# DGOSPREY

Version 0.0 beta

Generated by Doxygen 1.8.3.1

Tue Nov 24 2015 11:01:54

# **Contents**

1	Hiera	archical	Index	1
	1.1	Class I	Hierarchy	1
2	Clas	s Index		3
	2.1	Class I	_ist	3
3	File	Index		6
	3.1	File Lis	st	6
4	Clas	s Docu	mentation	9
	4.1	Adsorb	pentProperties Class Reference	9
		4.1.1	Detailed Description	10
		4.1.2	Constructor & Destructor Documentation	10
		4.1.3	Member Function Documentation	10
		4.1.4	Member Data Documentation	10
	4.2	ARNOI	LDI_DATA Struct Reference	12
		4.2.1	Detailed Description	12
		4.2.2	Member Data Documentation	12
	4.3	BACKT	FRACK_DATA Struct Reference	14
		4.3.1	Detailed Description	14
		4.3.2	Member Data Documentation	14
	4.4	BedHe	atAccumulation Class Reference	15
		4.4.1	Detailed Description	16
		4.4.2	Constructor & Destructor Documentation	16
		4.4.3	Member Function Documentation	16
		4.4.4	Member Data Documentation	16
	4.5	BedPro	operties Class Reference	16
		4.5.1	Detailed Description	17
		4.5.2	Constructor & Destructor Documentation	17
		4.5.3	Member Function Documentation	18
		4.5.4	Member Data Documentation	18
	4.6	BedWa	allHeatTransfer Class Reference	20
		4.6.1	Detailed Description	20
		4.6.2	Constructor & Destructor Documentation	20
		4.6.3	Member Function Documentation	20
		4.6.4	Member Data Documentation	20
	4.7	BiCGS	TAB_DATA Struct Reference	21
		4.7.1	Detailed Description	22
		4.7.2	Member Data Documentation	22

4.8	CGS_E	DATA Struct Reference	25
	4.8.1	Detailed Description	26
	4.8.2	Member Data Documentation	26
4.9	Column	TemperatureIC Class Reference	28
	4.9.1	Detailed Description	29
	4.9.2	Constructor & Destructor Documentation	29
	4.9.3	Member Function Documentation	29
	4.9.4	Member Data Documentation	29
4.10	Concer	ntrationIC Class Reference	29
	4.10.1	Detailed Description	29
	4.10.2	Constructor & Destructor Documentation	29
	4.10.3	Member Function Documentation	29
	4.10.4	Member Data Documentation	30
4.11	DGAdv	rection Class Reference	30
	4.11.1	Detailed Description	31
	4.11.2	Constructor & Destructor Documentation	31
	4.11.3	Member Function Documentation	31
	4.11.4	Member Data Documentation	31
4.12	DGAnis	sotropicDiffusion Class Reference	32
	4.12.1	Detailed Description	33
	4.12.2	Constructor & Destructor Documentation	33
	4.12.3	Member Function Documentation	33
	4.12.4	Member Data Documentation	33
4.13	DGCol	umnHeatAdvection Class Reference	34
	4.13.1	Detailed Description	35
	4.13.2	Constructor & Destructor Documentation	36
	4.13.3	Member Function Documentation	36
	4.13.4	Member Data Documentation	36
4.14	DGCol	umnHeatDispersion Class Reference	37
	4.14.1	Detailed Description	38
	4.14.2	Constructor & Destructor Documentation	38
	4.14.3	Member Function Documentation	38
	4.14.4	Member Data Documentation	39
4.15	DGCol	umnMassAdvection Class Reference	40
	4.15.1	Detailed Description	40
	4.15.2	Constructor & Destructor Documentation	41
	4.15.3	Member Function Documentation	41
	4.15.4	Member Data Documentation	41
4.16	DGCol	umnMassDispersion Class Reference	42
	4.16.1	Detailed Description	43

	4.16.2	Constructor & Destructor Documentation	43
	4.16.3	Member Function Documentation	43
	4.16.4	Member Data Documentation	43
4.17	DGCol	umnWallHeatFluxBC Class Reference	45
	4.17.1	Detailed Description	46
	4.17.2	Constructor & Destructor Documentation	46
	4.17.3	Member Function Documentation	46
	4.17.4	Member Data Documentation	46
4.18	DGCol	umnWallHeatFluxLimitedBC Class Reference	48
	4.18.1	Detailed Description	49
	4.18.2	Constructor & Destructor Documentation	49
	4.18.3	Member Function Documentation	49
	4.18.4	Member Data Documentation	50
4.19	DGCor	ncentrationIC Class Reference	51
	4.19.1	Detailed Description	52
	4.19.2	Constructor & Destructor Documentation	52
	4.19.3	Member Function Documentation	52
	4.19.4	Member Data Documentation	52
4.20	DGFlux	xBC Class Reference	52
	4.20.1	Detailed Description	53
	4.20.2	Constructor & Destructor Documentation	53
	4.20.3	Member Function Documentation	53
	4.20.4	Member Data Documentation	54
4.21	DGFlux	xLimitedBC Class Reference	55
	4.21.1	Detailed Description	56
	4.21.2	Constructor & Destructor Documentation	56
	4.21.3	Member Function Documentation	56
	4.21.4	Member Data Documentation	56
4.22	DGHea	atFluxBC Class Reference	58
	4.22.1	Detailed Description	59
	4.22.2	Constructor & Destructor Documentation	59
	4.22.3	Member Function Documentation	59
	4.22.4	Member Data Documentation	59
4.23	DGHea	atFluxLimitedBC Class Reference	61
	4.23.1	Detailed Description	62
	4.23.2	Constructor & Destructor Documentation	62
	4.23.3	Member Function Documentation	63
	4.23.4	Member Data Documentation	63
4.24	DGMas	ssFluxBC Class Reference	65
	4.24.1	Detailed Description	66

	4.24.2	Constructor & Destructor Documentation	66
	4.24.3	Member Function Documentation	66
	4.24.4	Member Data Documentation	67
4.25	DGMas	ssFluxLimitedBC Class Reference	68
	4.25.1	Detailed Description	70
	4.25.2	Constructor & Destructor Documentation	70
	4.25.3	Member Function Documentation	70
	4.25.4	Member Data Documentation	71
4.26	Dgospr	eyApp Class Reference	73
	4.26.1	Detailed Description	73
	4.26.2	Constructor & Destructor Documentation	73
	4.26.3	Member Function Documentation	73
4.27	Dispers	sion Class Reference	74
	4.27.1	Detailed Description	74
	4.27.2	Constructor & Destructor Documentation	74
	4.27.3	Member Function Documentation	74
	4.27.4	Member Data Documentation	75
4.28	FINCH	_DATA Struct Reference	75
	4.28.1	Detailed Description	79
	4.28.2	Member Data Documentation	79
4.29	FlowPr	operties Class Reference	88
	4.29.1	Detailed Description	90
	4.29.2	Constructor & Destructor Documentation	90
	4.29.3	Member Function Documentation	90
	4.29.4	Member Data Documentation	91
4.30	FluidHe	eatTransfer Class Reference	93
	4.30.1	Detailed Description	93
	4.30.2	Constructor & Destructor Documentation	94
	4.30.3	Member Function Documentation	94
	4.30.4	Member Data Documentation	94
4.31	GAdve	ction Class Reference	94
	4.31.1	Detailed Description	95
	4.31.2	Constructor & Destructor Documentation	95
	4.31.3	Member Function Documentation	95
	4.31.4	Member Data Documentation	95
4.32	GAniso	tropicDiffusion Class Reference	95
	4.32.1	Detailed Description	96
	4.32.2	Constructor & Destructor Documentation	96
	4.32.3	Member Function Documentation	96
	4.32.4	Member Data Documentation	96

4.33	GColumnHeatAdvection Class Reference	97
	I.33.1 Detailed Description	98
	I.33.2 Constructor & Destructor Documentation	98
	I.33.3 Member Function Documentation	98
	4.33.4 Member Data Documentation	98
4.34	GColumnHeatDispersion Class Reference	99
	I.34.1 Detailed Description	100
	I.34.2 Constructor & Destructor Documentation	100
	4.34.3 Member Function Documentation	100
	4.34.4 Member Data Documentation	100
4.35	GColumnMassAdvection Class Reference	101
	4.35.1 Detailed Description	101
	4.35.2 Constructor & Destructor Documentation	101
	4.35.3 Member Function Documentation	101
	4.35.4 Member Data Documentation	102
4.36	GColumnMassDispersion Class Reference	102
	k.36.1 Detailed Description	103
	4.36.2 Constructor & Destructor Documentation	
	4.36.3 Member Function Documentation	103
	1.36.4 Member Data Documentation	103
4.37	GCR_DATA Struct Reference	104
	I.37.1 Detailed Description	
	I.37.2 Member Data Documentation	105
4.38	GMRESLP_DATA Struct Reference	
	I.38.1 Detailed Description	108
	I.38.2 Member Data Documentation	109
4.39	GMRESR_DATA Struct Reference	
	I.39.1 Detailed Description	
	I.39.2 Member Data Documentation	
4.40	GMRESRP_DATA Struct Reference	
	I.40.1 Detailed Description	
	1.40.2 Member Data Documentation	
4.41	GPAST_DATA Struct Reference	
	I.41.1 Detailed Description	117
	I.41.2 Member Data Documentation	118
4.42	GSTA_DATA Struct Reference	119
	I.42.1 Detailed Description	
	1.42.2 Member Data Documentation	
4.43	HeatConductivity Class Reference	
	I.43.1 Detailed Description	120

CONTENTS vi

	4.43.2	Constructor & Destructor Documentation	. 120
	4.43.3	Member Function Documentation	. 120
	4.43.4	Member Data Documentation	. 120
4.44	KMS_D	DATA Struct Reference	. 120
	4.44.1	Detailed Description	. 121
	4.44.2	Member Data Documentation	. 122
4.45	LinearD	OrivingForce Class Reference	. 123
	4.45.1	Detailed Description	. 124
	4.45.2	Constructor & Destructor Documentation	. 124
	4.45.3	Member Function Documentation	. 124
	4.45.4	Member Data Documentation	. 124
4.46	MAGPI	IE_Adsorption Class Reference	. 124
	4.46.1	Detailed Description	. 125
	4.46.2	Constructor & Destructor Documentation	. 125
	4.46.3	Member Function Documentation	. 125
	4.46.4	Member Data Documentation	. 126
4.47	MAGPI	IE_AdsorptionHeat Class Reference	. 126
	4.47.1	Detailed Description	. 127
	4.47.2	Constructor & Destructor Documentation	. 127
	4.47.3	Member Function Documentation	. 127
	4.47.4	Member Data Documentation	. 127
4.48	MAGPI	IE_DATA Struct Reference	. 127
	4.48.1	Detailed Description	. 128
	4.48.2	Member Data Documentation	. 128
4.49	MAGPI	IE_HeatAccumulation Class Reference	. 128
	4.49.1	Detailed Description	. 129
	4.49.2	Constructor & Destructor Documentation	. 129
	4.49.3	Member Function Documentation	. 129
	4.49.4	Member Data Documentation	. 129
4.50	MAGPI	IE_Perturbation Class Reference	. 129
	4.50.1	Detailed Description	. 130
	4.50.2	Constructor & Destructor Documentation	. 130
	4.50.3	Member Function Documentation	. 130
	4.50.4	Member Data Documentation	. 130
4.51	Magpie	eAdsorbateProperties Class Reference	. 131
	4.51.1	Detailed Description	. 132
	4.51.2	Constructor & Destructor Documentation	. 132
	4.51.3	Member Function Documentation	. 132
	4.51.4	Member Data Documentation	. 133
4.52	Matrix<	< T > Class Template Reference	. 134

CONTENTS vii

	4.52.1 Detailed Description	137
	4.52.2 Constructor & Destructor Documentation	137
	4.52.3 Member Function Documentation	138
	4.52.4 Member Data Documentation	145
4.53	MIXED_GAS Struct Reference	146
	4.53.1 Detailed Description	146
	4.53.2 Member Data Documentation	147
4.54	mSPD_DATA Struct Reference	148
	4.54.1 Detailed Description	148
	4.54.2 Member Data Documentation	149
4.55	NUM_JAC_DATA Struct Reference	149
	4.55.1 Detailed Description	149
	4.55.2 Member Data Documentation	150
4.56	OPTRANS_DATA Struct Reference	150
	4.56.1 Detailed Description	150
	4.56.2 Member Data Documentation	150
4.57	PCG_DATA Struct Reference	151
	4.57.1 Detailed Description	152
	4.57.2 Member Data Documentation	152
4.58	PICARD_DATA Struct Reference	154
	4.58.1 Detailed Description	154
	4.58.2 Member Data Documentation	
4.59	PJFNK_DATA Struct Reference	156
	4.59.1 Detailed Description	
	4.59.2 Member Data Documentation	157
4.60	PURE_GAS Struct Reference	161
	4.60.1 Detailed Description	161
	4.60.2 Member Data Documentation	161
4.61	RetardedTimeDerivative Class Reference	162
	4.61.1 Detailed Description	163
	4.61.2 Constructor & Destructor Documentation	163
		163
		163
4.62		163
	2000	165
		165
4.63		169
	4.63.1 Detailed Description	
	4.63.2 Member Data Documentation	
4.64	SKUA_DATA Struct Reference	172

CONTENTS viii

	4.64.1	Detailed Description	. 173
	4.64.2	Member Data Documentation	. 174
4.65	SKUA_	PARAM Struct Reference	. 176
	4.65.1	Detailed Description	. 176
	4.65.2	Member Data Documentation	. 177
4.66	SYSTE	EM_DATA Struct Reference	. 178
	4.66.1	Detailed Description	. 178
	4.66.2	Member Data Documentation	. 179
4.67	TotalCo	olumnPressure Class Reference	. 180
	4.67.1	Detailed Description	. 181
	4.67.2	Constructor & Destructor Documentation	. 181
	4.67.3	Member Function Documentation	. 181
	4.67.4	Member Data Documentation	. 181
4.68	TotalPre	essureIC Class Reference	. 182
	4.68.1	Detailed Description	. 182
	4.68.2	Constructor & Destructor Documentation	. 182
	4.68.3	Member Function Documentation	. 182
	4.68.4	Member Data Documentation	. 182
4.69	WallAm	nbientHeatTransfer Class Reference	. 183
	4.69.1	Detailed Description	. 183
	4.69.2	Constructor & Destructor Documentation	. 183
	4.69.3	Member Function Documentation	. 183
	4.69.4	Member Data Documentation	. 183
4.70		eatAccumulation Class Reference	
	4.70.1	Detailed Description	. 184
	4.70.2	Constructor & Destructor Documentation	. 184
	4.70.3	Member Function Documentation	. 184
	4.70.4	Member Data Documentation	. 184
File	Docume	entation	185
5.1	Adsorbe	pentProperties.h File Reference	. 185
	5.1.1	Detailed Description	
	5.1.2	Function Documentation	
5.2	BedHea	atAccumulation.h File Reference	. 186
	5.2.1	Function Documentation	. 187
5.3	BedPro	pperties.h File Reference	
	5.3.1	Function Documentation	
5.4	BedWa	ıllHeatTransfer.h File Reference	. 187
	5.4.1	Function Documentation	. 187
5.5	Column	nTemperatureIC.h File Reference	

