Process.C の 608 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().

```

616 {
617
618     int StartEnd[6];
619
620     int ndiv = dfi_domain.GlobalDivision[0]*
621               dfi_domain.GlobalDivision[1]*
622               dfi_domain.GlobalDivision[2];
623     int head2[3],tail2[3];
624     if( readflag == CIO::E_CIO_SAMEDIV_SAMERES || readflag ==
        CIO::E_CIO_DIFFDIV_SAMERES ) {
625         for(int i=0; i<3; i++) {
626             head2[i]=head[i];
627             tail2[i]=tail[i];
628         }
629     } else {
630         for(int i=0; i<3; i++) {
631             if( head[i] < 0 ) head2[i]=head[i]/2;
632             else head2[i]=(head[i]+1)/2;
633             if( tail[i] < 0 ) tail2[i]=tail[i]/2;
634             else tail2[i]=(tail[i]+1)/2;
635         }
636     }
637
638     //x 方向の絞り込み
639     for( headT::iterator it=mapHeadX.begin(); it!=mapHeadX.end(); it++ )
640     {
641         if( head2[0] >= (*it).first ) StartEnd[0] = (*it).second;
642         if( tail2[0] >= (*it).first ) StartEnd[3] = (*it).second;
643         else break;
644     }
645
646     //y 方向の絞り込み
647     for( headT::iterator it=mapHeadY.begin(); it!=mapHeadY.end(); it++ )
648     {
649         if( head2[1] >= (*it).first ) StartEnd[1] = (*it).second;
650         if( tail2[1] >= (*it).first ) StartEnd[4] = (*it).second;
651         else break;
652     }
653
654     //z 方向の絞り込み
655     for( headT::iterator it=mapHeadZ.begin(); it!=mapHeadZ.end(); it++ )
656     {
657         if( head2[2] >= (*it).first ) StartEnd[2] = (*it).second;
658         if( tail2[2] >= (*it).first ) StartEnd[5] = (*it).second;
659         else break;
660     }
661
662     readRankList.clear();
663
664     for(int k=StartEnd[2]; k<=StartEnd[5]; k++) {
665         for(int j=StartEnd[1]; j<=StartEnd[4]; j++) {
666             for(int i=StartEnd[0]; i<=StartEnd[3]; i++) {
667                 int rank = m_rankMap[_CIO_IDX_IJK(i, j, k, dfi_domain.GlobalDivision[0],
668                 dfi_domain.GlobalDivision[1],
669                 dfi_domain.GlobalDivision[2], 0)];
670                 if( rank<0 ) continue;
671
672                 readRankList.push_back(rank);
673             }
674         }
675     }
676     return CIO::E_CIO_SUCCESS;
677 }

```

#### 6.11.4.3 void cio\_Process::CreateHeadMap ( std::set< int > head, headT & map )

head map の生成

引数

in	<i>head</i>	head インデックス
out	<i>map</i>	head map

cio\_Process.C の 527 行で定義されています。

参照元 CheckReadRank(), CreateRankList(), と CreateRankMap().

```

529 {
530
531     map.clear();

```



```

532
533     int cnt=0;
534     for (std::set<int>::iterator it=head.begin(); it!=head.end(); it++)
535     {
536         int key=*it;
537         map.insert(headT::value_type(key, cnt));
538         cnt++;
539     }
540 }

```

#### 6.11.4.4 void cio\_Process::CreateHeadMap ( int \* head, int ndiv, headT & map )

head map の生成

引数

in	<i>head</i>	head インデックス
in	<i>ndiv</i>	分割数
out	<i>map</i>	head map

cio\_Process.C の 544 行で定義されています。

```

547 {
548     map.clear();
549     map.clear();
550
551     for (int i=0; i<ndiv; i++)
552     {
553         map.insert(headT::value_type(head[i], i));
554     }
555 }

```

#### 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、読み込み、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	<i>dfi_domain</i>	DFI の domain 情報
out	<i>mapHeadX</i>	headX をキーにした位置情報マップ
out	<i>mapHeadY</i>	headX をキーにした位置情報マップ
out	<i>mapHeadZ</i>	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     vector<cio_ActiveSubDomain> subDomainInfo;
251
252     CIO::E_CIO_ERRORCODE ret;
253
254     ret = CreateSubDomainInfo(dfi_domain, subDomainInfo);
255     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(headZ,div[0],mapHeadZ);
348
349 for( int k=0;k<div[2];k++ ){
350 for( int j=0;j<div[1];j++ ){
351 for( int i=0;i<div[0];i++ ){
352     int rankNo = m_rankMap[_CIO_IDX_IJK(i,j,k,div[0],div[1],div[2],0)];
353     if( rankNo < 0 ) continue;
354     rank.RankID = rankNo;
355     rank.VoxelSize[0]=(headX[i]+nvX[i]-1)-headX[i]+1;
356     rank.VoxelSize[1]=(headY[j]+nvY[j]-1)-headY[j]+1;
357     rank.VoxelSize[2]=(headZ[k]+nvZ[k]-1)-headZ[k]+1;
358     rank.HeadIndex[0]=headX[i];
359     rank.HeadIndex[1]=headY[j];
360     rank.HeadIndex[2]=headZ[k];
361     rank.TailIndex[0]=headX[i]+nvX[i]-1;
362     rank.TailIndex[1]=headY[j]+nvY[j]-1;
363     rank.TailIndex[2]=headZ[k]+nvZ[k]-1;
364     RankList.push_back(rank);
365 }}}
366
367 delete [] nvX;
368 delete [] nvY;
369 delete [] nvZ;
370 delete [] headX;
371 delete [] headY;
372 delete [] headZ;
373
374 return CIO::E_CIO_SUCCESS;
375 }

```

#### 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 481 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().

```

484 {
485
486     size_t ndiv = size_t(div[0]) * size_t(div[1]) * size_t(div[2]);
487     int *rankMap = new int[ndiv];
488     if( !rankMap ) return NULL;
489
490     for( size_t i=0;i<ndiv;i++ ) rankMap[i] = -1;
491
492     // 活性サブドメイン情報配置位置に 0 をセット
493     for( int i=0; i<subDomainInfo.size(); i++ )
494     {
495         // サブドメイン情報
496         const cio_ActiveSubDomain dom = subDomainInfo[i];
497
498         // 位置を取得
499         const int *pos = dom.GetPos();
500         if( !pos )
501         {
502             delete [] rankMap;
503             return NULL;

```

```

504     }
505
506     //0 をセット
507     rankMap[_CIO_IDX_IJK(pos[0],pos[1],pos[2],div[0],div[1],div[2],0)] = 0;
508 }
509
510 //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
511 int rankCount = 0;
512 for( int k=0;k<div[2];k++ ){
513     for( int j=0;j<div[1];j++ ){
514         for( int i=0;i<div[0];i++ ){
515             if( rankMap[_CIO_IDX_IJK(i,j,k,div[0],div[1],div[2],0)] == 0 )
516             {
517                 rankMap[_CIO_IDX_IJK(i,j,k,div[0],div[1],div[2],0)] = rankCount;
518                 rankCount++;
519             }
520         }
521     }
522     return rankMap;
523 }

```

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 560 行で定義されています。

参照先 \_CIO\_IDX\_IJK, CreateHeadMap(), と RankList.

```

564 {
565
566     int i,j,k;
567
568     std::set<int>headx,heady,headz;
569     for(int i=0; i<RankList.size(); i++ ) {
570         headx.insert(RankList[i].HeadIndex[0]);
571         heady.insert(RankList[i].HeadIndex[1]);
572         headz.insert(RankList[i].HeadIndex[2]);
573     }
574     CreateHeadMap(headx,mapHeadX);
575     CreateHeadMap(heady,mapHeadY);
576     CreateHeadMap(headz,mapHeadZ);
577
578     size_t ndiv = div[0]*div[1]*div[2];
579
580     int *rankMap = new int[ndiv];
581     for(int i=0; i<ndiv; i++) rankMap[i]=-1;
582
583     headT::iterator it;
584
585     for(int n=0; n<RankList.size(); n++)
586     {
587         it=mapHeadX.find(RankList[n].HeadIndex[0]);
588         i=it->second;
589
590         it=mapHeadY.find(RankList[n].HeadIndex[1]);
591         j=it->second;
592
593         it=mapHeadZ.find(RankList[n].HeadIndex[2]);
594         k=it->second;
595
596         int rnkPos=_CIO_IDX_IJK(i,j,k,div[0],div[1],div[2],0);
597

```

```

598     rankMap[_CIO_IDX_IJK(i,j,k,div[0],div[1],div[2],0)] = n;
599 }
600
601 return rankMap;
602
603 }

```

#### 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	<i>dfi_domain</i>	DFI の domain 情報
out	<i>subDomainInfo</i>	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	<i>ident</i>	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 456 行で定義されています。

参照元 ReadActiveSubdomainFile().

```

457 {
458     char magick_c[] = "SBDM";
459     int magick_i=0;
460
461     //cheak match

```

```

462  magick_i = (magick_c[3]<<24) + (magick_c[2]<<16) + (magick_c[1]<<8) + magick_c[0];
463  if( magick_i == ident )
464  {
465      return 1;
466  }
467
468  //chack unmatched
469  magick_i = (magick_c[0]<<24) + (magick_c[1]<<16) + (magick_c[2]<<8) + magick_c[3];
470  if( magick_i == ident )
471  {
472      return 0;
473  }
474
475  //unknown format
476  return -1;
477 }

```

#### 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 の読み込み

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
190         iret = rank.Read(tpCntl, label_leaf);
191         if( iret == CIO::E_CIO_SUCCESS ) {
192             RankList.push_back(rank);
193         } else return iret;
194     }
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 380 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().

```

384 {
385     if( subDomainFile.empty() ) return CIO::E_CIO_ERROR_OPEN_SBDM;
386
387     // ファイルオープン
388     FILE*fp = fopen( subDomainFile.c_str(), "rb" );
389     if( !fp ) return CIO::E_CIO_ERROR_OPEN_SBDM;
390
391     //エンディアン識別子
392     int ident;
393     if( fread( &ident, sizeof(int), 1, fp ) != 1 )
394     {
395         fclose(fp);
396         return CIO::E_CIO_ERROR_READ_SBDM_HEADER;
397     }
398
399     //エンディアンチェック
400     if( isMatchEndianSbdmMagick( ident ) < 0 )
401     {
402         fclose(fp);
403         return CIO::E_CIO_ERROR_READ_SBDM_FORMAT;
404     }
405
406     // 領域分割数
407     if( fread( div, sizeof(int), 3, fp ) != 3 )
408     {
409         fclose(fp);
410         return CIO::E_CIO_ERROR_READ_SBDM_DIV;
411     }
412
413     if( isMatchEndianSbdmMagick( ident ) == 0 ) BSWAPVEC(div,3);
414
415     // contents
416     size_t nc = size_t(div[0]) * size_t(div[1]) * size_t(div[2]);
417     unsigned char *contents = new unsigned char[nc];
418     if( fread( contents, sizeof(unsigned char), nc, fp ) != nc )
419     {
420         delete [] contents;
421         fclose(fp);
422         return CIO::E_CIO_ERROR_READ_SBDM_CONTENTS;
423     }
424
425     // ファイルクローズ
426     fclose(fp);
427
428     size_t ptr = 0;
429     // 活性ドメイン情報の生成
430     for( int k=0;k<div[2];k++ ){
431         for( int j=0;j<div[1];j++ ){
432             for( int i=0;i<div[0];i++ ){
433                 if( contents[ptr] == 0x01 )
434                 {
435                     int pos[3] = {i,j,k};
436                     cio_ActiveSubDomain dom( pos );
437                     subDomainInfo.push_back( dom );

```

```

438     }
439     ptr++;
440 }}}}
441
442 // contents の delete
443 delete [] contents;
444
445 // 活性ドメインの数をチェック
446 if( subDomainInfo.size() == 0 )
447 {
448     return CIO::E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO;
449 }
450
451 return CIO::E_CIO_SUCCESS;
452 }

```

#### 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 682 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().

```

684 {
685
686     fprintf(fp, "Process {\n");
687     fprintf(fp, "\n");
688
689     cio_Rank rank;
690
691     for(int i=0; i<RankList.size(); i++) {
692         _CIO_WRITE_TAB(fp, tab+1);
693         fprintf(fp, "Rank[%d] {\n", i);
694         fprintf(fp, "\n");
695
696         rank = RankList[i];
697
698         //Rank 要素の出力
699         if( rank.Write(fp,tab+2) != CIO::E_CIO_SUCCESS ) return CIO::E_CIO_ERROR;
700
701         fprintf(fp, "\n");
702         _CIO_WRITE_TAB(fp, tab+1);
703         fprintf(fp, "}\n");
704     }
705
706     fprintf(fp, "\n");
707     fprintf(fp, "}\n");
708
709     return CIO::E_CIO_SUCCESS;
710 }

```

## 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&lt;cio\_Rank&gt; 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	<i>tpCntl</i>	cio_TextParser クラス
in	<i>label_leaf</i>	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* 要素を読み込む (*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 要素を読み込む (inde.dfi)

#### 引数

in	<i>tpCntl</i>	cio_TextParser クラス
in	<i>label_leaf</i>	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( strcmp(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	<i>fp</i>	ファイルポインタ
in	<i>tab</i>	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[%d] {\n", j);
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(), と cio\_Process::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(), と cio\_Process::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(), と cio\_Process::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     // Node があるかチェック
175     ierror = tp->getAllLabels(labels);
176
177     if (ierror != 0)
178     {
179         cout << "ERROR in TextParser::getAllLabels file: " << " ERROR CODE "<< ierror << endl;
180         return false;
181     }
182
183     for (int i = 0; i < labels.size(); i++) {
184         node = labels[i].substr(0, len);
185
186         if ( !strcasecmp(node.c_str(), label.c_str()) )
187         {
188             str = labels[i].substr(len+1);
189             next = str.find("/");
190
191             if ( next == 0 )
192             {
193                 inode++;
194             }
195             else
196             {
197                 if ( chkstr != str.substr(0, next) )
198                 {
199                     chkstr = str.substr(0, next);
200                     inode++;
201                 }
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	<i>label</i>	取得する変数のラベル (絶対パス)
out	<i>ct</i>	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	<i>label</i>	取得するベクトルのラベル (絶対パス)
out	<i>vec</i>	ベクトル格納配列ポインタ
in	<i>nvec</i>	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	<i>label</i>	取得するベクトルのラベル（絶対パス）
out	<i>vec</i>	ベクトル格納配列ポインタ
in	<i>nvec</i>	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	<i>label</i>	取得するベクトルのラベル（絶対パス）
out	<i>vec</i>	ベクトル格納配列ポインタ
in	<i>nvec</i>	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	<i>filename</i>	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 readT-Pfile().

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* 要素を読み込む (*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	<i>step</i>	ステップ番号
in	<i>time</i>	時刻
in	<i>minmax</i>	minmax
in	<i>Ncomp</i>	コンポーネント数
in	<i>avr_mode</i>	Average があるかないかのフラグ
in	<i>step_avr</i>	Average step
in	<i>time_avr</i>	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	<i>step</i>	取得するステップ
out	<i>vec_min</i>	取得した min の合成値
out	<i>vec_max</i>	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 要素を読み込む (inde.dfi)

## 引数

in	<i>tpCntl</i>	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     std::string str;
230     std::string label_base,label_leaf;
231
232     cio_Slice slice;
233
234     int nnode=0;
235
236     CIO::E_CIO_ERRORCODE iret;
237
238     //TimeSlice
239     nnode=0;
240     label_base = "/TimeSlice";
241     if ( tpCntl.chkNode(label_base) ) //があれば
242     {
243         nnode = tpCntl.countLabels(label_base);
244     }
245
246     for (int i=0; i<nnode; i++) {
247         int ncnt=0;
248
249
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( strcmp(str.substr(0,5).c_str(), "Slice") ) continue;
257     label_leaf=label_base+"/"+str;
258
259     //Slice 要素の読み込み
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[%i] {\n", i);
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().

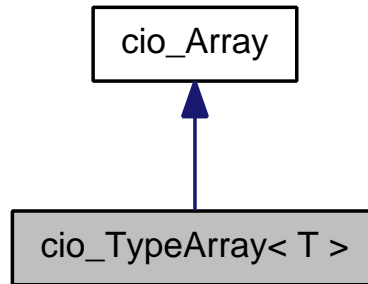
このクラスの説明は次のファイルから生成されました:

- [cio\\_TimeSlice.h](#)
- [cio\\_TimeSlice.C](#)

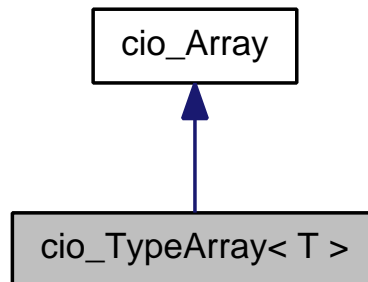
## 6.16 クラス テンプレート `cio_TypeArray< T >`

```
#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ックスを (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&lt;class T &gt; cio\_TypeArray&lt; T &gt;::\_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&lt;class T &gt; cio\_TypeArray&lt; T &gt;::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     cio_TypeArray<T> *dst = dynamic_cast<cio_TypeArray<T>*>(dstptr);
397     if( !dst )
398     {
399         return 1;
400     }
401
402     // データタイプのチェック
403     if( src->getDataType() != dst->getDataType() )
404     {
405         return 2;
406     }
407     CIO::E_CIO_DTYPE dtype = src->getDataType();
408 }
```

```

409 // 配列形状
410 if( src->getArrayShape() != dst->getArrayShape() )
411 {
412     return 3;
413 }
414 CIO::E_CIO_ARRAYSHAPE shape = src->getArrayShape();
415
416 // 成分数
417 if( src->getNcomp() != src->getNcomp() )
418 {
419     return 4;
420 }
421 int ncomp = src->getNcomp();
422
423 // コピーの範囲
424 int gcS = src->getGcInt();
425 const int *headS = src->getHeadIndex();
426 const int *tailS = src->getTailIndex();
427 int gcD = dst->getGcInt();
428 const int *headD = dst->getHeadIndex();
429 const int *tailD = dst->getTailIndex();
430 int sta[3],end[3];
431 for( int i=0;i<3;i++ )
432 {
433     sta[i] = (headS[i]-gcS>=headD[i]-gcD) ? headS[i]-gcS : headD[i]-gcD;
434     end[i] = (tailS[i]+gcS<=tailD[i]+gcD) ? tailS[i]+gcS : tailD[i]+gcD;
435 }
436 for( int i=0;i<3;i++ )
437 {
438     sta[i] = (_sta[i]>=sta[i]) ? _sta[i] : sta[i];
439     end[i] = (_end[i]<=end[i]) ? _end[i] : end[i];
440 }
441
442 // コピー
443 if( m_shape == CIO::E_CIO_IJKN )
444 {
445     for( int n=0;n<ncomp;n++ ){
446         for( int k=sta[2];k<=end[2];k++ ){
447             for( int j=sta[1];j<=end[1];j++ ){
448                 for( int i=sta[0];i<=end[0];i++ ){
449                     dst->hval(i,j,k,n) = src->hval(i,j,k,n);
450                 }
451             }
452         }
453     }
454 }
455 else
456 {
457     for( int k=sta[2];k<=end[2];k++ ){
458         for( int j=sta[1];j<=end[1];j++ ){
459             for( int i=sta[0];i<=end[0];i++ ){
460                 for( int n=0;n<ncomp;n++ ){
461                     dst->hval(n,i,j,k) = src->hval(n,i,j,k);
462                 }
463             }
464         }
465     }
466 }
467 return 0;
468 }

```

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&lt;class T&gt; cio\_TypeArray&lt; T&gt;::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&lt;class T&gt; cio\_TypeArray&lt; T&gt;::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&lt;class T&gt; size\_t cio\_TypeArray&lt; T&gt;::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 546 行で定義されています。

参照先 BSWAPVEC, DBSWAPVEC, と SBSWAPVEC.

```
547 {
548     if( !fp ) return size_t(0);
549     size_t ndata = getArrayLength();
550     size_t nread = fread(m_data, sizeof(T), ndata, fp);
551     if( !bMatchEndian )
552     {
553         size_t bsz = sizeof(T);
554         if( bsz == 2 )
555         {
556             SBSWAPVEC(m_data, nread);
557         }
558         else if( bsz == 4 )
559         {
560             BSWAPVEC(m_data, nread);
561         }
562         else if( bsz == 8 )
563         {
564             DBSWAPVEC(m_data, nread);
565         }
566     }
567     return nread;
568 }
```

## 6.16.3.9 template&lt;class T&gt; cio\_TypeArray&lt; T&gt;::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 572 行で定義されています。

```
573 {
574     if( !fp ) return size_t(0);
575     return fwrite(m_data, sizeof(T), getArrayLength(), fp);
576 }
```

## 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
- std::string `GetUnit` (const std::string Name, int &ret)  
単位の取り出し ("m","cm",...)
- std::string `GetBaseName` (const std::string Name, int &ret)  
ベース名値の取り出し ("L0","P0",...)
- double `GetBaseValue` (const std::string Name, int &ret)  
ベース値の取り出し
- std::string `GetDiffName` (const std::string Name, int &ret)  
*DiffName* の取り出し
- double `GetDiffValue` (const std::string Name, int &ret)  
*Diff Value* の取り出し
- `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 70 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 129 行で定義されています。

```
130 {
131
132 }
```

#### 6.17.2.2 cio\_Unit::~~cio\_Unit ( )

デストラクタ

cio\_Unit.C の 136 行で定義されています。

参照先 UnitList.