5

	5.5.1	Function Documentation	188
5.6	Concent	trationIC.h File Reference	188
	5.6.1	Function Documentation	188
5.7	DGAdve	ection.h File Reference	188
	5.7.1	Detailed Description	188
	5.7.2	Function Documentation	189
5.8	DGAnis	otropicDiffusion.h File Reference	189
	5.8.1	Detailed Description	189
	5.8.2	Function Documentation	190
5.9	DGColu	mnHeatAdvection.h File Reference	190
	5.9.1	Detailed Description	191
	5.9.2	Macro Definition Documentation	191
	5.9.3	Function Documentation	191
5.10	DGColu	mnHeatDispersion.h File Reference	191
	5.10.1	Detailed Description	192
	5.10.2	Macro Definition Documentation	192
	5.10.3	Function Documentation	192
5.11	DGColu	mnMassAdvection.h File Reference	193
	5.11.1	Detailed Description	193
	5.11.2	Function Documentation	193
5.12	DGColu	mnMassDispersion.h File Reference	194
	5.12.1	Detailed Description	194
	5.12.2	Function Documentation	195
5.13		mnWallHeatFluxBC.h File Reference	
	5.13.1	Detailed Description	195
	5.13.2	Function Documentation	195
5.14		mnWallHeatFluxLimitedBC.h File Reference	
	5.14.1	Detailed Description	196
	5.14.2	Function Documentation	196
5.15	DGCond	centrationIC.h File Reference	196
		Function Documentation	
5.16		BC.h File Reference	197
		Detailed Description	197
		Function Documentation	198
5.17		LimitedBC.h File Reference	198
		Detailed Description	198
		Function Documentation	199
5.18		tFluxBC.h File Reference	
		Detailed Description	
	5.18.2	Function Documentation	200

CONTENTS x

5.19	DGHeatFluxLimitedBC.h File Reference	200
	5.19.1 Detailed Description	200
	5.19.2 Function Documentation	200
5.20	DGMassFluxBC.h File Reference	200
	5.20.1 Detailed Description	201
	5.20.2 Function Documentation	201
5.21	DGMassFluxLimitedBC.h File Reference	201
	5.21.1 Detailed Description	202
	5.21.2 Function Documentation	202
5.22	DgospreyApp.h File Reference	202
	5.22.1 Detailed Description	203
	5.22.2 Function Documentation	203
5.23	DgospreyRevision.h File Reference	203
	5.23.1 Macro Definition Documentation	203
5.24	DgospreyRevision.h File Reference	203
	5.24.1 Macro Definition Documentation	203
5.25	Dispersion.h File Reference	204
	5.25.1 Function Documentation	204
5.26	egret.h File Reference	204
	5.26.1 Detailed Description	205
	5.26.2 Macro Definition Documentation	206
	5.26.3 Function Documentation	207
5.27	error.h File Reference	208
	5.27.1 Detailed Description	208
	5.27.2 Macro Definition Documentation	209
	5.27.3 Enumeration Type Documentation	209
	5.27.4 Function Documentation	210
5.28	finch.h File Reference	211
	5.28.1 Detailed Description	212
	5.28.2 Enumeration Type Documentation	213
	5.28.3 Function Documentation	214
5.29	flock.h File Reference	217
	5.29.1 Detailed Description	217
5.30	FlowProperties.h File Reference	218
	5.30.1 Detailed Description	218
	5.30.2 Macro Definition Documentation	219
	5.30.3 Function Documentation	219
5.31	FluidHeatTransfer.h File Reference	219
	5.31.1 Function Documentation	219
5.32	GAdvection.h File Reference	219

CONTENTS xi

	5.32.1 Function Documentation	219
5.33	GAnisotropicDiffusion.h File Reference	219
	5.33.1 Function Documentation	220
5.34	GColumnHeatAdvection.h File Reference	220
	5.34.1 Function Documentation	220
5.35	GColumnHeatDispersion.h File Reference	220
	5.35.1 Function Documentation	220
5.36	GColumnMassAdvection.h File Reference	220
	5.36.1 Function Documentation	221
5.37	GColumnMassDispersion.h File Reference	221
	5.37.1 Function Documentation	221
5.38	HeatConductivity.h File Reference	221
	5.38.1 Function Documentation	221
5.39	lark.h File Reference	221
	5.39.1 Detailed Description	224
	5.39.2 Macro Definition Documentation	225
	5.39.3 Enumeration Type Documentation	225
	5.39.4 Function Documentation	226
5.40	LinearDrivingForce.h File Reference	236
	5.40.1 Function Documentation	237
5.41	macaw.h File Reference	237
	5.41.1 Detailed Description	237
	5.41.2 Macro Definition Documentation	238
5.42	magpie.h File Reference	238
	5.42.1 Detailed Description	240
	5.42.2 Macro Definition Documentation	240
	5.42.3 Function Documentation	241
5.43	MAGPIE_Adsorption.h File Reference	246
	5.43.1 Detailed Description	246
	5.43.2 Function Documentation	246
5.44	MAGPIE_AdsorptionHeat.h File Reference	247
	5.44.1 Detailed Description	247
	5.44.2 Function Documentation	247
5.45	MAGPIE_HeatAccumulation.h File Reference	247
	5.45.1 Function Documentation	248
5.46	MAGPIE_Perturbation.h File Reference	248
	5.46.1 Detailed Description	248
	5.46.2 Function Documentation	249
5.47	MagpieAdsorbateProperties.h File Reference	249
	5.47.1 Detailed Description	249

1 Hierarchical Index

		5.47.2 Function Documentation	250
	5.48	RetardedTimeDerivative.h File Reference	250
		5.48.1 Function Documentation	250
	5.49	scopsowl.h File Reference	250
		5.49.1 Detailed Description	252
		5.49.2 Macro Definition Documentation	252
		5.49.3 Function Documentation	252
	5.50	skua.h File Reference	257
		5.50.1 Detailed Description	258
		5.50.2 Macro Definition Documentation	259
		5.50.3 Function Documentation	259
	5.51	TotalColumnPressure.h File Reference	262
		5.51.1 Detailed Description	263
		5.51.2 Function Documentation	263
	5.52	TotalPressureIC.h File Reference	263
		5.52.1 Function Documentation	263
	5.53	WallAmbientHeatTransfer.h File Reference	264
		5.53.1 Function Documentation	264
	5.54	WallHeatAccumulation.h File Reference	264
		5.54.1 Function Documentation	264
Ind	dex		264
1	Hie	rarchical Index	
1.1	Cla	ass Hierarchy	
Thi	s inhe	eritance list is sorted roughly, but not completely, alphabetically:	
			10
		OLDI_DATA /ernel	12
	M	IAGPIE_Adsorption	124
	M	IAGPIE_AdsorptionHeat	126
	M	IAGPIE_Perturbation	129
	To	otalColumnPressure	180
	BAC	KTRACK_DATA	14
	BiCG	STAB_DATA	21
	CGS DGK	_DATA ernel	25

DGAdvection	30
DGColumnHeatAdvection	34
DGColumnMassAdvection	40
DGAnisotropicDiffusion	32
DGColumnHeatDispersion	37
DGColumnMassDispersion	42
FINCH_DATA	75
GCR_DATA	104
GMRESLP_DATA	108
GMRESR_DATA	110
GMRESRP_DATA	113
GPAST_DATA	117
GSTA_DATA InitialCondition	119
ColumnTemperatureIC	28
ConcentrationIC	29
DGConcentrationIC	51
<b>TotalPressureIC</b> IntegratedBC	182
DGFluxBC	52
DGColumnWallHeatFluxBC	45
DGHeatFluxBC	58
DGMassFluxBC	65
DGFluxLimitedBC	55
DGColumnWallHeatFluxLimitedBC	48
DGHeatFluxLimitedBC	61
DGMassFluxLimitedBC	68
Kernel	
BedWallHeatTransfer	20
Dispersion	74
FluidHeatTransfer	93
GAdvection	94
GColumnHeatAdvection	97

GColumnMassAdvection	101
GAnisotropicDiffusion	95
GColumnHeatDispersion	99
GColumnMassDispersion	102
HeatConductivity	119
LinearDrivingForce	123
MAGPIE_HeatAccumulation	128
WallAmbientHeatTransfer	183
KMS_DATA	120
MAGPIE_DATA Material	127
AdsorbentProperties	9
BedProperties	16
FlowProperties	88
MagpieAdsorbateProperties	131
Matrix < T >	134
Matrix< double >	134
MIXED_GAS MooseApp	146
DgospreyApp	73
mSPD_DATA	148
NUM_JAC_DATA	149
OPTRANS_DATA	150
PCG_DATA	151
PICARD_DATA	154
PJFNK_DATA	156
PURE_GAS	161
SCOPSOWL_DATA	163
SCOPSOWL_PARAM_DATA	169
SKUA_DATA	172
SKUA_PARAM	176
SYSTEM_DATA TimeDerivative	178

2 Class Index

	BedHeatAccumulation	15
	RetardedTimeDerivative	162
	WallHeatAccumulation	184
2	Class Index	
2.1	Class List	
He	re are the classes, structs, unions and interfaces with brief descriptions:	
	AdsorbentProperties AdsorbentProperties class object inherits from Material object	9
	ARNOLDI_DATA  Data structure for the construction of the Krylov subspaces for a linear system	12
	BACKTRACK_DATA  Data structure for the implementation of Backtracking Linesearch	14
	BedHeatAccumulation	15
	BedProperties BedProperties class object inherits from Material object	16
	BedWallHeatTransfer	20
	BiCGSTAB_DATA  Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems	21
	CGS_DATA  Data structure for the implementation of the CGS algorithm for non-symmetric linear systems	25
	ColumnTemperaturelC	28
	ConcentrationIC	29
	DGAdvection DGAdvection class object inherits from DGKernel object	30
	DGAnisotropicDiffusion DGAnisotropicDiffusion class object inherits from DGKernel object	32
	DGColumnHeatAdvection DGColumnHeatAdvection class object inherits from DGAdvection object	34
	DGColumnHeatDispersion DGColumnHeatDispersion class object inherits from DGAnisotropicDiffusion object	37
	DGColumnMassAdvection DGColumnMassAdvection class object inherits from DGAdvection object	40
	DGColumnMassDispersion DGColumnMassDispersion class object inherits from DGAnisotropicDiffusion object	42
	DGColumnWallHeatFluxBC DGColumnWallHeatFluxBC class object inherits from DGFluxBC object	45

2.1 Class List 5

DGColumnWallHeatFluxLimitedBC DGColumnWallHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object	48
DGConcentrationIC	<b>5</b> 1
DGFluxBC DGFluxBC class object inherits from IntegratedBC object	52
DGFluxLimitedBC DGFluxLimitedBC class object inherits from IntegratedBC object	55
DGHeatFluxBC DGHeatFluxBC class object inherits from DGFluxBC object	58
DGHeatFluxLimitedBC DGHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object	61
DGMassFluxBC DGMassFluxBC class object inherits from DGFluxBC object	6
DGMassFluxLimitedBC DGMassFluxLimitedBC class object inherits from DGFluxLimitedBC object	68
DgospreyApp DgospreyApp inherits from the MooseApp object	73
Dispersion	74
FINCH_DATA Data structure for the FINCH object	75
FlowProperties FlowProperties class object inherits from Material object	88
FluidHeatTransfer	93
GAdvection	94
GAnisotropicDiffusion	95
GColumnHeatAdvection	97
GColumnHeatDispersion	99
GColumnMassAdvection	101
GColumnMassDispersion	102
GCR_DATA  Data structure for the implementation of the GCR algorithm for non-symmetric linear systems	104
GMRESLP_DATA  Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning	108
GMRESR_DATA  Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMR-ESR)	110
GMRESRP_DATA Data structure for the Restarted GMRES algorithm with Right Preconditioning	113

2.1 Class List 6

GPAST_DATA GPAST Data Structure	117
GSTA_DATA GSTA Data Structure	119
HeatConductivity	119
KMS_DATA  Data structure for the implemenation of the Krylov Multi-Space (KMS) Method	120
LinearDrivingForce	123
MAGPIE_Adsorption Magpie Adsorption class inherits from AuxKernel	124
MAGPIE_AdsorptionHeat Magpie Adsorption Heat class inherits from AuxKernel	126
MAGPIE_DATA MAGPIE Data Structure	127
MAGPIE_HeatAccumulation	128
MAGPIE_Perturbation Magpie Perturbation class inherits from AuxKernel	129
MagpieAdsorbateProperties MagpieAdsorbateProperties class object inherits from Material object	131
Matrix < T > Templated C++ Matrix Class Object (click Matrix to go to function definitions)	134
MIXED_GAS  Data structure holding information necessary for computing mixed gas properties	146
mSPD_DATA MSPD Data Structure	148
NUM_JAC_DATA  Data structure to form a numerical jacobian matrix with finite differences	149
OPTRANS_DATA  Data structure for implementation of linear operator transposition	150
PCG_DATA  Data structure for implementation of the PCG algorithms for symmetric linear systems	151
PICARD_DATA  Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems	154
PJFNK_DATA  Data structure for the implementation of the PJFNK algorithm for non-linear systems	156
PURE_GAS  Data structure holding all the parameters for each pure gas spieces	161
RetardedTimeDerivative	162
SCOPSOWL_DATA Primary data structure for SCOPSOWL simulations	163