```
137 {
138
139     UnitList.clear();
140
141 }
```

### 6.17.3 関数

#### 6.17.3.1 std::string cio\_Unit::GetBaseName ( const std::string Name, int &ret )

ベース名値の取り出し ("L0","P0",...)

## 引数

in	<i>Name</i>	取り出す単位の種類
out	<i>ret</i>	return code

## 戻り値

## ベース名

cio\_Unit.C の 223 行で定義されています。

参照先 CIO::E\_CIO\_SUCCESS, CIO::E\_CIO\_WARN\_GETUNIT, と UnitList.

```

224 {
225     map<std::string,cio_UnitElem>::iterator it;
226
227     //Name をキーにして cio_UnitElem を検索
228     it=UnitList.find(Name);
229
230     //見つからなかった場合は空白を返す
231     if( it == UnitList.end() ) {
232         ret = CIO::E_CIO_WARN_GETUNIT;
233         return "";
234     }
235
236     //ベース名を返す
237     ret = CIO::E_CIO_SUCCESS;
238     return (*it).second.BaseName;
239 }
240 }
```

### 6.17.3.2 double cio\_Unit::GetBaseValue ( const std::string Name, int & ret )

## ベース値の取り出し

## 引数

in	<i>Name</i>	取り出す単位の種類
out	<i>ret</i>	return code

## 戻り値

## ベース値

cio\_Unit.C の 244 行で定義されています。

参照先 CIO::E\_CIO\_SUCCESS, CIO::E\_CIO\_WARN\_GETUNIT, と UnitList.

```

245 {
246     map<std::string,cio_UnitElem>::iterator it;
247
248     //Name をキーにして cio_UnitElem を検索
249     it=UnitList.find(Name);
250
251     //見つからなかった場合は 0.0 を返す
252     if( it == UnitList.end() ) {
253         ret = CIO::E_CIO_WARN_GETUNIT;
254         return 0.0;
255     }
256
257     //ベース値を返す
258     ret = CIO::E_CIO_SUCCESS;
259     return (*it).second.BaseValue;
260 }
261 }
```

### 6.17.3.3 std::string cio\_Unit::GetDiffName ( const std::string Name, int & ret )

## DiffName の取り出し

## 引数

in	<i>Name</i>	取り出す単位の種類
out	<i>ret</i>	return code

## 戻り値

true : DiffName

cio\_Unit.C の 265 行で定義されています。

参照先 CIO::E\_CIO\_SUCCESS, CIO::E\_CIO\_WARN\_GETUNIT, と UnitList.

```

266 {
267     map<std::string,cio_UnitElem>::iterator it;
268
269     //Name をキーにして cio_UnitElem を検索
270     it=UnitList.find(Name);
271
272     //見つからなかった場合は空白を返す
273     if( it == UnitList.end() ) {
274         ret = CIO::E_CIO_WARN_GETUNIT;
275         return "";
276     }
277
278     //DiffName を返す
279     ret = CIO::E_CIO_SUCCESS;
280     return (*it).second.DiffName;
281 }
```

6.17.3.4 double cio\_Unit::GetDiffValue ( const std::string *Name*, int & *ret* )

## Diff Value の取り出し

## 引数

in	<i>Name</i>	取り出す単位の種類
out	<i>ret</i>	return code

## 戻り値

true : Diff Value

cio\_Unit.C の 285 行で定義されています。

参照先 CIO::E\_CIO\_SUCCESS, CIO::E\_CIO\_WARN\_GETUNIT, と UnitList.

```

286 {
287     map<std::string,cio_UnitElem>::iterator it;
288
289     //Name をキーにして cio_UnitElem を検索
290     it=UnitList.find(Name);
291
292     //見つからなかった場合は 0.0 を返す
293     if( it == UnitList.end() ) {
294         ret = CIO::E_CIO_WARN_GETUNIT;
295         return 0.0;
296     }
297
298     //Diff Value を返す
299     ret = CIO::E_CIO_SUCCESS;
300     return (*it).second.DiffValue;
301
302 }
```

6.17.3.5 std::string cio\_Unit::GetUnit ( const std::string *Name*, int & *ret* )

## 単位の取り出し ("m","cm",,,)

## 引数

in	<i>Name</i>	取り出す単位の種類
out	<i>ret</i>	return code

## 戻り値

## 単位

cio\_Unit.C の 202 行で定義されています。

参照先 CIO::E\_CIO\_SUCCESS, CIO::E\_CIO\_WARN\_GETUNIT, と UnitList.

```

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         ret = CIO::E_CIO_WARN_GETUNIT;
212         return "";
213     }
214
215     //単位を返す
216     ret = CIO::E_CIO_SUCCESS;
217     return (*it).second.Unit;
218 }
219 }
```

### 6.17.3.6 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	<i>Name</i>	取り出す単位の種類
out	<i>unit</i>	取得した unit クラス

## 戻り値

error code

cio\_Unit.C の 180 行で定義されています。

参照先 CIO::E\_CIO\_ERROR, CIO::E\_CIO\_SUCCESS, と UnitList.

```

182 {
183     map<std::string,cio_UnitElem>::iterator it;
184
185     //Name をキーにして cio_UnitElem を検索
186     it=UnitList.find(Name);
187
188     //見つからなかった場合は NULL を返す
189     if( it == UnitList.end() ) {
190         return CIO::E_CIO_ERROR;
191     }
192
193     //UnitElem を返す
194     unit = (*it).second;
195
196     return CIO::E_CIO_SUCCESS;
197 }
198 }
```

### 6.17.3.7 CIO::E\_CIO\_ERRORCODE cio\_Unit::Read ( cio\_TextParser *tpCntl* )

read Unit(inde.dfi)

## 引数

in	<i>tpCntl</i>	cio_TextParser クラス
----	---------------	--------------------

## 戻り値

error code

cio\_Unit.C の 146 行で定義されています。

参照先 CIO::E\_CIO\_SUCCESS, cio\_UnitElem::Read(), と UnitList.

参照元 cio\_DFI::ReadInit().

```

147 {
148
149     //Length
150     cio_UnitElem unitLength;
151     if( unitLength.Read(tpCntl, "Length", "L0") == CIO::E_CIO_SUCCESS ) {
152         UnitList.insert( map<std::string, cio_UnitElem>::value_type( "Length", unitLength) );
153     }
154
155     //Velocity
156     cio_UnitElem unitVelocity;
157     if( unitVelocity.Read(tpCntl, "Velocity", "V0") == CIO::E_CIO_SUCCESS ) {
158         UnitList.insert( map<std::string, cio_UnitElem>::value_type( "Velocity", unitVelocity) );
159     }
160
161     //Pressure
162     cio_UnitElem unitPressure;
163     if( unitPressure.Read(tpCntl, "Pressure", "P0", "DiffPrs") == CIO::E_CIO_SUCCESS ) {
164         UnitList.insert( map<std::string, cio_UnitElem>::value_type( "Pressure", unitPressure) );
165     }
166
167     //Temperature
168     cio_UnitElem unitTemperature;
169     if( unitTemperature.Read(tpCntl, "Temperature", "BaseTemp", "DiffTemp") ==
        CIO::E_CIO_SUCCESS ) {
170         UnitList.insert( map<std::string, cio_UnitElem>::value_type( "Temperature", unitTemperature) );
171     }
172
173     return CIO::E_CIO_SUCCESS;
174 }
175 }
```

### 6.17.3.8 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	<i>fp</i>	ファイルポインタ
in	<i>tab</i>	インデント

## 戻り値

error code

cio\_Unit.C の 307 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::WriteIndexDfiFile().

```

309 {
310
311     fprintf(fp, "Unit {\n");
312     fprintf(fp, "\n");
313
314     map<std::string, cio_UnitElem>::iterator it;
315     for( it=UnitList.begin(); it!=UnitList.end(); it++ ) {
```

```

316     if ( (*it).second.Write(fp,tab+1) != CIO::E_CIO_SUCCESS ) return
      CIO::E_CIO_ERROR;
317 }
318
319 fprintf(fp, "\n");
320 fprintf(fp, "}\n");
321 fprintf(fp, "\n");
322
323 return CIO::E_CIO_SUCCESS;
324
325 }

```

## 6.17.4 変数

### 6.17.4.1 map<std::string,cio\_UnitElem> cio\_Unit::UnitList

cio\_Unit.h の 74 行で定義されています。

参照元 cio\_DFI::AddUnit(), GetBaseName(), GetBaseValue(), GetDiffName(), GetDiffValue(), 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 \(const std::string \\_Name, const std::string \\_Unit, const std::string \\_BaseName, const double \\_BaseValue, std::string \\_DiffName="", double \\_DiffValue=0.0\)](#)
- [~cio\\_UnitElem \(\)](#)
- [CIO::E\\_CIO\\_ERRORCODE Read \(cio\\_TextParser tpCntl, const std::string \\_Name, const std::string \\_BaseName, const std::string \\_DiffName=""\)](#)  
Unit 要素の読み込み
- [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,,*)
- std::string [BaseName](#)  
代表値のラベル (*L0,P0,,*)
- double [BaseValue](#)  
代表値
- std::string [DiffName](#)  
差があるときの名前、空の時 *DiffValue* 未定義
- double [DiffValue](#)  
差の値

### 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 BaseName, BaseValue, DiffName, DiffValue, Name, と Unit.

```
25 {
26
27   Name="";
28   Unit="";
29   BaseName="";
30   BaseValue=0.0;
31   DiffName="";
32   DiffValue=0.0;
33
34 }
```

#### 6.18.2.2 cio\_UnitElem::cio\_UnitElem ( const std::string \_Name, const std::string \_Unit, const std::string \_BaseName, const double \_BaseValue, std::string \_DiffName = "", double \_DiffValue = 0.0 )

コンストラクタ

cio\_Unit.C の 38 行で定義されています。

参照先 BaseName, BaseValue, DiffName, DiffValue, Name, と Unit.

```
44 {
45   Name      = _Name;
46   Unit      = _Unit;
47   BaseName  = _BaseName;
48   BaseValue = _BaseValue;
49   DiffName  = _DiffName;
50   DiffValue = _DiffValue;
51 }
```

#### 6.18.2.3 cio\_UnitElem::~cio\_UnitElem ( )

デストラクタ

cio\_Unit.C の 56 行で定義されています。

```
57 {
58
59
60 }
```

### 6.18.3 関数

#### 6.18.3.1 CIO::E\_CIO\_ERRORCODE cio\_UnitElem::Read ( cio\_TextParser tpCntl, const std::string \_Name, const std::string \_BaseName, const std::string \_DiffName = "" )

Unit 要素の読み込み

## 引数

in	<i>tpCntl</i>	cio_TextParser クラス
in	<i>_Name</i>	単位種類 ("Length","Velocity","Pressure",...)
in	<i>_BaseName</i>	名前 ("L0","V0","P0",...)
in	<i>_DiffName</i>	差があるときの差の名前 ("DiffPrs","DiffTemp",...)

## 戻り値

error code

cio\_Unit.C の 65 行で定義されています。

参照先 BaseName, BaseValue, DiffName, DiffValue, CIO::E\_CIO\_SUCCESS, CIO::E\_CIO\_WARN\_GETUNIT, cio\_TextParser::GetValue(), Name, と Unit.

参照元 cio\_Unit::Read().

```

69 {
70
71     std::string str,label;
72     double dt;
73
74     //単位系の読み込み
75     Name = _Name;
76     label="/Unit/"+Name;
77     if ( !(tpCntl.GetValue(label, &str) ) )
78     {
79         return CIO::E_CIO_WARN_GETUNIT;
80     }
81     Unit=str;
82
83     //値の読み込み
84     BaseName = _BaseName;
85     label="/Unit/"+BaseName;
86     if ( !(tpCntl.GetValue(label, &dt) ) )
87     {
88         dt=0.0;
89     }
90     BaseValue=dt;
91
92     //diff の読み込み
93     DiffName = _DiffName;
94     if( !DiffName.empty() ) {
95         label = "/Unit/"+DiffName;
96         if ( !(tpCntl.GetValue(label, &dt) ) )
97         {
98             dt=0.0;
99         }
100         DiffValue=dt;
101     }
102
103     return CIO::E_CIO_SUCCESS;
104
105 }
```

### 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	<i>fp</i>	ファイルポインタ
in	<i>tab</i>	インデント

## 戻り値

error code

cio\_Unit.C の 110 行で定義されています。

参照先 \_CIO\_WRITE\_TAB, BaseName, BaseValue, DiffName, DiffValue, CIO::E\_CIO\_SUCCESS, Name, と Unit.



```

111 {
112
113     _CIO_WRITE_TAB(fp, tab);
114     fprintf(fp, "%-11s = \"%s\"\n", Name.c_str(), Unit.c_str());
115     _CIO_WRITE_TAB(fp, tab);
116     fprintf(fp, "%-11s = %e\n", BaseName.c_str(), BaseValue);
117     if( !DiffName.empty() ) {
118         _CIO_WRITE_TAB(fp, tab);
119         fprintf(fp, "%-11s = %e\n", DiffName.c_str(), DiffValue);
120     }
121
122     return CIO::E_CIO_SUCCESS;
123
124 }

```

## 6.18.4 変数

### 6.18.4.1 std::string cio\_UnitElem::BaseName

代表値のラベル (L0,P0,...)

cio\_Unit.h の 24 行で定義されています。

参照元 cio\_UnitElem(), Read(), と Write().

### 6.18.4.2 double cio\_UnitElem::BaseValue

代表値

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::DiffName

差があるときの名前、空の時DiffValue 未定義

cio\_Unit.h の 26 行で定義されています。

参照元 cio\_UnitElem(), Read(), と Write().

### 6.18.4.4 double cio\_UnitElem::DiffValue

差の値

cio\_Unit.h の 27 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::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(), Read(), と Write().

### 6.18.4.6 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

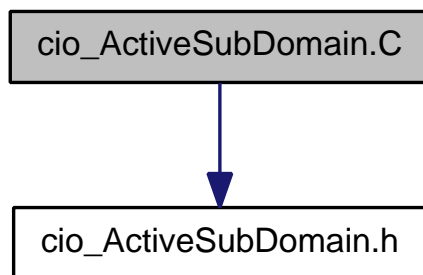
# ファイル

### 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

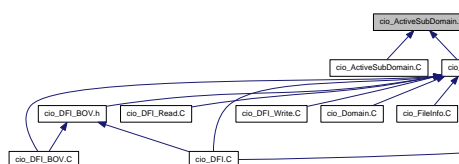
作者

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。



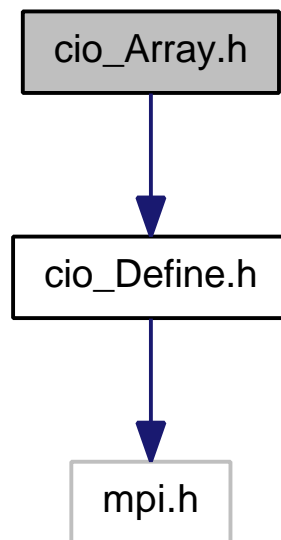
## 構成

- class [cio\\_ActiveSubDomain](#)

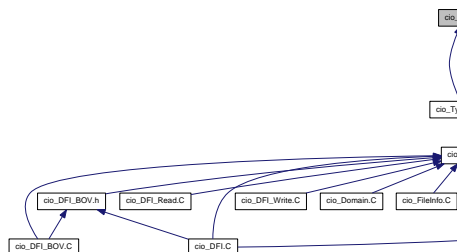
## 7.3 cio\_Array.h

```
#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 )

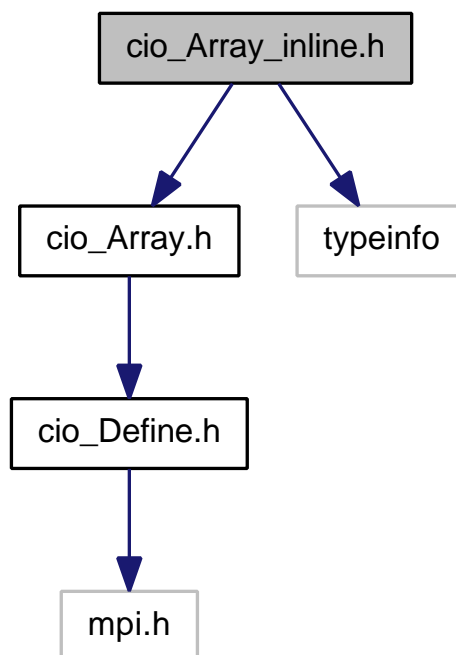
参照元 cio\_Array::interp\_coarse().

## 7.4 cio\_Array\_inline.h

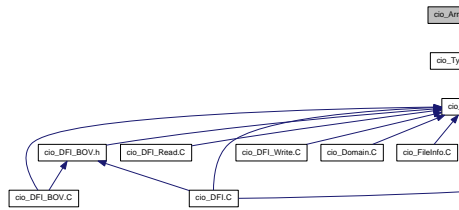
```
#include "cio_Array.h"
```

```
#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`

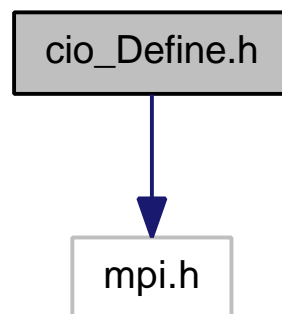
`cio_Array_inline.h` の 17 行で定義されています。

## 7.5 `cio_Define.h`

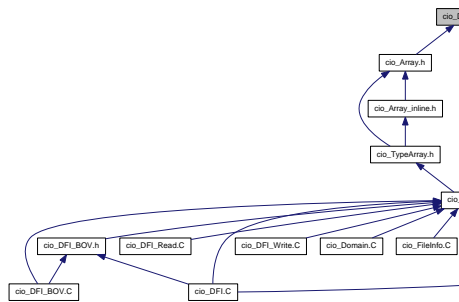
CIO の定義マクロ記述ヘッダーファイル

```
#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\_GUIDECCELL = 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\_OPENERERROR = 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_OPENERERROR = 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_OPENERERROR
= 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	<code>_I</code>	i 方向インデクス
in	<code>_J</code>	j 方向インデクス
in	<code>_NI</code>	i 方向インデクスサイズ
in	<code>_NJ</code>	j 方向インデクスサイズ
in	<code>_VC</code>	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	<code>_I</code>	i 方向インデクス
in	<code>_J</code>	j 方向インデクス
in	<code>_K</code>	k 方向インデクス
in	<code>_NI</code>	i 方向インデクスサイズ
in	<code>_NJ</code>	j 方向インデクスサイズ
in	<code>_NK</code>	k 方向インデクスサイズ
in	<code>_VC</code>	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	<code>_I</code>	i 方向インデクス
in	<code>_J</code>	j 方向インデクス
in	<code>_K</code>	k 方向インデクス
in	<code>_N</code>	成分インデクス
in	<code>_NI</code>	i 方向インデクスサイズ
in	<code>_NJ</code>	j 方向インデクスサイズ
in	<code>_NK</code>	k 方向インデクスサイズ
in	<code>_VC</code>	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	<code>_N</code>	成分インデクス
in	<code>_I</code>	i 方向インデクス
in	<code>_J</code>	j 方向インデクス
in	<code>_NI</code>	i 方向インデクスサイズ
in	<code>_NJ</code>	j 方向インデクスサイズ
in	<code>_NN</code>	成分数
in	<code>_VC</code>	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_IJK( _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) -&gt; 1 次元インデクス変換マクロ

## 引数

in	<code>_N</code>	成分インデクス
in	<code>_I</code>	i 方向インデクス
in	<code>_J</code>	j 方向インデクス
in	<code>_K</code>	k 方向インデクス
in	<code>_NN</code>	成分数
in	<code>_NI</code>	i 方向インデクスサイズ
in	<code>_NJ</code>	j 方向インデクスサイズ
in	<code>_NK</code>	k 方向インデクスサイズ
in	<code>_VC</code>	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	<code>_FP</code>	ファイルポインタ
in	<code>_NTAB</code>	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\_Domain::Write(), cio\_UnitElem::Write(), cio\_FileInfo::Write(), cio\_TimeSlice::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"`

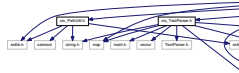
`cio_Define.h` の 35 行で定義されています。

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

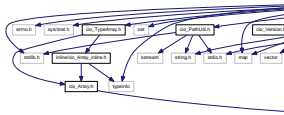
cio\_DFI.C で定義されています。

## 7.7 cio\_DFI.h

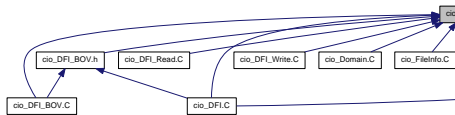
### cio\_DFI Class Header

```
#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。

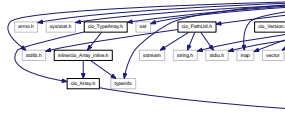


## 7.9 cio\_DFI\_BOV.h

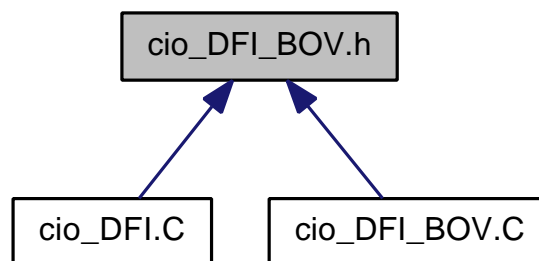
[cio\\_DFI\\_BOV](#) Class Header