3 File Index 7

	SCOPSOWL_PARAM_DATA Data structure for the species' parameters in SCOPSOWL	169
	SKUA_DATA  Data structure for all simulation information in SKUA	172
	SKUA_PARAM Data structure for species' parameters in SKUA	176
	SYSTEM_DATA System Data Structure	178
	TotalColumnPressure Total Column Pressure class inherits from AuxKernel	180
	TotalPressureIC	182
	WallAmbientHeatTransfer	183
	WallHeatAccumulation	184
3	File Index	
3.1	I File List	
He	ere is a list of all files with brief descriptions:	
	AdsorbentProperties.h  Material Properties kernel that will setup and hold all information associated with the adsorben	t 185
	BedHeatAccumulation.h	186
	BedProperties.h	187
	BedWallHeatTransfer.h	187
	ColumnTemperaturelC.h	187
	ConcentrationIC.h	188
	DGAdvection.h Discontinous Galerkin kernel for advection	188
	DGAnisotropicDiffusion.h Discontinous Galerkin kernel for anisotropic diffusion	189
	DGColumnHeatAdvection.h Discontinous Galerkin kernel for energy advection in a fixed-bed column	190
	DGColumnHeatDispersion.h Discontinous Galerkin kernel for energy dispersion in a fixed-bed column	191
	DGColumnMassAdvection.h Discontinous Galerkin kernel for mass advection in a fixed-bed column	193
	DGColumnMassDispersion.h Discontinous Galerkin kernel for mass dispersion in a fixed-bed column	194
	DGColumnWallHeatFluxBC.h  Boundary Condition kernel for the heat flux across the wall of the fixed-bed column	195

3.1 File List 8

DGColumnWallHeatFluxLimitedBC.h  Boundary Condition kernel for a dirichlet-like boundary condition of heat on the column wall	196
DGConcentrationIC.h	196
DGFluxBC.h  Boundary Condition kernel for the flux across a boundary of the domain	197
DGFluxLimitedBC.h  Boundary Condition kernel to mimic a Dirichlet BC for DG methods	198
DGHeatFluxBC.h  Boundary Condition kernel for the heat flux in and out of the ends of the fixed-bed column	199
DGHeatFluxLimitedBC.h Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet	200
DGMassFluxBC.h  Boundary Condition kernel for the mass flux in and out of the ends of the fixed-bed column	200
DGMassFluxLimitedBC.h  Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet	201
DgospreyApp.h Registration object for creating a registering DGOSPREY kernels	202
base/DgospreyRevision.h	203
DgospreyRevision.h	203
Dispersion.h	204
egret.h Estimation of Gas-phase pRopErTies	204
error.h All error types are defined here	208
finch.h Flux-limiting Implicit Non-oscillatory Conservative High-resolution scheme	211
flock.h FundamentaL Off-gas Collection of Kernels	217
FlowProperties.h  Material Properties kernel that will setup and calculate gas flow properties based on physical characteristics	218
FluidHeatTransfer.h	219
GAdvection.h	219
GAnisotropicDiffusion.h	219
GColumnHeatAdvection.h	220
GColumnHeatDispersion.h	220
GColumnMassAdvection.h	220
GColumnMassDispersion.h	221

4 Class Documentation

HeatConductivity.h	221
lark.h Linear Algebra Residual Kernels	221
LinearDrivingForce.h	236
macaw.h MAtrix CAlculation Workspace	237
magpie.h Multicomponent Adsorption Generalized Procedure for Isothermal Equilibria	238
MAGPIE_Adsorption.h  Auxillary kernel to calculate adsorption equilibria of a particular gas species in the system	246
MAGPIE_AdsorptionHeat.h  Auxillary kernel to calculate heat of adsorption of a particular gas species in the system	247
MAGPIE_HeatAccumulation.h	247
MAGPIE_Perturbation.h  Auxillary kernel to calculate the perturbed adsorption equilibria of a particular gas species in the system	248
MagpieAdsorbateProperties.h  Material Properties kernel that will setup and hold all information associated with MAGPIE simulations	249
RetardedTimeDerivative.h	250
scopsowl.h Simultaneously Coupled Objects for Pore and Surface diffusion Operations With Linear systems	250
skua.h Surface Kinetics for Uptake by Adsorption	257
TotalColumnPressure.h  Auxillary kernel to calculate total column pressure based on temperature and concentrations	262
TotalPressureIC.h	263
WallAmbientHeatTransfer.h	264
WallHeatAccumulation.h	264

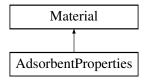
# 4 Class Documentation

# 4.1 AdsorbentProperties Class Reference

 ${\bf Adsorbent Properties} \ class \ object \ inherits \ from \ Material \ object.$ 

#include <AdsorbentProperties.h>

Inheritance diagram for AdsorbentProperties:



## **Public Member Functions**

AdsorbentProperties (const InputParameters &parameters)

Required constructor for objects in MOOSE.

## **Protected Member Functions**

virtual void computeQpProperties ()

Required function override for Material objects in MOOSE.

## **Private Attributes**

· Real binder fraction

Binder fraction in the biporous adsorbent pellet (0 means no binder material)

Real \_binder\_porosity

Macro-porosity of the binder material in the adsorbent pellet.

Real \_crystal\_radius

Nominal radius of the adsorbent crystals suspended in the binder (um)

· Real \_pellet\_diameter

Nominal diameter of the adsorbent pellets in the system (cm)

Real \_macropore\_radius

Nominal size of the macro-pores in the pellet (cm)

· Real \_rhos

Density of the adsorbent pellet (g/cm<sup>\(\circ\)</sup>3)

Real hs

Heat capacity of the adsorbent pellet (J/g/K)

MaterialProperty< Real > & \_pellet\_density

MaterialProperty for the pellet density (g/cm<sup>\(\)</sup>3)

MaterialProperty< Real > & \_pellet\_heat\_capacity

MaterialProperty for the pellet heat capacity (J/g/K)

VariableValue & \_temperature

Reference to the coupled column temperature.

std::vector< unsigned int > \_index

List of indices for the coupled gases.

std::vector< VariableValue \* > \_gas\_conc

Pointer list for the coupled gases.

# 4.1.1 Detailed Description

# AdsorbentProperties class object inherits from Material object.

This class object inherits from the Material object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will set up the structural information about adsorbent in the system that will be used when determining flow properties, as well as some kinetic properties for adsorption dynamics.

Definition at line 57 of file AdsorbentProperties.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AdsorbentProperties::AdsorbentProperties ( const InputParameters & parameters )

Required constructor for objects in MOOSE.

4.1.3 Member Function Documentation

4.1.3.1 virtual void AdsorbentProperties::computeQpProperties() [protected], [virtual]

Required function override for Material objects in MOOSE.

This function computes the material properties when they are needed by other MOOSE objects.

4.1.4 Member Data Documentation

**4.1.4.1 Real AdsorbentProperties::\_binder\_fraction** [private]

Binder fraction in the biporous adsorbent pellet (0 means no binder material)

Definition at line 69 of file AdsorbentProperties.h.

**4.1.4.2** Real AdsorbentProperties::\_binder\_porosity [private]

Macro-porosity of the binder material in the adsorbent pellet.

Definition at line 70 of file AdsorbentProperties.h.

**4.1.4.3** Real AdsorbentProperties::\_crystal\_radius [private]

Nominal radius of the adsorbent crystals suspended in the binder (um)

Definition at line 71 of file AdsorbentProperties.h.

**4.1.4.4** std::vector<VariableValue \*> AdsorbentProperties::\_gas\_conc [private]

Pointer list for the coupled gases.

Definition at line 82 of file AdsorbentProperties.h.

**4.1.4.5 Real AdsorbentProperties::\_hs** [private]

Heat capacity of the adsorbent pellet (J/g/K)

Definition at line 75 of file AdsorbentProperties.h.

**4.1.4.6** std::vector<unsigned int> AdsorbentProperties::\_index [private]

List of indices for the coupled gases.

Definition at line 81 of file AdsorbentProperties.h.

**4.1.4.7 Real AdsorbentProperties::\_macropore\_radius** [private]

Nominal size of the macro-pores in the pellet (cm)

Definition at line 73 of file AdsorbentProperties.h.

**4.1.4.8** MaterialProperty<Real>& AdsorbentProperties::\_pellet\_density [private]

MaterialProperty for the pellet density (g/cm<sup>3</sup>)

Definition at line 77 of file AdsorbentProperties.h.

**4.1.4.9** Real AdsorbentProperties::\_pellet\_diameter [private]

Nominal diameter of the adsorbent pellets in the system (cm)

Definition at line 72 of file AdsorbentProperties.h.

4.1.4.10 MaterialProperty<Real>& AdsorbentProperties::\_pellet\_heat\_capacity [private]

MaterialProperty for the pellet heat capacity (J/g/K)

Definition at line 78 of file AdsorbentProperties.h.

**4.1.4.11 Real AdsorbentProperties::\_rhos** [private]

Density of the adsorbent pellet (g/cm<sup>3</sup>)

Definition at line 74 of file AdsorbentProperties.h.

**4.1.4.12** VariableValue& AdsorbentProperties::\_temperature [private]

Reference to the coupled column temperature.

Definition at line 80 of file AdsorbentProperties.h.

The documentation for this class was generated from the following file:

· AdsorbentProperties.h

## 4.2 ARNOLDI\_DATA Struct Reference

Data structure for the construction of the Krylov subspaces for a linear system.

```
#include <lark.h>
```

## **Public Attributes**

int k

Desired size of the Krylov subspace.

• int iter

Actual size of the Krylov subspace.

· double beta

Normalization parameter.

double hp1

Additional row element of H (separate storage for holding)

• bool Output = true

True = print messages to console.

std::vector< Matrix< double >> Vk

(N) x (k) orthonormal vector basis stored as a vector of column matrices

Matrix< double > Hkp1

(k+1) x (k) upper Hessenberg matrix

Matrix< double > yk

(k) x (1) vector search direction

Matrix< double > e1

(k) x (1) orthonormal vector with 1 in first position

Matrix< double > w

(N) x (1) interim result of the matrix\_vector multiplication

Matrix< double > v

(N) x (1) holding cell for the column entries of Vk and other interims

Matrix< double > sum

(N) x (1) running sum of subspace vectors for use in altering w

## 4.2.1 Detailed Description

Data structure for the construction of the Krylov subspaces for a linear system.

C-style object used in conjunction with the Arnoldi algorithm to construct an orthonormal basis and upper Hessenberg representation of a given linear operator. This is used to solve a linear system both iteratively (i.e., in conjunction with GMRESLP) and directly (i.e., in conjunction with FOM). Alternatively, you can just store the factorized components for later use in another routine.

Definition at line 120 of file lark.h.

## 4.2.2 Member Data Documentation

## 4.2.2.1 double ARNOLDI\_DATA::beta

Normalization parameter.

Definition at line 125 of file lark.h.

## 4.2.2.2 Matrix < double > ARNOLDI\_DATA::e1

(k) x (1) orthonormal vector with 1 in first position

Definition at line 133 of file lark.h.

# 4.2.2.3 Matrix < double > ARNOLDI\_DATA::Hkp1

(k+1) x (k) upper Hessenberg matrix

Definition at line 131 of file lark.h.

# 4.2.2.4 double ARNOLDI\_DATA::hp1

Additional row element of H (separate storage for holding)

Definition at line 126 of file lark.h.

# 4.2.2.5 int ARNOLDI\_DATA::iter

Actual size of the Krylov subspace.

Definition at line 123 of file lark.h.

## 4.2.2.6 int ARNOLDI\_DATA::k

Desired size of the Krylov subspace.

Definition at line 122 of file lark.h.

## 4.2.2.7 bool ARNOLDI\_DATA::Output = true

True = print messages to console.

Definition at line 128 of file lark.h.

## 4.2.2.8 Matrix < double > ARNOLDI\_DATA::sum

(N) x (1) running sum of subspace vectors for use in altering w

Definition at line 136 of file lark.h.

```
4.2.2.9 Matrix<double> ARNOLDI_DATA::v
```

(N) x (1) holding cell for the column entries of Vk and other interims

Definition at line 135 of file lark.h.

```
4.2.2.10 std::vector< Matrix<double> > ARNOLDI_DATA::Vk
```

(N) x (k) orthonormal vector basis stored as a vector of column matrices

Definition at line 130 of file lark.h.

```
4.2.2.11 Matrix<double> ARNOLDI_DATA::w
```

(N) x (1) interim result of the matrix\_vector multiplication

Definition at line 134 of file lark.h.

```
4.2.2.12 Matrix<double> ARNOLDI_DATA::yk
```

(k) x (1) vector search direction

Definition at line 132 of file lark.h.

The documentation for this struct was generated from the following file:

· lark.h

## 4.3 BACKTRACK\_DATA Struct Reference

Data structure for the implementation of Backtracking Linesearch.

```
#include <lark.h>
```

## **Public Attributes**

• int fun call = 0

Number of function calls made during line search.

• double alpha = 1e-4

Scaling parameter for determination of search step size.

• double rho = 0.1

Scaling parameter for to change step size by.

• double lambdaMin =DBL EPSILON

Smallest allowable step length.

double normFkp1

New residual norm of the Newton step.

• bool constRho = false

True = use a constant value for rho.

Matrix< double > Fk

Old residual vector of the Newton step.

Matrix< double > xk

Old solution vector of the Newton step.

## 4.3.1 Detailed Description

Data structure for the implementation of Backtracking Linesearch.

C-style object used in conjunction with the Backtracking Linesearch algorithm to smooth out convergence of Netwon based iterative methods for non-linear systems of equations. The actual algorithm has been separated from the interior of the Newton method so that it can be included in any future Newton based iterative methods being developed.

Definition at line 474 of file lark.h.

## 4.3.2 Member Data Documentation

# 4.3.2.1 double BACKTRACK\_DATA::alpha = 1e-4

Scaling parameter for determination of search step size.

Definition at line 477 of file lark.h.

## 4.3.2.2 bool BACKTRACK\_DATA::constRho = false

True = use a constant value for rho.

Definition at line 482 of file lark.h.

## 4.3.2.3 Matrix < double > BACKTRACK\_DATA::Fk

Old residual vector of the Newton step.

Definition at line 484 of file lark.h.

## 4.3.2.4 int BACKTRACK\_DATA::fun\_call = 0

Number of function calls made during line search.

Definition at line 476 of file lark.h.

## 4.3.2.5 double BACKTRACK\_DATA::lambdaMin = DBL EPSILON

Smallest allowable step length.

Definition at line 479 of file lark.h.

## 4.3.2.6 double BACKTRACK\_DATA::normFkp1

New residual norm of the Newton step.

Definition at line 480 of file lark.h.

# 4.3.2.7 double BACKTRACK\_DATA::rho = 0.1

Scaling parameter for to change step size by.

Definition at line 478 of file lark.h.

# 4.3.2.8 Matrix < double > BACKTRACK\_DATA::xk

Old solution vector of the Newton step.

Definition at line 485 of file lark.h.

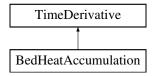
The documentation for this struct was generated from the following file:

· lark.h

# 4.4 BedHeatAccumulation Class Reference

#include <BedHeatAccumulation.h>

Inheritance diagram for BedHeatAccumulation:



# **Public Member Functions**

BedHeatAccumulation (const InputParameters &parameters)

## **Protected Member Functions**

- · virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

#### **Private Attributes**

const MaterialProperty< Real > & \_heat\_retardation

# 4.4.1 Detailed Description

Definition at line 12 of file BedHeatAccumulation.h.

- 4.4.2 Constructor & Destructor Documentation
- 4.4.2.1 BedHeatAccumulation::BedHeatAccumulation ( const InputParameters & parameters )
- 4.4.3 Member Function Documentation
- 4.4.3.1 virtual Real BedHeatAccumulation::computeQpJacobian() [protected], [virtual]
- 4.4.3.2 virtual Real BedHeatAccumulation::computeQpResidual() [protected], [virtual]
- 4.4.4 Member Data Documentation
- **4.4.4.1 const MaterialProperty < Real > & BedHeatAccumulation::\_heat\_retardation** [private]

Definition at line 23 of file BedHeatAccumulation.h.

The documentation for this class was generated from the following file:

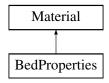
• BedHeatAccumulation.h

# 4.5 BedProperties Class Reference

BedProperties class object inherits from Material object.

```
#include <BedProperties.h>
```

Inheritance diagram for BedProperties:



## **Public Member Functions**

• BedProperties (const InputParameters &parameters)

Required constructor for objects in MOOSE.

# **Protected Member Functions**

• virtual void computeQpProperties ()

Required function override for Material objects in MOOSE.

## **Private Attributes**

· Real \_length

Bed length (cm)

• Real \_din

Column inner diameter (cm)

· Real \_dout

Column outer diameter (cm)

Real eb

Bulk porosity of the bed.

Real \_Kz

Axial thermal conductivity of the bed (J/hr/cm/K)

· Real \_rhow

Density of the column wall (g/cm $^{\wedge}$ 3)

• Real \_hw

Heat capacity of the column wall (J/g/K)

· Real \_Uw

Bed-Wall heat transfer coefficient (J/hr/cm\^2/K)

Real \_Ua

External-Wall heat transfer coefficient (J/hr/cm<sup>\(\)</sup>2/K)

MaterialProperty< Real > & \_inner\_dia

MaterialProperty for column inner diameter.

MaterialProperty< Real > & \_outer\_dia

MaterialProperty for column outer diameter.

• MaterialProperty< Real > & \_porosity

MaterialProperty for bulk porosity of the bed.

MaterialProperty< Real > & \_conductivity

MaterialProperty for thermal conductivity of the bed.

MaterialProperty< Real > & \_wall\_density

MaterialProperty for column wall density.

MaterialProperty< Real > & \_wall\_heat\_capacity

MaterialProperty for column wall heat capacity.

MaterialProperty < Real > & \_bed\_wall\_transfer\_coeff

MaterialProperty for bed-wall heat transfer coefficient.

MaterialProperty < Real > & \_wall\_exterior\_transfer\_coeff

MaterialProperty for exterior-wall heat transfer coefficient.

• VariableValue & \_temperature

Reference to the coupled column temperature.

std::vector< unsigned int > \_index

List of indices for the species in the system.

std::vector< VariableValue \* > \_gas\_conc

Pointer list of the gas species concentrations.

# 4.5.1 Detailed Description

BedProperties class object inherits from Material object.

This class object inherits from the Material object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will set up the parameters of the fixed-bed column. Those parameters include: length, diameter, thermal conductivity, heat transfer coefficients, bulk porosity, etc.

Definition at line 55 of file BedProperties.h.

- 4.5.2 Constructor & Destructor Documentation
- 4.5.2.1 BedProperties::BedProperties ( const InputParameters & parameters )

Required constructor for objects in MOOSE.

- 4.5.3 Member Function Documentation
- 4.5.3.1 virtual void BedProperties::computeQpProperties( ) [protected], [virtual]

Required function override for Material objects in MOOSE.

This function computes the material properties when they are needed by other MOOSE objects.

- 4.5.4 Member Data Documentation
- 4.5.4.1 MaterialProperty < Real > & BedProperties::\_bed\_wall\_transfer\_coeff [private]

MaterialProperty for bed-wall heat transfer coefficient.

Definition at line 83 of file BedProperties.h.

**4.5.4.2** MaterialProperty<Real>& BedProperties::\_conductivity [private]

MaterialProperty for thermal conductivity of the bed.

Definition at line 80 of file BedProperties.h.

**4.5.4.3 Real BedProperties::\_din** [private]

Column inner diameter (cm)

Definition at line 68 of file BedProperties.h.

**4.5.4.4 Real BedProperties::\_dout** [private]

Column outer diameter (cm)

Definition at line 69 of file BedProperties.h.

```
4.5.4.5 Real BedProperties::_eb [private]
Bulk porosity of the bed.
Definition at line 70 of file BedProperties.h.
4.5.4.6 std::vector < Variable Value *> Bed Properties::_gas_conc [private]
Pointer list of the gas species concentrations.
Definition at line 88 of file BedProperties.h.
4.5.4.7 Real BedProperties::_hw [private]
Heat capacity of the column wall (J/g/K)
Definition at line 73 of file BedProperties.h.
4.5.4.8 std::vector<unsigned int> BedProperties::_index [private]
List of indices for the species in the system.
Definition at line 87 of file BedProperties.h.
4.5.4.9 MaterialProperty<Real>& BedProperties::_inner_dia [private]
MaterialProperty for column inner diameter.
Definition at line 77 of file BedProperties.h.
4.5.4.10 Real BedProperties::_Kz [private]
Axial thermal conductivity of the bed (J/hr/cm/K)
Definition at line 71 of file BedProperties.h.
4.5.4.11 Real BedProperties::_length [private]
Bed length (cm)
Definition at line 67 of file BedProperties.h.
4.5.4.12 MaterialProperty<Real>& BedProperties::_outer_dia [private]
MaterialProperty for column outer diameter.
Definition at line 78 of file BedProperties.h.
4.5.4.13 MaterialProperty<Real>& BedProperties::_porosity [private]
MaterialProperty for bulk porosity of the bed.
Definition at line 79 of file BedProperties.h.
4.5.4.14 Real BedProperties::_rhow [private]
Density of the column wall (g/cm<sup>3</sup>)
Definition at line 72 of file BedProperties.h.
4.5.4.15 VariableValue& BedProperties::_temperature [private]
Reference to the coupled column temperature.
```

Definition at line 86 of file BedProperties.h.

**4.5.4.16 Real BedProperties::\_Ua** [private]

External-Wall heat transfer coefficient (J/hr/cm<sup>2</sup>/K)

Definition at line 75 of file BedProperties.h.

**4.5.4.17 Real BedProperties::\_Uw** [private]

Bed-Wall heat transfer coefficient (J/hr/cm<sup>^</sup>2/K)

Definition at line 74 of file BedProperties.h.

**4.5.4.18** MaterialProperty<Real>& BedProperties::\_wall\_density [private]

MaterialProperty for column wall density.

Definition at line 81 of file BedProperties.h.

4.5.4.19 MaterialProperty<Real>& BedProperties::\_wall\_exterior\_transfer\_coeff [private]

MaterialProperty for exterior-wall heat transfer coefficient.

Definition at line 84 of file BedProperties.h.

4.5.4.20 MaterialProperty<Real>& BedProperties::\_wall\_heat\_capacity [private]

MaterialProperty for column wall heat capacity.

Definition at line 82 of file BedProperties.h.

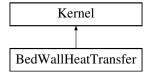
The documentation for this class was generated from the following file:

· BedProperties.h

# 4.6 BedWallHeatTransfer Class Reference

#include <BedWallHeatTransfer.h>

Inheritance diagram for BedWallHeatTransfer:



# **Public Member Functions**

• BedWallHeatTransfer (const InputParameters &parameters)

# **Protected Member Functions**

- virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

# **Private Attributes**

- const MaterialProperty< Real > & \_bed\_wall\_transfer\_coeff
- const MaterialProperty< Real > & \_inner\_dia

- const MaterialProperty< Real > & \_outer\_dia
- VariableValue & \_column\_temp

## 4.6.1 Detailed Description

Definition at line 11 of file BedWallHeatTransfer.h.

- 4.6.2 Constructor & Destructor Documentation
- 4.6.2.1 BedWallHeatTransfer::BedWallHeatTransfer ( const InputParameters & parameters )
- 4.6.3 Member Function Documentation
- 4.6.3.1 virtual Real BedWallHeatTransfer::computeQpJacobian() [protected], [virtual]
- 4.6.3.2 virtual Real BedWallHeatTransfer::computeQpResidual() [protected], [virtual]
- 4.6.4 Member Data Documentation
- **4.6.4.1** const MaterialProperty<Real>& BedWallHeatTransfer::\_bed\_wall\_transfer\_coeff [private]

Definition at line 21 of file BedWallHeatTransfer.h.

**4.6.4.2 VariableValue& BedWallHeatTransfer::\_column\_temp** [private]

Definition at line 25 of file BedWallHeatTransfer.h.

4.6.4.3 const MaterialProperty<Real>& BedWallHeatTransfer::\_inner\_dia [private]

Definition at line 22 of file BedWallHeatTransfer.h.

**4.6.4.4 const MaterialProperty<Real>& BedWallHeatTransfer::\_outer\_dia** [private]

Definition at line 23 of file BedWallHeatTransfer.h.

The documentation for this class was generated from the following file:

· BedWallHeatTransfer.h

# 4.7 BiCGSTAB\_DATA Struct Reference

Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems.

```
#include <lark.h>
```

# **Public Attributes**

• int maxit = 0

Maximum allowable iterations - default = min(2\*vector\_size,1000)

• int iter = 0

Actual number of iterations.

· bool breakdown

Boolean to determine if the method broke down.

double alpha

Step size parameter for next solution.

· double beta

Step size parameter for search direction.

· double rho

Scaling parameter for alpha and beta.

double rho old

Previous scaling parameter for alpha and beta.

· double omega

Scaling parameter and additional step length.

· double omega old

Previous scaling parameter and step length.

• double tol rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

• double tol\_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

· double res

Absolute residual norm.

· double relres

Relative residual norm.

· double relres base

Initial residual norm.

double bestres

Best found residual norm.

• bool Output = true

True = print messages to console.

Matrix< double > x

Current solution to the linear system.

Matrix< double > bestx

Best found solution to the linear system.

• Matrix< double > r

Residual vector for the linear system.

• Matrix< double > r0

Initial residual vector.

Matrix< double > v

Search direction for p.

Matrix< double > p

Search direction for updating.

Matrix< double > y

Preconditioned search direction.

Matrix< double > s

Residual updating vector.

Matrix< double > z

Preconditioned residual updating vector.

• Matrix< double > t

Search direction for resdidual updates.

# 4.7.1 Detailed Description

Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems.

C-style object used in conjunction with the Bi-Conjugate Gradient STABalized (BiCGSTAB) algorithm to solve a linear system of equations. This algorithm is generally more efficient than any GMRES or GCR variant, but may not always reduce the residual at each step. However, if used with preconditioning, then this algorithm is very efficient, especially when used for solving grid-based linear systems.

Definition at line 249 of file lark.h.

## 4.7.2 Member Data Documentation

## 4.7.2.1 double BiCGSTAB\_DATA::alpha

Step size parameter for next solution.

Definition at line 255 of file lark.h.

# 4.7.2.2 double BiCGSTAB\_DATA::bestres

Best found residual norm.

Definition at line 266 of file lark.h.

## 4.7.2.3 Matrix < double > BiCGSTAB\_DATA::bestx

Best found solution to the linear system.

Definition at line 271 of file lark.h.

# 4.7.2.4 double BiCGSTAB\_DATA::beta

Step size parameter for search direction.

Definition at line 256 of file lark.h.

## 4.7.2.5 bool BiCGSTAB\_DATA::breakdown

Boolean to determine if the method broke down.

Definition at line 253 of file lark.h.

## 4.7.2.6 int BiCGSTAB\_DATA::iter = 0

Actual number of iterations.

Definition at line 252 of file lark.h.

# 4.7.2.7 int BiCGSTAB\_DATA::maxit = 0

Maximum allowable iterations - default = min(2\*vector size,1000)

Definition at line 251 of file lark.h.

# 4.7.2.8 double BiCGSTAB\_DATA::omega

Scaling parameter and additional step length.

Definition at line 259 of file lark.h.

# 4.7.2.9 double BiCGSTAB\_DATA::omega\_old

Previous scaling parameter and step length.

Definition at line 260 of file lark.h.

# 4.7.2.10 bool BiCGSTAB\_DATA::Output = true

True = print messages to console.

Definition at line 268 of file lark.h.

# 4.7.2.11 Matrix<double> BiCGSTAB\_DATA::p

Search direction for updating.

Definition at line 275 of file lark.h.

4.7.2.12 Matrix<double> BiCGSTAB\_DATA::r

Residual vector for the linear system.

Definition at line 272 of file lark.h.

4.7.2.13 Matrix<double> BiCGSTAB\_DATA::r0

Initial residual vector.

Definition at line 273 of file lark.h.

4.7.2.14 double BiCGSTAB\_DATA::relres

Relative residual norm.

Definition at line 264 of file lark.h.

4.7.2.15 double BiCGSTAB\_DATA::relres\_base

Initial residual norm.

Definition at line 265 of file lark.h.

4.7.2.16 double BiCGSTAB\_DATA::res

Absolute residual norm.

Definition at line 263 of file lark.h.

4.7.2.17 double BiCGSTAB\_DATA::rho

Scaling parameter for alpha and beta.

Definition at line 257 of file lark.h.

4.7.2.18 double BiCGSTAB\_DATA::rho\_old

Previous scaling parameter for alpha and beta.

Definition at line 258 of file lark.h.

4.7.2.19 Matrix<double> BiCGSTAB\_DATA::s

Residual updating vector.

Definition at line 277 of file lark.h.

4.7.2.20 Matrix<double> BiCGSTAB\_DATA::t

Search direction for resdidual updates.

Definition at line 279 of file lark.h.

4.7.2.21 double BiCGSTAB\_DATA::tol\_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

Definition at line 262 of file lark.h.

4.7.2.22 double BiCGSTAB\_DATA::tol\_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 261 of file lark.h.

4.7.2.23 Matrix<double> BiCGSTAB\_DATA::v

Search direction for p.

Definition at line 274 of file lark.h.

4.7.2.24 Matrix < double > BiCGSTAB\_DATA::x

Current solution to the linear system.

Definition at line 270 of file lark.h.