```
#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

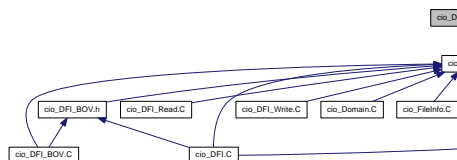
作者

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



作者

kero

[cio\\_DFI\\_SPH.C](#) で定義されています。

## 7.13 cio\_DFI\_SPH.h

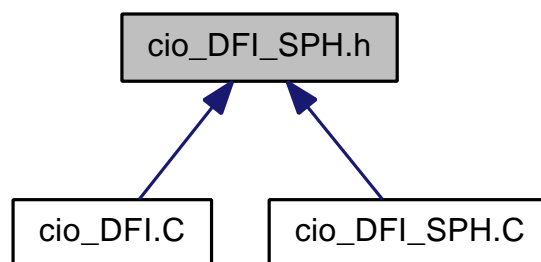
[cio\\_DFI\\_SPH](#) Class Header

```
#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

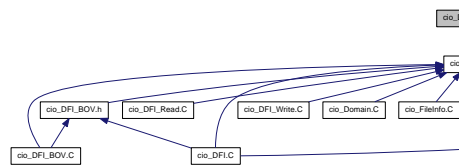
[cio\\_DFI\\_SPH.h](#) で定義されています。

## 7.14 cio\_DFI\_Write.C

[cio\\_DFI](#) Class



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

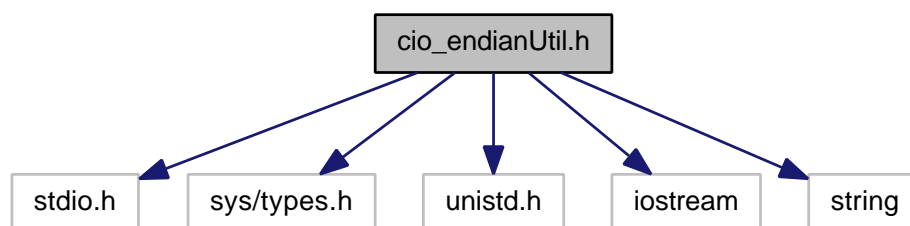
[cio\\_Domain.h](#) で定義されています。

## 7.17 cio\_endianUtil.h

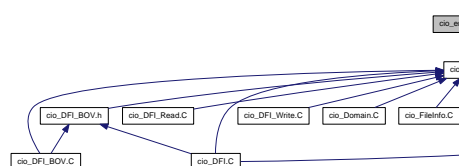
エンディアンユーティリティマクロ・関数ファイル

```
#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。

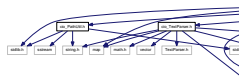
参照元 cio\_TypeArray< T >::readBinary().

## 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

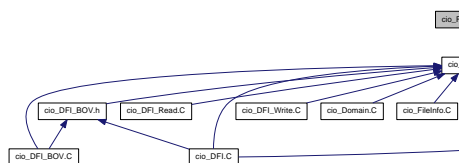
kero

[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

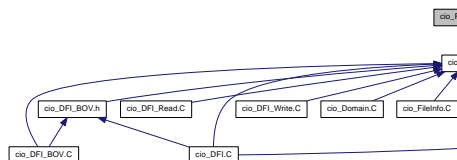
kero

[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

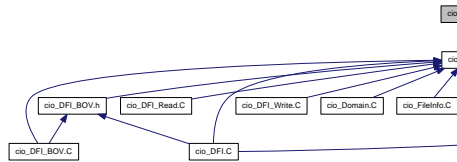
- class [cio\\_FilePath](#)



## 7.26 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

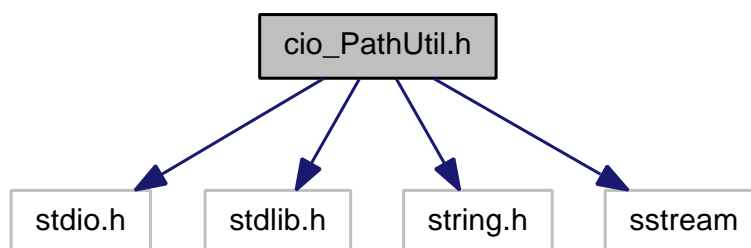
kero

[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。

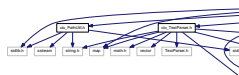
## 7.28 cio\_Process.C

[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

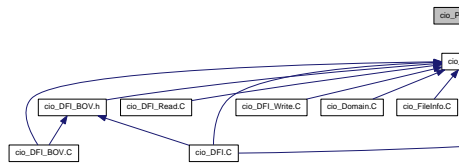
kero

[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

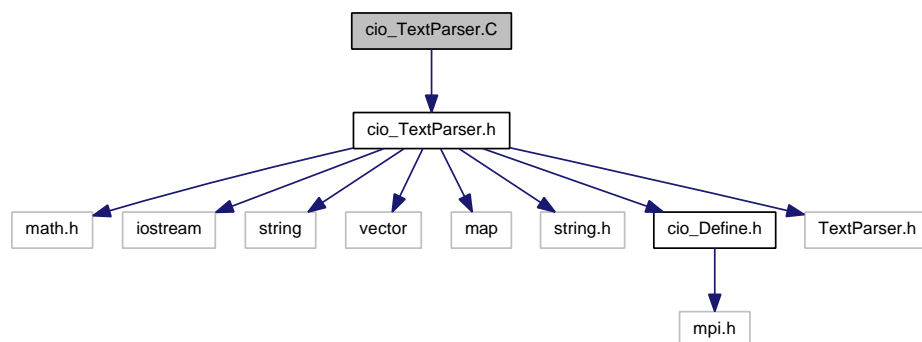
[cio\\_Process.h](#) で定義されています。

## 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

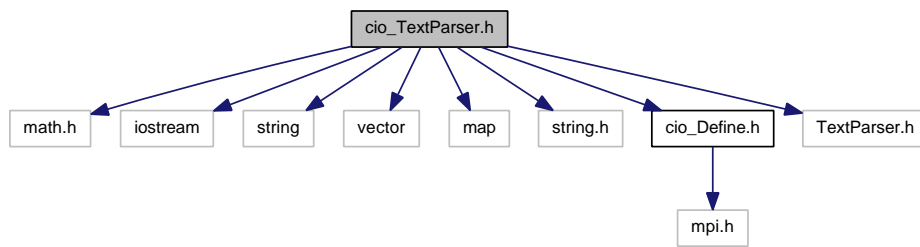
[cio\\_TextParser.C](#) で定義されています。

## 7.31 cio\_TextParser.h

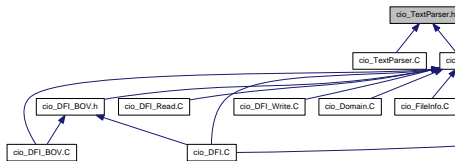
TextParser Control class Header.

```
#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

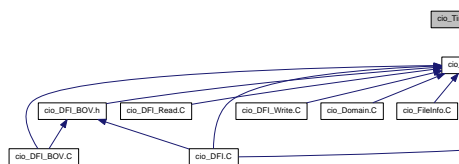
kero

[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

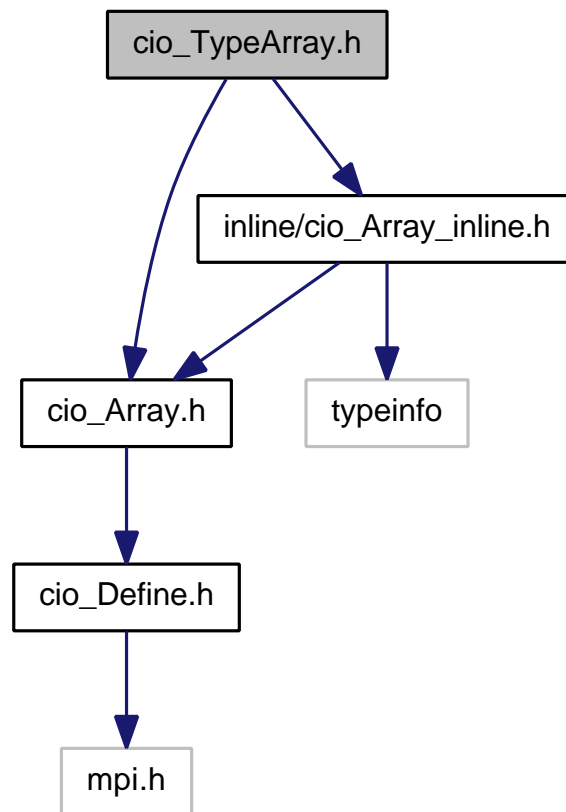
kero

[cio\\_TimeSlice.h](#) で定義されています。

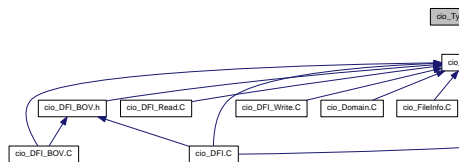
## 7.34 cio\_TypeArray.h

```
#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

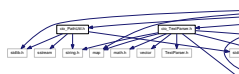
- class `cio_TypeArray< T >`

## 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

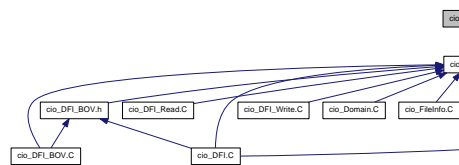
kero

[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](#) "1.3.6"
- #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 "1.3.6"`

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 recvcnt, 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 recvttype, 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, [46](#)
- ~cio\_DFI\_BOV
  - cio\_DFI\_BOV, [91](#)
- ~cio\_DFI\_SPH
  - cio\_DFI\_SPH, [100](#)
- ~cio\_Domain
  - cio\_Domain, [110](#)
- ~cio\_FileInfo
  - cio\_FileInfo, [115](#)
- ~cio\_FilePath
  - cio\_FilePath, [123](#)
- ~cio\_Interval\_Mngr
  - cio\_Interval\_Mngr, [127](#)
- ~cio\_MPI
  - cio\_MPI, [137](#)
- ~cio\_Process
  - cio\_Process, [140](#)
- ~cio\_Rank
  - cio\_Rank, [152](#)
- ~cio\_Slice
  - cio\_Slice, [155](#)
- ~cio\_TextParser
  - cio\_TextParser, [160](#)
- ~cio\_TimeSlice
  - cio\_TimeSlice, [171](#)
- ~cio\_TypeArray
  - cio\_TypeArray, [178](#)
- ~cio\_Unit
  - cio\_Unit, [183](#)
- ~cio\_UnitElem
  - cio\_UnitElem, [189](#)
- \_CIO\_IDX\_IJ
  - cio\_Define.h, [199](#)
- \_CIO\_IDX\_IJK
  - cio\_Define.h, [199](#)
- \_CIO\_IDX\_IJKN
  - cio\_Define.h, [201](#)
- \_CIO\_IDX\_NIJ
  - cio\_Define.h, [201](#)
- \_CIO\_IDX\_NIJK
  - cio\_Define.h, [202](#)
- \_CIO\_TAB\_STR
  - cio\_Define.h, [202](#)
- \_CIO\_WRITE\_TAB
  - cio\_Define.h, [202](#)
- \_DATA\_UNKNOWN
  - cio\_DFI\_SPH, [98](#)
- \_DOUBLE
  - cio\_DFI\_SPH, [98](#)
- \_FLOAT
  - cio\_DFI\_SPH, [98](#)
- \_REAL\_UNKNOWN
  - cio\_DFI\_SPH, [98](#)
- \_SCALAR
  - cio\_DFI\_SPH, [98](#)
- \_VECTOR
  - cio\_DFI\_SPH, [98](#)
- \_getArraySize
  - cio\_Array, [29](#)
- \_getArraySizeInt
  - cio\_Array, [29](#)
- \_val
  - cio\_TypeArray, [178](#)
- ActiveSubdomainFile
  - cio\_Domain, [112](#)
- AddSlice
  - cio\_TimeSlice, [171](#)
- AddUnit
  - cio\_DFI, [47](#)
- ArrayShape
  - cio\_FileInfo, [120](#)
- AveragedStep
  - cio\_Slice, [158](#)
- AveragedTime
  - cio\_Slice, [158](#)
- avr\_mode
  - cio\_Slice, [158](#)
- BSWAP16
  - cio\_endianUtil.h, [212](#)
- BSWAP32
  - cio\_endianUtil.h, [212](#)
- BSWAP64
  - cio\_endianUtil.h, [212](#)
- BSWAP\_X\_16
  - cio\_endianUtil.h, [212](#)
- BSWAP\_X\_32
  - cio\_endianUtil.h, [213](#)
- BSWAP\_X\_64
  - cio\_endianUtil.h, [213](#)
- BSWAPVEC
  - cio\_endianUtil.h, [213](#)
- BaseName