4.7.2.25 Matrix<double> BiCGSTAB\_DATA::y

Preconditioned search direction.

Definition at line 276 of file lark.h.

4.7.2.26 Matrix < double > BiCGSTAB\_DATA::z

Preconditioned residual updating vector.

Definition at line 278 of file lark.h.

The documentation for this struct was generated from the following file:

· lark.h

#### 4.8 CGS\_DATA Struct Reference

Data structure for the implementation of the CGS algorithm for non-symmetric linear systems.

```
#include <lark.h>
```

#### **Public Attributes**

• int maxit = 0

Maximum allowable iterations - default = min(2\*vector\_size,1000)

• int iter = 0

Actual number of iterations.

bool breakdown

Boolean to determine if the method broke down.

double alpha

Step size parameter for next solution.

· double beta

Step size parameter for search direction.

• double rho

Scaling parameter for alpha and beta.

· double sigma

Scaling parameter and additional step length.

double tol\_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

• double tol\_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

double res

Absolute residual norm.

· double relres

Relative residual norm.

double relres\_base

Initial residual norm.

· double bestres

Best found residual norm.

• bool Output = true

True = print messages to console.

Matrix< double > x

Current solution to the linear system.

Matrix< double > bestx

Best found solution to the linear system.

Matrix< double > r

Residual vector for the linear system.

Matrix< double > r0

Initial residual vector.

Matrix< double > u

Search direction for v.

Matrix< double > w

Updates sigma and u.

Matrix< double > v

Search direction for x.

Matrix< double > p

Preconditioning result for w, z, and matvec for Ax.

• Matrix< double > c

Holds the matvec result between A and p.

Matrix< double > z

Full search direction for x.

## 4.8.1 Detailed Description

Data structure for the implementation of the CGS algorithm for non-symmetric linear systems.

C-style object to be used in conjunction with the Conjugate Gradient Squared (CGS) algorithm to solve linear systems of equations. This algorithm is slightly less computational work than BiCGSTAB, but is much less stable. As a result, I do not recommend using this algorithm unless you also use some form of preconditioning.

Definition at line 288 of file lark.h.

# 4.8.2 Member Data Documentation

4.8.2.1 double CGS\_DATA::alpha

Step size parameter for next solution.

Definition at line 294 of file lark.h.

4.8.2.2 double CGS\_DATA::bestres

Best found residual norm.

Definition at line 303 of file lark.h.

4.8.2.3 Matrix < double > CGS\_DATA::bestx

Best found solution to the linear system.

Definition at line 308 of file lark.h.

4.8.2.4 double CGS\_DATA::beta

Step size parameter for search direction.

Definition at line 295 of file lark.h.

4.8.2.5 bool CGS\_DATA::breakdown

Boolean to determine if the method broke down.

Definition at line 292 of file lark.h.

4.8.2.6 Matrix<double> CGS\_DATA::c

Holds the matvec result between A and p.

Definition at line 315 of file lark.h.

4.8.2.7 int CGS\_DATA::iter = 0

Actual number of iterations.

Definition at line 291 of file lark.h.

4.8.2.8 int CGS\_DATA::maxit = 0

Maximum allowable iterations - default = min(2\*vector\_size,1000)

Definition at line 290 of file lark.h.

4.8.2.9 bool CGS\_DATA::Output = true

True = print messages to console.

Definition at line 305 of file lark.h.

4.8.2.10 Matrix<double> CGS\_DATA::p

Preconditioning result for w, z, and matvec for Ax.

Definition at line 314 of file lark.h.

4.8.2.11 Matrix<double> CGS\_DATA::r

Residual vector for the linear system.

Definition at line 309 of file lark.h.

4.8.2.12 Matrix<double> CGS\_DATA::r0

Initial residual vector.

Definition at line 310 of file lark.h.

4.8.2.13 double CGS\_DATA::relres

Relative residual norm.

Definition at line 301 of file lark.h.

4.8.2.14 double CGS\_DATA::relres\_base

Initial residual norm.

Definition at line 302 of file lark.h.

4.8.2.15 double CGS\_DATA::res

Absolute residual norm.

Definition at line 300 of file lark.h.

4.8.2.16 double CGS\_DATA::rho

Scaling parameter for alpha and beta.

Definition at line 296 of file lark.h.

4.8.2.17 double CGS\_DATA::sigma

Scaling parameter and additional step length.

Definition at line 297 of file lark.h.

4.8.2.18 double CGS\_DATA::tol\_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

Definition at line 299 of file lark.h.

4.8.2.19 double CGS\_DATA::tol\_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 298 of file lark.h.

4.8.2.20 Matrix<double> CGS\_DATA::u

Search direction for v.

Definition at line 311 of file lark.h.

4.8.2.21 Matrix<double> CGS\_DATA::v

Search direction for x.

Definition at line 313 of file lark.h.

4.8.2.22 Matrix<double> CGS\_DATA::w

Updates sigma and u.

Definition at line 312 of file lark.h.

4.8.2.23 Matrix<double> CGS\_DATA::x

Current solution to the linear system.

Definition at line 307 of file lark.h.

4.8.2.24 Matrix<double> CGS\_DATA::z

Full search direction for x.

Definition at line 316 of file lark.h.

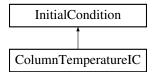
The documentation for this struct was generated from the following file:

· lark.h

# 4.9 ColumnTemperaturelC Class Reference

#include <ColumnTemperatureIC.h>

Inheritance diagram for ColumnTemperatureIC:



## **Public Member Functions**

- ColumnTemperatureIC (const InputParameters &parameters)
- virtual Real value (const Point &p)

#### **Private Attributes**

• Real \_TC\_IC

#### 4.9.1 Detailed Description

Definition at line 10 of file ColumnTemperatureIC.h.

- 4.9.2 Constructor & Destructor Documentation
- 4.9.2.1 ColumnTemperaturelC::ColumnTemperaturelC ( const InputParameters & parameters )
- 4.9.3 Member Function Documentation
- 4.9.3.1 virtual Real ColumnTemperaturelC::value ( const Point & p ) [virtual]
- 4.9.4 Member Data Documentation
- **4.9.4.1 Real ColumnTemperaturelC::\_TC\_IC** [private]

Definition at line 17 of file ColumnTemperatureIC.h.

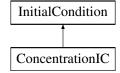
The documentation for this class was generated from the following file:

• ColumnTemperatureIC.h

## 4.10 ConcentrationIC Class Reference

#include <ConcentrationIC.h>

Inheritance diagram for ConcentrationIC:



#### **Public Member Functions**

- ConcentrationIC (const InputParameters &parameters)
- virtual Real value (const Point &p)

#### **Private Attributes**

- Real \_y\_IC
- VariableValue & temperature
- VariableValue & \_total\_pressure

## 4.10.1 Detailed Description

Definition at line 10 of file ConcentrationIC.h.

- 4.10.2 Constructor & Destructor Documentation
- 4.10.2.1 ConcentrationIC::ConcentrationIC ( const InputParameters & parameters )
- 4.10.3 Member Function Documentation
- 4.10.3.1 virtual Real ConcentrationIC::value ( const Point & p ) [virtual]
- 4.10.4 Member Data Documentation
- **4.10.4.1 VariableValue& ConcentrationIC::\_temperature** [private]

Definition at line 18 of file ConcentrationIC.h.

**4.10.4.2** VariableValue& ConcentrationIC::\_total\_pressure [private]

Definition at line 19 of file ConcentrationIC.h.

**4.10.4.3 Real ConcentrationIC::\_y\_IC** [private]

Definition at line 17 of file ConcentrationIC.h.

The documentation for this class was generated from the following file:

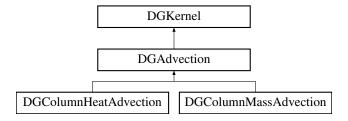
· ConcentrationIC.h

# 4.11 DGAdvection Class Reference

DGAdvection class object inherits from DGKernel object.

#include <DGAdvection.h>

Inheritance diagram for DGAdvection:



#### **Public Member Functions**

DGAdvection (const InputParameters &parameters)

Required constructor for objects in MOOSE.

#### **Protected Member Functions**

• virtual Real computeQpResidual (Moose::DGResidualType type)

Required residual function for DG kernels in MOOSE.

virtual Real computeQpJacobian (Moose::DGJacobianType type)

Required Jacobian function for DG kernels in MOOSE.

#### **Protected Attributes**

RealVectorValue \_velocity

Vector of velocity.

Real \_vx

x-component of velocity (optional - set in input file)

Real \_vy

y-component of velocity (optional - set in input file)

Real vz

z-component of velocity (optional - set in input file)

#### 4.11.1 Detailed Description

DGAdvection class object inherits from DGKernel object.

This class object inherits from the DGKernel object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinous Galerkin formulation of advection physics in the MOOSE framework. The only parameter for this kernel is a generic velocity vector, whose components can be set piecewise in the input file or by inheriting from this base class and manually altering the velocity vector.

Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 66 of file DGAdvection.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 DGAdvection::DGAdvection (const InputParameters & parameters)

Required constructor for objects in MOOSE.

4.11.3 Member Function Documentation

4.11.3.1 virtual Real DGAdvection::computeQpJacobian ( Moose::DGJacobianType type ) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented in DGColumnHeatAdvection, and DGColumnMassAdvection.

4.11.3.2 virtual Real DGAdvection::computeQpResidual ( Moose::DGResidualType type ) [protected], [virtual]

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

 $Reimplemented \ in \ DGColumn Heat Advection, \ and \ DGColumn Mass Advection.$ 

## 4.11.4 Member Data Documentation

**4.11.4.1 RealVectorValue DGAdvection::\_velocity** [protected]

Vector of velocity.

Definition at line 82 of file DGAdvection.h.

**4.11.4.2 Real DGAdvection::\_vx** [protected]

x-component of velocity (optional - set in input file)

Definition at line 83 of file DGAdvection.h.

**4.11.4.3 Real DGAdvection::\_vy** [protected]

y-component of velocity (optional - set in input file)

Definition at line 84 of file DGAdvection.h.

**4.11.4.4 Real DGAdvection::\_vz** [protected]

z-component of velocity (optional - set in input file)

Definition at line 85 of file DGAdvection.h.

The documentation for this class was generated from the following file:

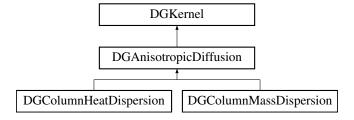
DGAdvection.h

# 4.12 DGAnisotropicDiffusion Class Reference

DGAnisotropicDiffusion class object inherits from DGKernel object.

#include <DGAnisotropicDiffusion.h>

Inheritance diagram for DGAnisotropicDiffusion:



## **Public Member Functions**

DGAnisotropicDiffusion (const InputParameters &parameters)
 Required constructor for objects in MOOSE.

## **Protected Member Functions**

virtual Real computeQpResidual (Moose::DGResidualType type)

Required residual function for DG kernels in MOOSE.

virtual Real computeQpJacobian (Moose::DGJacobianType type)

Required Jacobian function for DG kernels in MOOSE.

#### **Protected Attributes**

· Real epsilon

Penalty term for gradient jumps between the solution and test functions.

· Real sigma

Penalty term applied to element size.

RealTensorValue Diffusion

Diffusion tensor matrix parameter.

- Real Dxx
- Real \_Dxy
- Real Dxz
- Real \_Dyx
- Real \_Dyy
- Real \_Dyz
- Real \_Dzx
- Real Dzy
- Real \_Dzz

## 4.12.1 Detailed Description

DGAnisotropicDiffusion class object inherits from DGKernel object.

This class object inherits from the DGKernel object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinous Galerkin formulation of advection physics in the MOOSE framework. The only parameter for this kernel is a diffusion tensor, whose components can be set piecewise in the input file or by inheriting from this base class and manually altering the tensor matrix.

## Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 66 of file DGAnisotropicDiffusion.h.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 DGAnisotropicDiffusion::DGAnisotropicDiffusion ( const InputParameters & parameters )

Required constructor for objects in MOOSE.

4.12.3 Member Function Documentation

**4.12.3.1** virtual Real DGAnisotropicDiffusion::computeQpJacobian ( Moose::DGJacobianType *type* ) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented in DGColumnMassDispersion, and DGColumnHeatDispersion.

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented in DGColumnMassDispersion, and DGColumnHeatDispersion.

4.12.4 Member Data Documentation

**4.12.4.1 RealTensorValue DGAnisotropicDiffusion::\_Diffusion** [protected]

Diffusion tensor matrix parameter.

Definition at line 84 of file DGAnisotropicDiffusion.h.

**4.12.4.2 Real DGAnisotropicDiffusion::\_Dxx** [protected]

Definition at line 86 of file DGAnisotropicDiffusion.h.

**4.12.4.3 Real DGAnisotropicDiffusion::\_Dxy** [protected]

Definition at line 86 of file DGAnisotropicDiffusion.h.

**4.12.4.4 Real DGAnisotropicDiffusion::\_Dxz** [protected]

Definition at line 86 of file DGAnisotropicDiffusion.h.

4.12.4.5 Real DGAnisotropicDiffusion::\_Dyx [protected]

Definition at line 87 of file DGAnisotropicDiffusion.h.

**4.12.4.6 Real DGAnisotropicDiffusion::\_Dyy** [protected]

Definition at line 87 of file DGAnisotropicDiffusion.h.

**4.12.4.7 Real DGAnisotropicDiffusion::\_Dyz** [protected]

Definition at line 87 of file DGAnisotropicDiffusion.h.

**4.12.4.8 Real DGAnisotropicDiffusion::\_Dzx** [protected]

 $\label{lem:definition} \mbox{Definition at line 88 of file DGAnisotropic Diffusion.h.}$ 

**4.12.4.9 Real DGAnisotropicDiffusion::\_Dzy** [protected]

Definition at line 88 of file DGAnisotropicDiffusion.h.

**4.12.4.10 Real DGAnisotropicDiffusion::\_Dzz** [protected]

Definition at line 88 of file DGAnisotropicDiffusion.h.

**4.12.4.11 Real DGAnisotropicDiffusion::\_epsilon** [protected]

Penalty term for gradient jumps between the solution and test functions.

Definition at line 82 of file DGAnisotropicDiffusion.h.

4.12.4.12 Real DGAnisotropicDiffusion::\_sigma [protected]

Penalty term applied to element size.

Definition at line 83 of file DGAnisotropicDiffusion.h.

The documentation for this class was generated from the following file:

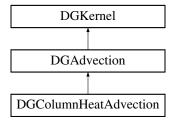
· DGAnisotropicDiffusion.h

## 4.13 DGColumnHeatAdvection Class Reference

DGColumnHeatAdvection class object inherits from DGAdvection object.

#include <DGColumnHeatAdvection.h>

Inheritance diagram for DGColumnHeatAdvection:



## **Public Member Functions**

• DGColumnHeatAdvection (const InputParameters &parameters)

Required constructor for objects in MOOSE.

## **Protected Member Functions**

- virtual Real computeQpResidual (Moose::DGResidualType type)

  Required residual function for DG kernels in MOOSE.
- virtual Real computeQpJacobian (Moose::DGJacobianType type)
   Required Jacobian function for DG kernels in MOOSE.

#### **Protected Attributes**

RealVectorValue \_velocity

Vector of velocity.

• Real \_vx

x-component of velocity (optional - set in input file)

Real \_vy

y-component of velocity (optional - set in input file)

Real \_vz

z-component of velocity (optional - set in input file)

## **Private Attributes**

const MaterialProperty < Real > & \_vel
 Reference to the velocity material property.

- const Material Property< Real > & \_gas\_density
  - Reference to the gas density material property.
- const MaterialProperty< Real > & \_gas\_heat\_capacity

Reference to the gas heat capacity material property.

#### 4.13.1 Detailed Description

DGColumnHeatAdvection class object inherits from DGAdvection object.

This class object inherits from the DGAdvection object in DGOSPREY. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinous Galerkin formulation of the heat advection physics in a fixed-bed column. Parameters for this kernel are given as material properties and will be used to override the inherited classes velocity vector.

Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 64 of file DGColumnHeatAdvection.h.

- 4.13.2 Constructor & Destructor Documentation
- 4.13.2.1 DGColumnHeatAdvection::DGColumnHeatAdvection ( const InputParameters & parameters )

Required constructor for objects in MOOSE.

- 4.13.3 Member Function Documentation
- **4.13.3.1** virtual Real DGColumnHeatAdvection::computeQpJacobian ( Moose::DGJacobianType *type* ) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGAdvection.

```
4.13.3.2 virtual Real DGColumnHeatAdvection::computeQpResidual ( Moose::DGResidualType type ) [protected], [virtual]
```

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGAdvection.

- 4.13.4 Member Data Documentation
- **4.13.4.1** const MaterialProperty<Real>& DGColumnHeatAdvection::\_gas\_density [private]

Reference to the gas density material property.

Definition at line 82 of file DGColumnHeatAdvection.h.

4.13.4.2 const MaterialProperty < Real > & DGColumnHeatAdvection::\_gas\_heat\_capacity [private]

Reference to the gas heat capacity material property.

Definition at line 83 of file DGColumnHeatAdvection.h.

4.13.4.3 const MaterialProperty<Real>& DGColumnHeatAdvection:\_vel [private]

Reference to the velocity material property.

Definition at line 81 of file DGColumnHeatAdvection.h.

**4.13.4.4 RealVectorValue DGAdvection::\_velocity** [protected], [inherited]

Vector of velocity.

Definition at line 82 of file DGAdvection.h.

4.13.4.5 Real DGAdvection::\_vx [protected], [inherited]

x-component of velocity (optional - set in input file)

Definition at line 83 of file DGAdvection.h.

4.13.4.6 Real DGAdvection::\_vy [protected], [inherited]

y-component of velocity (optional - set in input file)

Definition at line 84 of file DGAdvection.h.

**4.13.4.7 Real DGAdvection::\_vz** [protected], [inherited]

z-component of velocity (optional - set in input file)

Definition at line 85 of file DGAdvection.h.

The documentation for this class was generated from the following file:

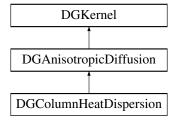
· DGColumnHeatAdvection.h

# 4.14 DGColumnHeatDispersion Class Reference

DGColumnHeatDispersion class object inherits from DGAnisotropicDiffusion object.

#include <DGColumnHeatDispersion.h>

Inheritance diagram for DGColumnHeatDispersion:



# **Public Member Functions**

DGColumnHeatDispersion (const InputParameters &parameters)
 Required constructor for objects in MOOSE.

## **Protected Member Functions**

- virtual Real computeQpResidual (Moose::DGResidualType type)
   Required residual function for DG kernels in MOOSE.
- virtual Real computeQpJacobian (Moose::DGJacobianType type)

Required Jacobian function for DG kernels in MOOSE.

#### **Protected Attributes**

Real epsilon

Penalty term for gradient jumps between the solution and test functions.

· Real sigma

Penalty term applied to element size.

• RealTensorValue \_Diffusion

Diffusion tensor matrix parameter.

- Real Dxx
- Real Dxy
- Real \_Dxz
- Real \_Dyx
- Real Dyy
- Real Dyz
- Real Dzx
- Real \_Dzy
- Real Dzz

#### **Private Attributes**

const MaterialProperty < Real > & \_conductivity

Reference to the thermal conductivity material property.

## 4.14.1 Detailed Description

DGColumnHeatDispersion class object inherits from DGAnisotropicDiffusion object.

This class object inherits from the DGAnisotropicDiffusion object in OSPREY. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinous Galerkin formulation of the heat dispersion physics in a fixed-bed column. Parameters for this kernel are given as material properties and will be used to override the inherited classes diffusion tensor.

Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 64 of file DGColumnHeatDispersion.h.

- 4.14.2 Constructor & Destructor Documentation
- 4.14.2.1 DGColumnHeatDispersion::DGColumnHeatDispersion ( const InputParameters & parameters )

Required constructor for objects in MOOSE.

- 4.14.3 Member Function Documentation
- **4.14.3.1** virtual Real DGColumnHeatDispersion::computeQpJacobian ( Moose::DGJacobianType *type* ) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGAnisotropicDiffusion.

**4.14.3.2** virtual Real DGColumnHeatDispersion::computeQpResidual( Moose::DGResidualType *type* ) [protected], [virtual]

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGAnisotropicDiffusion.

4.14.4 Member Data Documentation

**4.14.4.1** const MaterialProperty<Real>& DGColumnHeatDispersion::\_conductivity [private]

Reference to the thermal conductivity material property.

Definition at line 81 of file DGColumnHeatDispersion.h.

4.14.4.2 RealTensorValue DGAnisotropicDiffusion::\_Diffusion [protected], [inherited]

Diffusion tensor matrix parameter.

Definition at line 84 of file DGAnisotropicDiffusion.h.

4.14.4.3 Real DGAnisotropicDiffusion::\_Dxx [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

**4.14.4.4 Real DGAnisotropicDiffusion::\_Dxy** [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

**4.14.4.5** Real DGAnisotropicDiffusion::\_Dxz [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

**4.14.4.6 Real DGAnisotropicDiffusion::\_Dyx** [protected], [inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

**4.14.4.7 Real DGAnisotropicDiffusion::\_Dyy** [protected], [inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

4.14.4.8 Real DGAnisotropicDiffusion::\_Dyz [protected], [inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

**4.14.4.9 Real DGAnisotropicDiffusion::\_Dzx** [protected], [inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

**4.14.4.10 Real DGAnisotropicDiffusion::\_Dzy** [protected], [inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

**4.14.4.11 Real DGAnisotropicDiffusion::\_Dzz** [protected], [inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

**4.14.4.12** Real DGAnisotropicDiffusion::\_epsilon [protected], [inherited]

Penalty term for gradient jumps between the solution and test functions.

Definition at line 82 of file DGAnisotropicDiffusion.h.

4.14.4.13 Real DGAnisotropicDiffusion::\_sigma [protected], [inherited]

Penalty term applied to element size.

Definition at line 83 of file DGAnisotropicDiffusion.h.

The documentation for this class was generated from the following file:

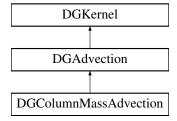
• DGColumnHeatDispersion.h

# 4.15 DGColumnMassAdvection Class Reference

DGColumnMassAdvection class object inherits from DGAdvection object.

#include <DGColumnMassAdvection.h>

Inheritance diagram for DGColumnMassAdvection:



## **Public Member Functions**

• DGColumnMassAdvection (const InputParameters &parameters)

Required constructor for objects in MOOSE.

## **Protected Member Functions**

- virtual Real computeQpResidual (Moose::DGResidualType type)

  Required residual function for DG kernels in MOOSE.
- virtual Real computeQpJacobian (Moose::DGJacobianType type)
   Required Jacobian function for DG kernels in MOOSE.

## **Protected Attributes**

- RealVectorValue \_velocity
  - Vector of velocity.
- Real \_vx

x-component of velocity (optional - set in input file)

Real \_vy

y-component of velocity (optional - set in input file)

Real \_vz

z-component of velocity (optional - set in input file)

**Private Attributes** 

const MaterialProperty < Real > & \_vel
 Reference to the velocity material property.

## 4.15.1 Detailed Description

DGColumnMassAdvection class object inherits from DGAdvection object.

This class object inherits from the DGAdvection object in DGOSPREY. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinous Galerkin formulation of the heat advection physics in a fixed-bed column. Parameters for this kernel are given as material properties and will be used to override the inherited classes velocity vector.

Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 63 of file DGColumnMassAdvection.h.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 DGColumnMassAdvection::DGColumnMassAdvection ( const InputParameters & parameters )

Required constructor for objects in MOOSE.

4.15.3 Member Function Documentation

**4.15.3.1** virtual Real DGColumnMassAdvection::computeQpJacobian ( Moose::DGJacobianType *type* ) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGAdvection.

```
4.15.3.2 virtual Real DGColumnMassAdvection::computeQpResidual ( Moose::DGResidualType type ) [protected], [virtual]
```

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGAdvection.

4.15.4 Member Data Documentation

**4.15.4.1 const MaterialProperty<Real>& DGColumnMassAdvection::\_vel** [private]

Reference to the velocity material property.

Definition at line 80 of file DGColumnMassAdvection.h.

**4.15.4.2 RealVectorValue DGAdvection::\_velocity** [protected], [inherited]

Vector of velocity.

Definition at line 82 of file DGAdvection.h.

```
4.15.4.3 Real DGAdvection::_vx [protected], [inherited]
```

x-component of velocity (optional - set in input file)

Definition at line 83 of file DGAdvection.h.

```
4.15.4.4 Real DGAdvection::_vy [protected], [inherited]
```

y-component of velocity (optional - set in input file)

Definition at line 84 of file DGAdvection.h.

```
4.15.4.5 Real DGAdvection::_vz [protected], [inherited]
```

z-component of velocity (optional - set in input file)

Definition at line 85 of file DGAdvection.h.

The documentation for this class was generated from the following file:

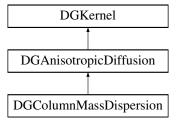
· DGColumnMassAdvection.h

## 4.16 DGColumnMassDispersion Class Reference

DGColumnMassDispersion class object inherits from DGAnisotropicDiffusion object.

#include <DGColumnMassDispersion.h>

Inheritance diagram for DGColumnMassDispersion:



### **Public Member Functions**

• DGColumnMassDispersion (const InputParameters &parameters)

Required constructor for objects in MOOSE.

#### **Protected Member Functions**

- virtual Real computeQpResidual (Moose::DGResidualType type)
  - Required residual function for DG kernels in MOOSE.
- virtual Real computeQpJacobian (Moose::DGJacobianType type)

Required Jacobian function for DG kernels in MOOSE.

# **Protected Attributes**

• Real \_epsilon

Penalty term for gradient jumps between the solution and test functions.

· Real sigma

Penalty term applied to element size.

RealTensorValue \_Diffusion

Diffusion tensor matrix parameter.

- Real \_Dxx
- Real Dxy
- Real Dxz
- Real Dyx
- Real \_Dyy
- Real \_Dyz
- Real \_Dzx
- Real \_Dzy
- Real Dzz

#### **Private Attributes**

unsigned int index

Index of the species of interest for this kernel.

· const MaterialProperty

```
< std::vector< Real > > & _dispersion
```

Reference to the axial dispersion material property.

- · const MaterialProperty
  - < std::vector< Real > > & \_molecular\_diffusion

Reference to the molecular diffusion material property.

## 4.16.1 Detailed Description

DGColumnMassDispersion class object inherits from DGAnisotropicDiffusion object.

This class object inherits from the DGAnisotropicDiffusion object in OSPREY. All public and protected members of this class are required function overrides. The object will provide residuals and Jacobians for the discontinuus Galerkin formulation of the mass dispersion physics in a fixed-bed column. Parameters for this kernel are given as material properties and will be used to override the inherited classes diffusion tensor.

Note

As a reminder, any DGKernel in MOOSE was be accompanied by the equivalent GKernel in order to provide the full residuals and Jacobians for the system.

Definition at line 65 of file DGColumnMassDispersion.h.

- 4.16.2 Constructor & Destructor Documentation
- 4.16.2.1 DGColumnMassDispersion::DGColumnMassDispersion ( const InputParameters & parameters )

Required constructor for objects in MOOSE.

- 4.16.3 Member Function Documentation
- **4.16.3.1** virtual Real DGColumnMassDispersion::computeQpJacobian ( Moose::DGJacobianType *type* ) [protected], [virtual]

Required Jacobian function for DG kernels in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGAnisotropicDiffusion.

**4.16.3.2** virtual Real DGColumnMassDispersion::computeQpResidual ( Moose::DGResidualType *type* ) [protected], [virtual]

Required residual function for DG kernels in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGAnisotropicDiffusion.

4.16.4 Member Data Documentation

**4.16.4.1 RealTensorValue DGAnisotropicDiffusion::\_Diffusion** [protected], [inherited]

Diffusion tensor matrix parameter.

Definition at line 84 of file DGAnisotropicDiffusion.h.

4.16.4.2 const MaterialProperty < std::vector < Real > > & DGColumnMassDispersion::\_dispersion [private]

Reference to the axial dispersion material property.

Definition at line 83 of file DGColumnMassDispersion.h.

**4.16.4.3 Real DGAnisotropicDiffusion::\_Dxx** [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

**4.16.4.4 Real DGAnisotropic Diffusion::\_Dxy** [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

**4.16.4.5** Real DGAnisotropicDiffusion::\_Dxz [protected], [inherited]

Definition at line 86 of file DGAnisotropicDiffusion.h.

**4.16.4.6 Real DGAnisotropicDiffusion::\_Dyx** [protected], [inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

**4.16.4.7 Real DGAnisotropicDiffusion::\_Dyy** [protected], [inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

4.16.4.8 Real DGAnisotropicDiffusion::\_Dyz [protected], [inherited]

Definition at line 87 of file DGAnisotropicDiffusion.h.