- cio\_UnitElem, 191
- BaseValue
  - cio\_UnitElem, 191
- By\_step
  - cio\_Interval\_Mngr, 127
- By\_time
  - cio\_Interval\_Mngr, 127
- 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\_GUIDECCELL, 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\_PROCFILNAME\_EMP-  
TY, [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, [196](#)
  - cio\_DFI\_inline.h, [208](#)
  - cio\_endianUtil.h, [213](#)
- CIO\_MEMFUN
  - cio\_Array\_inline.h, [196](#)
- CIO\_REVISION
  - cio\_Version.h, [224](#)
- CIO\_VERSION\_NO
  - cio\_Version.h, [224](#)
- calcNextStep
  - cio\_Interval\_Mngr, [127](#)
- calcNextTime
  - cio\_Interval\_Mngr, [127](#)
- CheckReadRank
  - cio\_DFI, [47](#)
  - cio\_Process, [140](#)
- CheckReadType
  - cio\_DFI, [48](#)
- CheckStartEnd
  - cio\_Process, [141](#)
- chkLabel
  - cio\_TextParser, [161](#)
- chkNode
  - cio\_TextParser, [161](#)
- 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, [193](#)
- cio\_ActiveSubDomain.h, [193](#)
- 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](#)
  - getNcomplnt, [34](#)
  - getTailIndex, [34](#)
  - instanceArray, [34–37](#)
  - interp\_coarse, [37](#)
  - m\_Sz, [41](#)
  - m\_Szl, [41](#)
  - m\_dtype, [39](#)
  - m\_gc, [39](#)
  - m\_gcl, [40](#)
  - m\_gcl, [40](#)
  - m\_headIndex, [40](#)
  - m\_ncomp, [40](#)
  - m\_ncompl, [40](#)
  - m\_shape, [40](#)
  - m\_sz, [40](#)
  - m\_szl, [41](#)
  - m\_tailIndex, [41](#)
  - readBinary, [39](#)
  - setHeadIndex, [39](#)
  - writeBinary, [39](#)
- cio\_Array.h, [194](#)

- cio\_interp\_ijkn\_r4\_, 195
- cio\_interp\_ijkn\_r8\_, 195
- cio\_interp\_nijk\_r4\_, 195
- cio\_interp\_nijk\_r8\_, 195
- cio\_Array\_inline.h, 195
  - CIO\_INLINE, 196
  - CIO\_MEMFUN, 196
- cio\_Create\_dfiProcessInfo
  - cio\_DFI, 48
- cio\_DFI, 41
  - ~cio\_DFI, 46
  - AddUnit, 47
  - CheckReadRank, 47
  - CheckReadType, 48
  - cio\_Create\_dfiProcessInfo, 48
  - cio\_DFI, 46
  - cio\_DFI, 46
  - ConvDatatypeE2S, 49
  - ConvDatatypeS2E, 50
  - CreateReadStartEnd, 50
  - DFI\_Domain, 87
  - DFI\_Finfo, 87
  - DFI\_Fpath, 87
  - DFI\_MPI, 87
  - DFI\_Process, 87
  - DFI\_TimeSlice, 88
  - DFI\_Unit, 88
  - Generate\_DFI\_Name, 51
  - Generate\_Directory\_Path, 53
  - Generate\_FieldFileName, 53
  - get\_cio\_Datasize, 55
  - GetArrayShape, 56
  - GetArrayShapeString, 56
  - GetComponentVariable, 58
  - GetDFIGlobalDivision, 59
  - GetDFIGlobalVoxel, 59
  - GetDataType, 59
  - GetDataTypeString, 59
  - getMinMax, 60
  - GetNumComponent, 60
  - getVectorMinMax, 60
  - getVersionInfo, 61
  - GetcioDomain, 56
  - GetcioFileInfo, 57
  - GetcioFilePath, 57
  - GetcioMPI, 57
  - GetcioProcess, 57
  - GetcioTimeSlice, 58
  - GetcioUnit, 58
  - m\_RankID, 88
  - m\_comm, 88
  - m\_directoryPath, 88
  - m\_indexDfiName, 88
  - m\_intervalMngr, 88
  - m\_read\_type, 89
  - m\_readRankList, 89
  - MakeDirectory, 61
  - MakeDirectoryPath, 61
  - MakeDirectorySub, 62
  - normalizeBaseTime, 62
  - normalizeDeltT, 63
  - normalizeIntervalTime, 63
  - normalizeLastTime, 63
  - normalizeStartTime, 63
  - normalizeTime, 64
  - read\_Datarecord, 64
  - read\_HeaderRecord, 65
  - read\_averaged, 64
  - ReadData, 65–67
  - ReadFieldData, 69
  - ReadInit, 71
  - setComponentVariable, 74
  - setIntervalStep, 74
  - setIntervalTime, 75
  - SetTimeSliceFlag, 75
  - write\_DataRecord, 76
  - write\_HeaderRecord, 76
  - write\_averaged, 76
  - WriteData, 77, 79
  - WriteFieldData, 80
  - WriteIndexDfiFile, 81
  - Writelnit, 82, 83
  - WriteProcDfiFile, 85
- cio\_DFI.C, 205
- cio\_DFI.h, 205
- cio\_DFI\_BOV, 89
  - ~cio\_DFI\_BOV, 91
  - cio\_DFI\_BOV, 91
  - cio\_DFI\_BOV, 91
  - read\_Datarecord, 93
  - read\_HeaderRecord, 94
  - read\_averaged, 92
  - write\_DataRecord, 95
  - write\_HeaderRecord, 95
  - write\_averaged, 94
- cio\_DFI\_BOV.C, 206
- cio\_DFI\_BOV.h, 207
- cio\_DFI\_Read.C, 208
- cio\_DFI\_SPH, 96
  - ~cio\_DFI\_SPH, 100
  - \_DATA\_UNKNOWN, 98
  - \_DOUBLE, 98
  - \_FLOAT, 98
  - \_REAL\_UNKNOWN, 98
  - \_SCALAR, 98
  - \_VECTOR, 98
  - cio\_DFI\_SPH, 98
  - cio\_DFI\_SPH, 98
  - DataDims, 98
  - read\_Datarecord, 101
  - read\_HeaderRecord, 102
  - read\_averaged, 100
  - RealType, 98
  - write\_DataRecord, 106
  - write\_HeaderRecord, 106
  - write\_averaged, 105



- cio\_DFI\_SPH.C, 208
- cio\_DFI\_SPH.h, 209
- cio\_DFI\_Write.C, 209
- cio\_DFI\_inline.h, 207
  - CIO\_INLINE, 208
- cio\_Define.h, 196
  - \_CIO\_IDX\_IJ, 199
  - \_CIO\_IDX\_IJK, 199
  - \_CIO\_IDX\_IJKN, 201
  - \_CIO\_IDX\_NIJ, 201
  - \_CIO\_IDX\_NIJK, 202
  - \_CIO\_TAB\_STR, 202
  - \_CIO\_WRITE\_TAB, 202
  - D\_CIO\_BIG, 203
  - D\_CIO\_EXT\_BOV, 203
  - D\_CIO\_EXT\_SPH, 203
  - D\_CIO\_FLOAT32, 203
  - D\_CIO\_FLOAT64, 203
  - D\_CIO\_IJNK, 203
  - D\_CIO\_INT16, 203
  - D\_CIO\_INT32, 203
  - D\_CIO\_INT64, 203
  - D\_CIO\_INT8, 204
  - D\_CIO\_LITTLE, 204
  - D\_CIO\_NIJK, 204
  - D\_CIO\_OFF, 204
  - D\_CIO\_ON, 204
  - D\_CIO\_UINT16, 204
  - D\_CIO\_UINT32, 204
  - D\_CIO\_UINT64, 204
  - D\_CIO\_UINT8, 204
- cio\_Domain, 108
  - ~cio\_Domain, 110
  - ActiveSubdomainFile, 112
  - cio\_Domain, 109
  - cio\_Domain, 109
  - GlobalDivision, 112
  - GlobalOrigin, 112
  - GlobalRegion, 112
  - GlobalVoxel, 113
  - Read, 110
  - Write, 111
- cio\_Domain.C, 210
- cio\_Domain.h, 210
- cio\_FileInfo, 113
  - ~cio\_FileInfo, 115
  - ArrayShape, 120
  - cio\_FileInfo, 114
  - cio\_FileInfo, 114
  - Component, 121
  - ComponentVariable, 121
  - DataType, 121
  - DirectoryPath, 121
  - Endian, 121
  - FileFormat, 121
  - GetComponentVariable, 115
  - GuideCell, 121
  - Prefix, 122
  - Read, 115
  - setComponentVariable, 119
  - TimeSliceDirFlag, 122
  - Write, 119
- cio\_FileInfo.C, 214
- cio\_FileInfo.h, 214
- cio\_FilePath, 122
  - ~cio\_FilePath, 123
  - cio\_FilePath, 123
  - cio\_FilePath, 123
  - ProcDFIFile, 124
  - Read, 123
  - Write, 124
- cio\_FilePath.C, 215
- cio\_FilePath.h, 215
- cio\_Interval\_Mngr, 125
  - ~cio\_Interval\_Mngr, 127
  - By\_step, 127
  - By\_time, 127
  - calcNextStep, 127
  - calcNextTime, 127
  - cio\_Interval\_Mngr, 127
  - cio\_Interval\_Mngr, 127
  - dmod, 128
  - getIntervalStep, 128
  - getIntervalTime, 128
  - getMode, 128
  - getStartStep, 128
  - getStartTime, 129
  - initTrigger, 129
  - isLastStep, 130
  - isLastTime, 130
  - isStarted, 130
  - isTriggered, 130
  - m\_base\_step, 134
  - m\_base\_time, 134
  - m\_dt, 134
  - m\_intvl\_step, 134
  - m\_intvl\_time, 135
  - m\_last\_step, 135
  - m\_last\_time, 135
  - m\_mode, 135
  - m\_start\_step, 135
  - m\_start\_time, 135
  - normalizeBaseTime, 131
  - normalizeDeltT, 132
  - normalizeIntervalTime, 132
  - normalizeLastTime, 132
  - normalizeStartTime, 132
  - normalizeTime, 132
  - noset, 127
  - setInterval, 133
  - setLast, 133
  - setMode, 133
  - setStart, 134
  - type\_IO\_spec, 126
- cio\_Interval\_Mngr.h, 216
- cio\_MPI, 136

- ~cio\_MPI, 137
- cio\_MPI, 136
- cio\_MPI, 136
- NumberOfGroup, 138
- NumberOfRank, 138
- Read, 137
- Write, 137
- cio\_MPI.C, 216
- cio\_MPI.h, 217
- cio\_PathUtil.h, 217
  - MAXPATHLEN, 218
- cio\_Process, 138
  - ~cio\_Process, 140
  - CheckReadRank, 140
  - CheckStartEnd, 141
  - cio\_Process, 140
  - cio\_Process, 140
  - CreateHeadMap, 142, 143
  - CreateRankList, 143, 144
  - CreateRankMap, 145, 146
  - CreateSubDomainInfo, 147
  - headT, 140
  - isMatchEndianSbdmMagick, 147
  - m\_rankMap, 150
  - RankList, 150
  - Read, 148
  - ReadActiveSubdomainFile, 148
  - Write, 150
- cio\_Process.C, 218
- cio\_Process.h, 219
- cio\_Rank, 151
  - ~cio\_Rank, 152
  - cio\_Rank, 151
  - cio\_Rank, 151
  - HeadIndex, 154
  - HostName, 154
  - RankID, 154
  - Read, 152
  - TailIndex, 154
  - VoxelSize, 154
  - Write, 153
- cio\_Slice, 154
  - ~cio\_Slice, 155
  - AveragedStep, 158
  - AveragedTime, 158
  - avr\_mode, 158
  - cio\_Slice, 155
  - cio\_Slice, 155
  - Max, 159
  - Min, 159
  - Read, 156
  - step, 159
  - time, 159
  - VectorMax, 159
  - VectorMin, 159
  - Write, 157
- cio\_TextParser, 160
  - ~cio\_TextParser, 160
  - chkLabel, 161
  - chkNode, 161
  - cio\_TextParser, 160
  - cio\_TextParser, 160
  - countLabels, 162
  - GetNodeStr, 163
  - getTPinstance, 164
  - GetValue, 164, 165
  - GetVector, 167, 168
  - readTPfile, 169
  - remove, 170
  - tp, 170
- cio\_TextParser.C, 219
- cio\_TextParser.h, 220
- cio\_TimeSlice, 170
  - ~cio\_TimeSlice, 171
  - AddSlice, 171
  - cio\_TimeSlice, 171
  - cio\_TimeSlice, 171
  - getMinMax, 172
  - getVectorMinMax, 172
  - Read, 174
  - SliceList, 175
  - Write, 175
- cio\_TimeSlice.C, 220
- cio\_TimeSlice.h, 221
- cio\_TypeArray
  - ~cio\_TypeArray, 178
  - \_val, 178
  - cio\_TypeArray, 177, 178
  - cio\_TypeArray, 177, 178
  - copyArray, 179
  - getData, 180
  - hval, 180, 181
  - m\_data, 182
  - m\_outptr, 182
  - readBinary, 181
  - val, 181
  - writeBinary, 182
- cio\_TypeArray< T >, 176
- cio\_TypeArray.h, 221
- cio\_Unit, 182
  - ~cio\_Unit, 183
  - cio\_Unit, 183
  - cio\_Unit, 183
  - GetBaseName, 183
  - GetBaseValue, 184
  - GetDiffName, 184
  - GetDiffValue, 185
  - GetUnit, 185
  - GetUnitElem, 186
  - Read, 186
  - UnitList, 188
  - Write, 187
- cio\_Unit.C, 222
- cio\_Unit.h, 223
- cio\_UnitElem, 188
  - ~cio\_UnitElem, 189

- BaseName, [191](#)
- BaseValue, [191](#)
- cio\_UnitElem, [189](#)
- cio\_UnitElem, [189](#)
- DiffName, [191](#)
- DiffValue, [191](#)
- Name, [191](#)
- Read, [189](#)
- Unit, [191](#)
- Write, [190](#)
- cio\_Version.h, [223](#)
  - CIO\_REVISION, [224](#)
  - CIO\_VERSION\_NO, [224](#)
- cio\_endianUtil.h, [211](#)
  - BSWAP16, [212](#)
  - BSWAP32, [212](#)
  - BSWAP64, [212](#)
  - BSWAP\_X\_16, [212](#)
  - BSWAP\_X\_32, [213](#)
  - BSWAP\_X\_64, [213](#)
  - BSWAPVEC, [213](#)
  - CIO\_INLINE, [213](#)
  - DBSWAPVEC, [213](#)
  - SBSWAPVEC, [213](#)
- cio\_interp\_ijkn.h, [216](#)
- cio\_interp\_ijkn\_r4\_
  - cio\_Array.h, [195](#)
- cio\_interp\_ijkn\_r8\_
  - cio\_Array.h, [195](#)
- cio\_interp\_nijk.h, [216](#)
- cio\_interp\_nijk\_r4\_
  - cio\_Array.h, [195](#)
- cio\_interp\_nijk\_r8\_
  - cio\_Array.h, [195](#)
- 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, [121](#)
- ComponentVariable
  - cio\_FileInfo, [121](#)
- ConvDatatypeE2S
  - cio\_DFI, [49](#)
- ConvDatatypeS2E
  - cio\_DFI, [50](#)
- copyArray
  - cio\_Array, [29](#)
  - cio\_TypeArray, [179](#)
- countLabels
  - cio\_TextParser, [162](#)
- CreateHeadMap
  - cio\_Process, [142](#), [143](#)
- CreateRankList
  - cio\_Process, [143](#), [144](#)
- CreateRankMap
  - cio\_Process, [145](#), [146](#)
- CreateReadStartEnd
  - cio\_DFI, [50](#)
- CreateSubDomainInfo
  - cio\_Process, [147](#)
- D\_CIO\_BIG
  - cio\_Define.h, [203](#)
- D\_CIO\_EXT\_BOV
  - cio\_Define.h, [203](#)
- D\_CIO\_EXT\_SPH
  - cio\_Define.h, [203](#)
- D\_CIO\_FLOAT32
  - cio\_Define.h, [203](#)
- D\_CIO\_FLOAT64
  - cio\_Define.h, [203](#)
- D\_CIO\_IJNK
  - cio\_Define.h, [203](#)
- D\_CIO\_INT16
  - cio\_Define.h, [203](#)
- D\_CIO\_INT32
  - cio\_Define.h, [203](#)
- D\_CIO\_INT64
  - cio\_Define.h, [203](#)
- D\_CIO\_INT8
  - cio\_Define.h, [204](#)
- D\_CIO\_LITTLE
  - cio\_Define.h, [204](#)
- D\_CIO\_NIJK
  - cio\_Define.h, [204](#)
- D\_CIO\_OFF
  - cio\_Define.h, [204](#)
- D\_CIO\_ON
  - cio\_Define.h, [204](#)
- D\_CIO\_UINT16
  - cio\_Define.h, [204](#)
- D\_CIO\_UINT32
  - cio\_Define.h, [204](#)
- D\_CIO\_UINT64
  - cio\_Define.h, [204](#)
- D\_CIO\_UINT8
  - cio\_Define.h, [204](#)
- DBSWAPVEC
  - cio\_endianUtil.h, [213](#)
- DFI\_Domain
  - cio\_DFI, [87](#)
- DFI\_Finfo
  - cio\_DFI, [87](#)
- DFI\_Fpath

- cio\_DFI, [87](#)
- DFI\_MPI
  - cio\_DFI, [87](#)
- DFI\_Process
  - cio\_DFI, [87](#)
- DFI\_TimeSlice
  - cio\_DFI, [88](#)
- DFI\_Unit
  - cio\_DFI, [88](#)
- DataDims
  - cio\_DFI\_SPH, [98](#)
- DataType
  - cio\_FileInfo, [121](#)
- DiffName
  - cio\_UnitElem, [191](#)
- DiffValue
  - cio\_UnitElem, [191](#)
- DirectoryPath
  - cio\_FileInfo, [121](#)
- dmod
  - cio\_Interval\_Mngr, [128](#)
- 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](#)
- 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\_GUIDECCELL
  - 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\_OPENERERROR  
CIO, 12
- E\_CIO\_ERROR\_READ\_MPI  
CIO, 12
- E\_CIO\_ERROR\_READ\_PROCESS  
CIO, 12
- E\_CIO\_ERROR\_READ\_PROCFILE\_OPENERERROR  
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\_OPENERERROR  
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\_OPENERERROR  
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, [121](#)
- FileFormat
  - cio\_FileInfo, [121](#)
- Generate\_DFI\_Name
  - cio\_DFI, [51](#)
- Generate\_Directory\_Path
  - cio\_DFI, [53](#)
- Generate\_FieldFileName
  - cio\_DFI, [53](#)
- get\_cio\_Datasize
  - cio\_DFI, [55](#)
- getArrayLength
  - cio\_Array, [30](#)
- GetArrayShape
  - cio\_DFI, [56](#)
- getArrayShape
  - cio\_Array, [30](#)
- GetArrayShapeString
  - cio\_DFI, [56](#)
- getArrayShapeString
  - cio\_Array, [30](#)
- getArraySize
  - cio\_Array, [30](#)
- getArraySizeInt
  - cio\_Array, [31](#)
- GetBaseName
  - cio\_Unit, [183](#)
- GetBaseValue
  - cio\_Unit, [184](#)
- GetComponentVariable
  - cio\_DFI, [58](#)
  - cio\_FileInfo, [115](#)
- GetDFIGlobalDivision
  - cio\_DFI, [59](#)
- GetDFIGlobalVoxel
  - cio\_DFI, [59](#)
- getData
  - cio\_Array, [31](#)
  - cio\_TypeArray, [180](#)
- GetDataType
  - cio\_DFI, [59](#)
- getDataType
  - cio\_Array, [32](#)
- GetDataTypeString
  - cio\_DFI, [59](#)
- getDataTypeString
  - cio\_Array, [32](#)
- GetDiffName
  - cio\_Unit, [184](#)
- GetDiffValue
  - cio\_Unit, [185](#)
- getGc
  - cio\_Array, [33](#)
- getGcInt
  - cio\_Array, [33](#)
- getHeadIndex
  - cio\_Array, [33](#)
- getIntervalStep
  - cio\_Interval\_Mngr, [128](#)
- getIntervalTime
  - cio\_Interval\_Mngr, [128](#)
- getMinMax
  - cio\_DFI, [60](#)
  - cio\_TimeSlice, [172](#)
- getMode
  - cio\_Interval\_Mngr, [128](#)
- getNcomp
  - cio\_Array, [33](#)
- getNcomplnt
  - cio\_Array, [34](#)
- GetNodeStr
  - cio\_TextParser, [163](#)
- GetNumComponent
  - cio\_DFI, [60](#)
- GetPos
  - cio\_ActiveSubDomain, [23](#)
- getStartStep
  - cio\_Interval\_Mngr, [128](#)
- getStartTime
  - cio\_Interval\_Mngr, [129](#)
- getTPinstance