**4.16.4.9 Real DGAnisotropicDiffusion::\_Dzx** [protected], [inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

**4.16.4.10 Real DGAnisotropicDiffusion::\_Dzy** [protected], [inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

4.16.4.11 Real DGAnisotropicDiffusion::\_Dzz [protected], [inherited]

Definition at line 88 of file DGAnisotropicDiffusion.h.

**4.16.4.12 Real DGAnisotropicDiffusion::\_epsilon** [protected], [inherited]

Penalty term for gradient jumps between the solution and test functions.

Definition at line 82 of file DGAnisotropicDiffusion.h.

**4.16.4.13 unsigned int DGColumnMassDispersion::\_index** [private]

Index of the species of interest for this kernel.

Definition at line 82 of file DGColumnMassDispersion.h.

4.16.4.14 const MaterialProperty<std::vector<Real>>& DGColumnMassDispersion::\_molecular\_diffusion [private]

Reference to the molecular diffusion material property.

Definition at line 84 of file DGColumnMassDispersion.h.

4.16.4.15 Real DGAnisotropicDiffusion::\_sigma [protected], [inherited]

Penalty term applied to element size.

Definition at line 83 of file DGAnisotropicDiffusion.h.

The documentation for this class was generated from the following file:

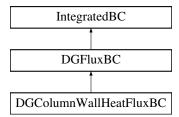
• DGColumnMassDispersion.h

## 4.17 DGColumnWallHeatFluxBC Class Reference

DGColumnWallHeatFluxBC class object inherits from DGFluxBC object.

#include <DGColumnWallHeatFluxBC.h>

Inheritance diagram for DGColumnWallHeatFluxBC:



# **Public Member Functions**

DGColumnWallHeatFluxBC (const InputParameters &parameters)
 Required constructor for BC objects in MOOSE.

#### **Protected Member Functions**

• virtual Real computeQpResidual ()

Required function override for BC objects in MOOSE.

virtual Real computeQpJacobian ()

Required function override for BC objects in MOOSE.

## **Protected Attributes**

· RealVectorValue \_velocity

Velocity vector in the system or at the boundary.

RealTensorValue \_Diffusion

Diffusivity tensory in the system or at the boundary.

- Real \_vx
- Real vy
- Real vz
- Real Dxx
- Real \_Dxy
- Real Dxz
- Real Dyx
- Real \_Dyy
- Real Dyz
- Real \_Dzx
- Real \_Dzy
- Real Dzz
- · Real \_u\_input

Value of the non-linear variable at the input of the boundary.

#### **Private Attributes**

VariableValue & wall temp

Reference to the coupled variable for wall temperature of the column.

const MaterialProperty< Real > & \_bed\_wall\_transfer\_coeff

Reference to the bed-wall transfer coefficient material property.

const MaterialProperty< Real > & \_conductivity

Reference to the thermal conductivity material property.

## 4.17.1 Detailed Description

DGColumnWallHeatFluxBC class object inherits from DGFluxBC object.

This class object inherits from the DGFluxBC object (see DGFluxBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxBC functions.

Definition at line 62 of file DGColumnWallHeatFluxBC.h.

#### 4.17.2 Constructor & Destructor Documentation

4.17.2.1 DGColumnWallHeatFluxBC::DGColumnWallHeatFluxBC ( const InputParameters & parameters )

Required constructor for BC objects in MOOSE.

## 4.17.3 Member Function Documentation

4.17.3.1 virtual Real DGColumnWallHeatFluxBC::computeQpJacobian() [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxBC.

```
4.17.3.2 virtual Real DGColumnWallHeatFluxBC::computeQpResidual() [protected], [virtual]
Required function override for BC objects in MOOSE.
This function returns a residual contribution for this object.
Reimplemented from DGFluxBC.
4.17.4 Member Data Documentation
4.17.4.1 const MaterialProperty < Real > & DGColumnWallHeatFluxBC::_bed_wall_transfer_coeff [private]
Reference to the bed-wall transfer coefficient material property.
Definition at line 81 of file DGColumnWallHeatFluxBC.h.
4.17.4.2 const MaterialProperty < Real > & DGColumnWallHeatFluxBC::_conductivity [private]
Reference to the thermal conductivity material property.
Definition at line 82 of file DGColumnWallHeatFluxBC.h.
4.17.4.3 RealTensorValue DGFluxBC::_Diffusion [protected], [inherited]
Diffusivity tensory in the system or at the boundary.
Definition at line 83 of file DGFluxBC.h.
4.17.4.4 Real DGFluxBC::_Dxx [protected], [inherited]
Definition at line 89 of file DGFluxBC.h.
4.17.4.5 Real DGFluxBC::_Dxy [protected], [inherited]
Definition at line 89 of file DGFluxBC.h.
4.17.4.6 Real DGFluxBC::_Dxz [protected], [inherited]
Definition at line 89 of file DGFluxBC.h.
4.17.4.7 Real DGFluxBC::_Dyx [protected], [inherited]
Definition at line 90 of file DGFluxBC.h.
4.17.4.8 Real DGFluxBC::_Dyy [protected], [inherited]
Definition at line 90 of file DGFluxBC.h.
4.17.4.9 Real DGFluxBC::_Dyz [protected], [inherited]
Definition at line 90 of file DGFluxBC.h.
4.17.4.10 Real DGFluxBC::_Dzx [protected], [inherited]
Definition at line 91 of file DGFluxBC.h.
4.17.4.11 Real DGFluxBC::_Dzy [protected], [inherited]
Definition at line 91 of file DGFluxBC.h.
4.17.4.12 Real DGFluxBC::_Dzz [protected], [inherited]
Definition at line 91 of file DGFluxBC.h.
```

4.17.4.13 Real DGFluxBC::\_u\_input [protected], [inherited]

Value of the non-linear variable at the input of the boundary.

Definition at line 94 of file DGFluxBC.h.

4.17.4.14 RealVectorValue DGFluxBC::\_velocity [protected], [inherited]

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

4.17.4.15 Real DGFluxBC::\_vx [protected], [inherited]

Definition at line 85 of file DGFluxBC.h.

4.17.4.16 Real DGFluxBC::\_vy [protected], [inherited]

Definition at line 86 of file DGFluxBC.h.

4.17.4.17 Real DGFluxBC::\_vz [protected], [inherited]

Definition at line 87 of file DGFluxBC.h.

4.17.4.18 VariableValue& DGColumnWallHeatFluxBC::\_wall\_temp [private]

Reference to the coupled variable for wall temperature of the column.

Definition at line 80 of file DGColumnWallHeatFluxBC.h.

The documentation for this class was generated from the following file:

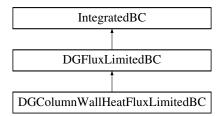
DGColumnWallHeatFluxBC.h

# 4.18 DGColumnWallHeatFluxLimitedBC Class Reference

DGColumnWallHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

#include <DGColumnWallHeatFluxLimitedBC.h>

Inheritance diagram for DGColumnWallHeatFluxLimitedBC:



# **Public Member Functions**

• DGColumnWallHeatFluxLimitedBC (const InputParameters &parameters)

Required constructor for BC objects in MOOSE.

## **Protected Member Functions**

• virtual Real computeQpResidual ()

Required function override for BC objects in MOOSE.

virtual Real computeQpJacobian ()

Required function override for BC objects in MOOSE.

## **Protected Attributes**

· Real \_epsilon

Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.

· Real sigma

Penalty term based on the size of the element at the boundary.

RealVectorValue velocity

Velocity vector in the system or at the boundary.

RealTensorValue \_Diffusion

Diffusivity tensory in the system or at the boundary.

- Real vx
- Real \_vy
- Real vz
- Real \_Dxx
- Real \_Dxy
- Real Dxz
- Real Dyx
- Real Dyy
- Real Dyz
- Real Dzx
- Real Dzy
- Real\_Dzz
- Real \_u\_input

Value of the non-linear variable at the input of the boundary.

# **Private Attributes**

VariableValue & \_wall\_temp

Reference to the coupled variable for wall temperature of the column.

const MaterialProperty< Real > & \_bed\_wall\_transfer\_coeff

Reference to the bed-wall transfer coefficient material property.

const MaterialProperty < Real > & \_conductivity

Reference to the thermal conductivity material property.

#### 4.18.1 Detailed Description

DGColumnWallHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

This class object inherits from the DGFluxLimitedBC object (see DGFluxLimitedBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxLimitedBC functions.

Definition at line 55 of file DGColumnWallHeatFluxLimitedBC.h.

#### 4.18.2 Constructor & Destructor Documentation

4.18.2.1 DGColumnWallHeatFluxLimitedBC::DGColumnWallHeatFluxLimitedBC ( const InputParameters & parameters )

Required constructor for BC objects in MOOSE.

```
4.18.3 Member Function Documentation
```

4.18.3.1 virtual Real DGColumnWallHeatFluxLimitedBC::computeQpJacobian() [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxLimitedBC.

4.18.3.2 virtual Real DGColumnWallHeatFluxLimitedBC::computeQpResidual( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGFluxLimitedBC.

4.18.4 Member Data Documentation

4.18.4.1 const MaterialProperty < Real > & DGColumnWallHeatFluxLimitedBC::\_bed\_wall\_transfer\_coeff [private]

Reference to the bed-wall transfer coefficient material property.

Definition at line 74 of file DGColumnWallHeatFluxLimitedBC.h.

4.18.4.2 const MaterialProperty < Real > & DGColumnWallHeatFluxLimitedBC::\_conductivity [private]

Reference to the thermal conductivity material property.

Definition at line 75 of file DGColumnWallHeatFluxLimitedBC.h.

4.18.4.3 RealTensorValue DGFluxLimitedBC::\_Diffusion [protected], [inherited]

Diffusivity tensory in the system or at the boundary.

Definition at line 79 of file DGFluxLimitedBC.h.

4.18.4.4 Real DGFluxLimitedBC::\_Dxx [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

4.18.4.5 Real DGFluxLimitedBC::\_Dxy [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

**4.18.4.6 Real DGFluxLimitedBC::\_Dxz** [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

**4.18.4.7 Real DGFluxLimitedBC::\_Dyx** [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

4.18.4.8 Real DGFluxLimitedBC::\_Dyy [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

**4.18.4.9 Real DGFluxLimitedBC::\_Dyz** [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

4.18.4.10 Real DGFluxLimitedBC::\_Dzx [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

**4.18.4.11 Real DGFluxLimitedBC::**\_Dzy [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

**4.18.4.12 Real DGFluxLimitedBC::\_Dzz** [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

**4.18.4.13 Real DGFluxLimitedBC::\_epsilon** [protected], [inherited]

Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.

Definition at line 72 of file DGFluxLimitedBC.h.

4.18.4.14 Real DGFluxLimitedBC::\_sigma [protected], [inherited]

Penalty term based on the size of the element at the boundary.

Definition at line 74 of file DGFluxLimitedBC.h.

4.18.4.15 Real DGFluxLimitedBC::\_u\_input [protected], [inherited]

Value of the non-linear variable at the input of the boundary.

Definition at line 90 of file DGFluxLimitedBC.h.

4.18.4.16 RealVectorValue DGFluxLimitedBC::\_velocity [protected], [inherited]

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

**4.18.4.17 Real DGFluxLimitedBC::\_vx** [protected], [inherited]

Definition at line 81 of file DGFluxLimitedBC.h.

**4.18.4.18 Real DGFluxLimitedBC::\_vy** [protected], [inherited]

Definition at line 82 of file DGFluxLimitedBC.h.

4.18.4.19 Real DGFluxLimitedBC::\_vz [protected], [inherited]

Definition at line 83 of file DGFluxLimitedBC.h.

4.18.4.20 VariableValue& DGColumnWallHeatFluxLimitedBC::\_wall\_temp [private]

Reference to the coupled variable for wall temperature of the column.

Definition at line 73 of file DGColumnWallHeatFluxLimitedBC.h.

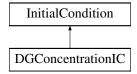
The documentation for this class was generated from the following file:

• DGColumnWallHeatFluxLimitedBC.h

## 4.19 DGConcentrationIC Class Reference

#include <DGConcentrationIC.h>

Inheritance diagram for DGConcentrationIC:



## **Public Member Functions**

- DGConcentrationIC (const InputParameters &parameters)
- virtual Real value (const Point &p)

## **Private Attributes**

- Real \_y\_IC
- · Real PT IC
- Real \_T\_IC

## 4.19.1 Detailed Description

Definition at line 10 of file DGConcentrationIC.h.

- 4.19.2 Constructor & Destructor Documentation
- 4.19.2.1 DGConcentrationIC::DGConcentrationIC ( const InputParameters & parameters )
- 4.19.3 Member Function Documentation
- **4.19.3.1** virtual Real DGConcentrationIC::value (const Point & p) [virtual]
- 4.19.4 Member Data Documentation
- 4.19.4.1 Real DGConcentrationIC::\_PT\_IC [private]

Definition at line 18 of file DGConcentrationIC.h.

**4.19.4.2 Real DGConcentrationIC::\_T\_IC** [private]

Definition at line 19 of file DGConcentrationIC.h.

4.19.4.3 Real DGConcentrationIC::\_y\_IC [private]

Definition at line 17 of file DGConcentrationIC.h.

The documentation for this class was generated from the following file:

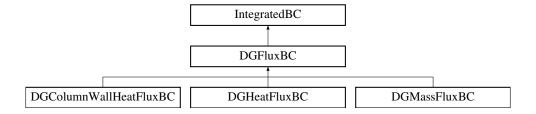
DGConcentrationIC.h

# 4.20 DGFluxBC Class Reference

DGFluxBC class object inherits from IntegratedBC object.

#include <DGFluxBC.h>

Inheritance diagram for DGFluxBC:



#### **Public Member Functions**

• DGFluxBC (const InputParameters &parameters)

Required constructor for BC objects in MOOSE.

#### **Protected Member Functions**

• virtual Real computeQpResidual ()

Required function override for BC objects in MOOSE.

• virtual Real computeQpJacobian ()

Required function override for BC objects in MOOSE.

#### **Protected Attributes**

• RealVectorValue \_velocity

Velocity vector in the system or at the boundary.

• RealTensorValue \_Diffusion

Diffusivity tensory in the system or at the boundary.

- Real \_vx
- Real \_vy
- Real \_vz
- Real \_Dxx
- Real \_Dxy
- Real \_Dxz
- Real \_Dyx
- Real \_Dyy
- Real \_Dyz
- Real \_Dzx
- Real \_Dzy
- Real \_Dzz
- · Real \_u\_input

Value of the non-linear variable at the input of the boundary.

## 4.20.1 Detailed Description

DGFluxBC class object inherits from IntegratedBC object.