- cio\_TextParser, 164
- getTailIndex
  - cio\_Array, 34
- GetUnit
  - cio\_Unit, 185
- GetUnitElem
  - cio\_Unit, 186
- GetValue
  - cio\_TextParser, 164, 165
- GetVector
  - cio\_TextParser, 167, 168
- getVectorMinMax
  - cio\_DFI, 60
  - cio\_TimeSlice, 172
- getVersionInfo
  - cio\_DFI, 61
- GetcioDomain
  - cio\_DFI, 56
- GetcioFileInfo
  - cio\_DFI, 57
- GetcioFilePath
  - cio\_DFI, 57
- GetcioMPI
  - cio\_DFI, 57
- GetcioProcess
  - cio\_DFI, 57
- GetcioTimeSlice
  - cio\_DFI, 58
- GetcioUnit
  - cio\_DFI, 58
- GlobalDivision
  - cio\_Domain, 112
- GlobalOrigin
  - cio\_Domain, 112
- GlobalRegion
  - cio\_Domain, 112
- GlobalVoxel
  - cio\_Domain, 113
- GuideCell
  - cio\_FileInfo, 121
- HeadIndex
  - cio\_Rank, 154
- headT
  - cio\_Process, 140
- HostName
  - cio\_Rank, 154
- hval
  - cio\_TypeArray, 180, 181
- initTrigger
  - cio\_Interval\_Mngr, 129
- instanceArray
  - cio\_Array, 34–37
- interp\_coarse
  - cio\_Array, 37
- isLastStep
  - cio\_Interval\_Mngr, 130
- isLastTime
  - cio\_Interval\_Mngr, 130
- isMatchEndianSbdmMagick
  - cio\_Process, 147
- isStarted
  - cio\_Interval\_Mngr, 130
- isTriggered
  - cio\_Interval\_Mngr, 130
- m\_RankID
  - cio\_DFI, 88
- m\_Sz
  - cio\_Array, 41
- m\_Szl
  - cio\_Array, 41
- m\_base\_step
  - cio\_Interval\_Mngr, 134
- m\_base\_time
  - cio\_Interval\_Mngr, 134
- m\_comm
  - cio\_DFI, 88
- m\_data
  - cio\_TypeArray, 182
- m\_directoryPath
  - cio\_DFI, 88
- m\_dt
  - cio\_Interval\_Mngr, 134
- m\_dtype
  - cio\_Array, 39
- m\_gc
  - cio\_Array, 39
- m\_gcl
  - cio\_Array, 40
- m\_gcl
  - cio\_Array, 40
- m\_headIndex
  - cio\_Array, 40
- m\_indexDfiName
  - cio\_DFI, 88
- m\_intervalMngr
  - cio\_DFI, 88
- m\_intvl\_step
  - cio\_Interval\_Mngr, 134
- m\_intvl\_time
  - cio\_Interval\_Mngr, 135
- m\_last\_step
  - cio\_Interval\_Mngr, 135
- m\_last\_time
  - cio\_Interval\_Mngr, 135
- m\_mode
  - cio\_Interval\_Mngr, 135
- m\_ncomp
  - cio\_Array, 40
- m\_ncompl
  - cio\_Array, 40
- m\_outptr
  - cio\_TypeArray, 182
- m\_pos
  - cio\_ActiveSubDomain, 25
- m\_rankMap

- cio\_Process, [150](#)
- m\_read\_type
  - cio\_DFI, [89](#)
- m\_readRankList
  - cio\_DFI, [89](#)
- m\_shape
  - cio\_Array, [40](#)
- m\_start\_step
  - cio\_Interval\_Mngr, [135](#)
- m\_start\_time
  - cio\_Interval\_Mngr, [135](#)
- m\_sz
  - cio\_Array, [40](#)
- m\_szl
  - cio\_Array, [41](#)
- m\_tailIndex
  - cio\_Array, [41](#)
- MAXPATHLEN
  - cio\_PathUtil.h, [218](#)
- MPI\_Allgather
  - mpi\_stubs.h, [225](#)
- MPI\_CHAR
  - mpi\_stubs.h, [224](#)
- MPI\_COMM\_WORLD
  - mpi\_stubs.h, [224](#)
- MPI\_Comm
  - mpi\_stubs.h, [225](#)
- MPI\_Comm\_rank
  - mpi\_stubs.h, [225](#)
- MPI\_Comm\_size
  - mpi\_stubs.h, [225](#)
- MPI\_Datatype
  - mpi\_stubs.h, [225](#)
- MPI\_Gather
  - mpi\_stubs.h, [226](#)
- MPI\_INT
  - mpi\_stubs.h, [225](#)
- MPI\_Init
  - mpi\_stubs.h, [226](#)
- MPI\_SUCCESS
  - mpi\_stubs.h, [225](#)
- MakeDirectory
  - cio\_DFI, [61](#)
- MakeDirectoryPath
  - cio\_DFI, [61](#)
- MakeDirectorySub
  - cio\_DFI, [62](#)
- Max
  - cio\_Slice, [159](#)
- Min
  - cio\_Slice, [159](#)
- mpi\_stubs.h, [224](#)
  - MPI\_Allgather, [225](#)
  - MPI\_CHAR, [224](#)
  - MPI\_COMM\_WORLD, [224](#)
  - MPI\_Comm, [225](#)
  - MPI\_Comm\_rank, [225](#)
  - MPI\_Comm\_size, [225](#)
  - MPI\_Datatype, [225](#)
  - MPI\_Gather, [226](#)
  - MPI\_INT, [225](#)
  - MPI\_Init, [226](#)
  - MPI\_SUCCESS, [225](#)
- Name
  - cio\_UnitElem, [191](#)
- normalizeBaseTime
  - cio\_DFI, [62](#)
  - cio\_Interval\_Mngr, [131](#)
- normalizeDeltaT
  - cio\_DFI, [63](#)
  - cio\_Interval\_Mngr, [132](#)
- normalizeIntervalTime
  - cio\_DFI, [63](#)
  - cio\_Interval\_Mngr, [132](#)
- normalizeLastTime
  - cio\_DFI, [63](#)
  - cio\_Interval\_Mngr, [132](#)
- normalizeStartTime
  - cio\_DFI, [63](#)
  - cio\_Interval\_Mngr, [132](#)
- normalizeTime
  - cio\_DFI, [64](#)
  - cio\_Interval\_Mngr, [132](#)
- noset
  - cio\_Interval\_Mngr, [127](#)
- NumberOfGroup
  - cio\_MPI, [138](#)
- NumberOfRank
  - cio\_MPI, [138](#)
- operator==
  - cio\_ActiveSubDomain, [24](#)
- Prefix
  - cio\_FileInfo, [122](#)
- ProcDFIFile
  - cio\_FilePath, [124](#)
- RankID
  - cio\_Rank, [154](#)
- RankList
  - cio\_Process, [150](#)
- Read
  - cio\_Domain, [110](#)
  - cio\_FileInfo, [115](#)
  - cio\_FilePath, [123](#)
  - cio\_MPI, [137](#)
  - cio\_Process, [148](#)
  - cio\_Rank, [152](#)
  - cio\_Slice, [156](#)
  - cio\_TimeSlice, [174](#)
  - cio\_Unit, [186](#)
  - cio\_UnitElem, [189](#)
- read\_Datarecord
  - cio\_DFI, [64](#)
  - cio\_DFI\_BOV, [93](#)



- cio\_DFI\_SPH, 101
- read\_HeaderRecord
  - cio\_DFI, 65
  - cio\_DFI\_BOV, 94
  - cio\_DFI\_SPH, 102
- read\_averaged
  - cio\_DFI, 64
  - cio\_DFI\_BOV, 92
  - cio\_DFI\_SPH, 100
- ReadActiveSubdomainFile
  - cio\_Process, 148
- readBinary
  - cio\_Array, 39
  - cio\_TypeArray, 181
- ReadData
  - cio\_DFI, 65–67
- ReadFieldData
  - cio\_DFI, 69
- ReadInit
  - cio\_DFI, 71
- readTPfile
  - cio\_TextParser, 169
- RealType
  - cio\_DFI\_SPH, 98
- remove
  - cio\_TextParser, 170
- SBSWAPVEC
  - cio\_endianUtil.h, 213
- setComponentVariable
  - cio\_DFI, 74
  - cio\_FileInfo, 119
- setHeadIndex
  - cio\_Array, 39
- setInterval
  - cio\_Interval\_Mngr, 133
- setIntervalStep
  - cio\_DFI, 74
- setIntervalTime
  - cio\_DFI, 75
- setLast
  - cio\_Interval\_Mngr, 133
- setMode
  - cio\_Interval\_Mngr, 133
- SetPos
  - cio\_ActiveSubDomain, 24
- setStart
  - cio\_Interval\_Mngr, 134
- SetTimeSliceFlag
  - cio\_DFI, 75
- SliceList
  - cio\_TimeSlice, 175
- step
  - cio\_Slice, 159
- TailIndex
  - cio\_Rank, 154
- time
  - cio\_Slice, 159
- TimeSliceDirFlag
  - cio\_FileInfo, 122
- tp
  - cio\_TextParser, 170
- type\_IO\_spec
  - cio\_Interval\_Mngr, 126
- Unit
  - cio\_UnitElem, 191
- UnitList
  - cio\_Unit, 188
- val
  - cio\_TypeArray, 181
- VectorMax
  - cio\_Slice, 159
- VectorMin
  - cio\_Slice, 159
- vfvPath\_emitDrive
  - CIO, 18
- VoxelSize
  - cio\_Rank, 154
- Write
  - cio\_Domain, 111
  - cio\_FileInfo, 119
  - cio\_FilePath, 124
  - cio\_MPI, 137
  - cio\_Process, 150
  - cio\_Rank, 153
  - cio\_Slice, 157
  - cio\_TimeSlice, 175
  - cio\_Unit, 187
  - cio\_UnitElem, 190
- write\_DataRecord
  - cio\_DFI, 76
  - cio\_DFI\_BOV, 95
  - cio\_DFI\_SPH, 106
- write\_HeaderRecord
  - cio\_DFI, 76
  - cio\_DFI\_BOV, 95
  - cio\_DFI\_SPH, 106
- write\_averaged
  - cio\_DFI, 76
  - cio\_DFI\_BOV, 94
  - cio\_DFI\_SPH, 105
- writeBinary
  - cio\_Array, 39
  - cio\_TypeArray, 182
- WriteData
  - cio\_DFI, 77, 79
- WriteFieldData
  - cio\_DFI, 80
- WriteIndexDfiFile
  - cio\_DFI, 81
- Writelnit
  - cio\_DFI, 82, 83
- WriteProcDfiFile
  - cio\_DFI, 85