This class object inherits from the IntegratedBC object. All public and protected members of this class are required function overrides. The flux BC uses the velocity and diffusivity in the system to apply a boundary condition based on whether or not material is leaving or entering the boundary.

Definition at line 63 of file DGFluxBC.h.

```
4.20.2 Constructor & Destructor Documentation
```

4.20.2.1 DGFluxBC::DGFluxBC ( const InputParameters & parameters )

Required constructor for BC objects in MOOSE.

4.20.3 Member Function Documentation

```
4.20.3.1 virtual Real DGFluxBC::computeQpJacobian ( ) [protected], [virtual]
```

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented in DGColumnWallHeatFluxBC, DGHeatFluxBC, and DGMassFluxBC.

```
4.20.3.2 virtual Real DGFluxBC::computeQpResidual() [protected], [virtual]
```

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented in DGColumnWallHeatFluxBC, DGHeatFluxBC, and DGMassFluxBC.

```
4.20.4 Member Data Documentation
```

```
4.20.4.1 RealTensorValue DGFluxBC::_Diffusion [protected]
```

Diffusivity tensory in the system or at the boundary.

Definition at line 83 of file DGFluxBC.h.

```
4.20.4.2 Real DGFluxBC::_Dxx [protected]
```

Definition at line 89 of file DGFluxBC.h.

```
4.20.4.3 Real DGFluxBC::_Dxy [protected]
```

Definition at line 89 of file DGFluxBC.h.

**4.20.4.4 Real DGFluxBC::\_Dxz** [protected]

Definition at line 89 of file DGFluxBC.h.

**4.20.4.5 Real DGFluxBC::\_Dyx** [protected]

Definition at line 90 of file DGFluxBC.h.

**4.20.4.6 Real DGFluxBC::\_Dyy** [protected]

Definition at line 90 of file DGFluxBC.h.

4.20.4.7 Real DGFluxBC::\_Dyz [protected]

Definition at line 90 of file DGFluxBC.h.

**4.20.4.8 Real DGFluxBC::\_Dzx** [protected]

Definition at line 91 of file DGFluxBC.h.

4.20.4.9 Real DGFluxBC::\_Dzy [protected]

Definition at line 91 of file DGFluxBC.h.

4.20.4.10 Real DGFluxBC::\_Dzz [protected]

Definition at line 91 of file DGFluxBC.h.

4.20.4.11 Real DGFluxBC::\_u\_input [protected]

Value of the non-linear variable at the input of the boundary.

Definition at line 94 of file DGFluxBC.h.

**4.20.4.12** RealVectorValue DGFluxBC::\_velocity [protected]

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

4.20.4.13 Real DGFluxBC::\_vx [protected]

Definition at line 85 of file DGFluxBC.h.

**4.20.4.14 Real DGFluxBC::\_vy** [protected]

Definition at line 86 of file DGFluxBC.h.

**4.20.4.15 Real DGFluxBC::\_vz** [protected]

Definition at line 87 of file DGFluxBC.h.

The documentation for this class was generated from the following file:

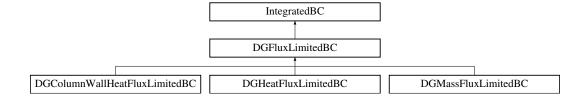
• DGFluxBC.h

## 4.21 DGFluxLimitedBC Class Reference

DGFluxLimitedBC class object inherits from IntegratedBC object.

#include <DGFluxLimitedBC.h>

Inheritance diagram for DGFluxLimitedBC:



# **Public Member Functions**

• DGFluxLimitedBC (const InputParameters &parameters)

Required constructor for BC objects in MOOSE.

#### **Protected Member Functions**

• virtual Real computeQpResidual ()

Required function override for BC objects in MOOSE.

virtual Real computeQpJacobian ()

Required function override for BC objects in MOOSE.

#### **Protected Attributes**

· Real epsilon

Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.

• Real \_sigma

Penalty term based on the size of the element at the boundary.

• RealVectorValue \_velocity

Velocity vector in the system or at the boundary.

• RealTensorValue \_Diffusion

Diffusivity tensory in the system or at the boundary.

- Real \_vx
- Real vy
- Real \_vz
- Real\_Dxx
- Real Dxy
- Real Dxz
- Real Dyx
- Real \_Dyy
- Real \_Dyz
- Real Dzx
- Real \_Dzy
- Real Dzz
- · Real \_u\_input

Value of the non-linear variable at the input of the boundary.

## 4.21.1 Detailed Description

DGFluxLimitedBC class object inherits from IntegratedBC object.

This class object inherits from the IntegratedBC object. All public and protected members of this class are required function overrides.

Definition at line 55 of file DGFluxLimitedBC.h.

# 4.21.2 Constructor & Destructor Documentation

4.21.2.1 DGFluxLimitedBC::DGFluxLimitedBC ( const InputParameters & parameters )

Required constructor for BC objects in MOOSE.

## 4.21.3 Member Function Documentation

4.21.3.1 virtual Real DGFluxLimitedBC::computeQpJacobian() [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented in DGColumnWallHeatFluxLimitedBC, DGHeatFluxLimitedBC, and DGMassFluxLimitedBC.

4.21.3.2 virtual Real DGFluxLimitedBC::computeQpResidual() [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented in DGColumnWallHeatFluxLimitedBC, DGHeatFluxLimitedBC, and DGMassFluxLimitedBC.

4.21.4 Member Data Documentation

**4.21.4.1 RealTensorValue DGFluxLimitedBC::\_Diffusion** [protected]

Diffusivity tensory in the system or at the boundary.

Definition at line 79 of file DGFluxLimitedBC.h.

**4.21.4.2 Real DGFluxLimitedBC::\_Dxx** [protected]

Definition at line 85 of file DGFluxLimitedBC.h.

**4.21.4.3 Real DGFluxLimitedBC::\_Dxy** [protected]

Definition at line 85 of file DGFluxLimitedBC.h.

**4.21.4.4 Real DGFluxLimitedBC::\_Dxz** [protected]

Definition at line 85 of file DGFluxLimitedBC.h.

**4.21.4.5 Real DGFluxLimitedBC::\_Dyx** [protected]

Definition at line 86 of file DGFluxLimitedBC.h.

**4.21.4.6 Real DGFluxLimitedBC::\_Dyy** [protected]

Definition at line 86 of file DGFluxLimitedBC.h.

**4.21.4.7 Real DGFluxLimitedBC::\_Dyz** [protected]

Definition at line 86 of file DGFluxLimitedBC.h.

**4.21.4.8 Real DGFluxLimitedBC::\_Dzx** [protected]

Definition at line 87 of file DGFluxLimitedBC.h.

**4.21.4.9 Real DGFluxLimitedBC::\_Dzy** [protected]

Definition at line 87 of file DGFluxLimitedBC.h.

**4.21.4.10** Real DGFluxLimitedBC::\_Dzz [protected]

Definition at line 87 of file DGFluxLimitedBC.h.

**4.21.4.11 Real DGFluxLimitedBC::\_epsilon** [protected]

Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.

Definition at line 72 of file DGFluxLimitedBC.h.

**4.21.4.12 Real DGFluxLimitedBC::\_sigma** [protected]

Penalty term based on the size of the element at the boundary.

Definition at line 74 of file DGFluxLimitedBC.h.

**4.21.4.13 Real DGFluxLimitedBC::\_u\_input** [protected]

Value of the non-linear variable at the input of the boundary.

Definition at line 90 of file DGFluxLimitedBC.h.

**4.21.4.14** RealVectorValue DGFluxLimitedBC::\_velocity [protected]

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

**4.21.4.15 Real DGFluxLimitedBC::\_vx** [protected]

Definition at line 81 of file DGFluxLimitedBC.h.

**4.21.4.16 Real DGFluxLimitedBC::\_vy** [protected]

Definition at line 82 of file DGFluxLimitedBC.h.

**4.21.4.17** Real DGFluxLimitedBC::\_vz [protected]

Definition at line 83 of file DGFluxLimitedBC.h.

The documentation for this class was generated from the following file:

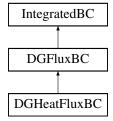
· DGFluxLimitedBC.h

# 4.22 DGHeatFluxBC Class Reference

DGHeatFluxBC class object inherits from DGFluxBC object.

#include <DGHeatFluxBC.h>

Inheritance diagram for DGHeatFluxBC:



# **Public Member Functions**

DGHeatFluxBC (const InputParameters &parameters)

Required constructor for BC objects in MOOSE.

#### **Protected Member Functions**

• virtual Real computeQpResidual ()

Required function override for BC objects in MOOSE.

• virtual Real computeQpJacobian ()

Required function override for BC objects in MOOSE.

#### **Protected Attributes**

RealVectorValue velocity

Velocity vector in the system or at the boundary.

RealTensorValue \_Diffusion

Diffusivity tensory in the system or at the boundary.

- Real \_vx
- Real \_vy
- Real vz
- Real Dxx
- Real \_Dxy
- Real Dxz
- Real \_Dyx
- Real Dyy
- . -
- Real \_DyzReal \_Dzx
- Real Dzy
- Real \_Dzz
- D. I.
- · Real \_u\_input

Value of the non-linear variable at the input of the boundary.

#### **Private Attributes**

• Real \_input\_temperature

Value of the column temperature at the inlet of the system.

const MaterialProperty< Real > & \_vel

Reference to the velocity material property.

const MaterialProperty< Real > & \_gas\_density

Reference to the gas density material property.

const MaterialProperty< Real > & \_gas\_heat\_capacity

Reference to the gas heat capacity material property.

const MaterialProperty< Real > & \_conductivity

Reference to the thermal conductivity material property.

# 4.22.1 Detailed Description

DGHeatFluxBC class object inherits from DGFluxBC object.

This class object inherits from the DGFluxBC object (see DGFluxBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxBC functions.

Definition at line 62 of file DGHeatFluxBC.h.

### 4.22.2 Constructor & Destructor Documentation

## 4.22.2.1 DGHeatFluxBC::DGHeatFluxBC ( const InputParameters & parameters )

Required constructor for BC objects in MOOSE.

```
4.22.3 Member Function Documentation
```

```
4.22.3.1 virtual Real DGHeatFluxBC::computeQpJacobian() [protected], [virtual]
```

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxBC.

```
4.22.3.2 virtual Real DGHeatFluxBC::computeQpResidual() [protected], [virtual]
```

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGFluxBC.

```
4.22.4 Member Data Documentation
```

```
4.22.4.1 const MaterialProperty<Real>& DGHeatFluxBC::_conductivity [private]
```

Reference to the thermal conductivity material property.

Definition at line 85 of file DGHeatFluxBC.h.

```
4.22.4.2 RealTensorValue DGFluxBC::_Diffusion [protected], [inherited]
```

Diffusivity tensory in the system or at the boundary.

Definition at line 83 of file DGFluxBC.h.

```
4.22.4.3 Real DGFluxBC::_Dxx [protected], [inherited]
```

Definition at line 89 of file DGFluxBC.h.

```
4.22.4.4 Real DGFluxBC::_Dxy [protected], [inherited]
```

Definition at line 89 of file DGFluxBC.h.

```
4.22.4.5 Real DGFluxBC::_Dxz [protected], [inherited]
```

Definition at line 89 of file DGFluxBC.h.

```
4.22.4.6 Real DGFluxBC::_Dyx [protected], [inherited]
```

Definition at line 90 of file DGFluxBC.h.

```
4.22.4.7 Real DGFluxBC::_Dyy [protected], [inherited]
```

Definition at line 90 of file DGFluxBC.h.

```
4.22.4.8 Real DGFluxBC::_Dyz [protected], [inherited]
```

Definition at line 90 of file DGFluxBC.h.

```
4.22.4.9 Real DGFluxBC::_Dzx [protected], [inherited]
```

Definition at line 91 of file DGFluxBC.h.

```
4.22.4.10 Real DGFluxBC::_Dzy [protected], [inherited]
```

Definition at line 91 of file DGFluxBC.h.

```
4.22.4.11 Real DGFluxBC::_Dzz [protected], [inherited]
Definition at line 91 of file DGFluxBC.h.
4.22.4.12 const MaterialProperty<Real>& DGHeatFluxBC::_gas_density [private]
Reference to the gas density material property.
Definition at line 83 of file DGHeatFluxBC.h.
4.22.4.13 const MaterialProperty < Real > & DGHeatFluxBC::_gas_heat_capacity [private]
Reference to the gas heat capacity material property.
Definition at line 84 of file DGHeatFluxBC.h.
4.22.4.14 Real DGHeatFluxBC::_input_temperature [private]
```

Value of the column temperature at the inlet of the system.

Definition at line 80 of file DGHeatFluxBC.h.

```
4.22.4.15 Real DGFluxBC::_u_input [protected], [inherited]
```

Value of the non-linear variable at the input of the boundary.

Definition at line 94 of file DGFluxBC.h.

```
4.22.4.16 const MaterialProperty<Real>& DGHeatFluxBC::_vel [private]
```

Reference to the velocity material property.

Definition at line 82 of file DGHeatFluxBC.h.

```
4.22.4.17 RealVectorValue DGFluxBC::_velocity [protected], [inherited]
```

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

```
4.22.4.18 Real DGFluxBC::_vx [protected], [inherited]
```

Definition at line 85 of file DGFluxBC.h.

```
4.22.4.19 Real DGFluxBC::_vy [protected], [inherited]
```

Definition at line 86 of file DGFluxBC.h.

```
4.22.4.20 Real DGFluxBC::_vz [protected], [inherited]
```

Definition at line 87 of file DGFluxBC.h.

The documentation for this class was generated from the following file:

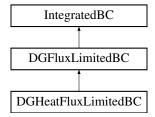
DGHeatFluxBC.h

## 4.23 DGHeatFluxLimitedBC Class Reference

DGHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

```
#include <DGHeatFluxLimitedBC.h>
```

Inheritance diagram for DGHeatFluxLimitedBC:



# **Public Member Functions**

• DGHeatFluxLimitedBC (const InputParameters &parameters)

Required constructor for BC objects in MOOSE.

#### **Protected Member Functions**

• virtual Real computeQpResidual ()

Required function override for BC objects in MOOSE.

• virtual Real computeQpJacobian ()

Required function override for BC objects in MOOSE.

#### **Protected Attributes**

• Real \_epsilon

Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.

· Real \_sigma

Penalty term based on the size of the element at the boundary.

• RealVectorValue \_velocity

Velocity vector in the system or at the boundary.

• RealTensorValue \_Diffusion

Diffusivity tensory in the system or at the boundary.

- · Real\_vx
- Real \_vy
- Real \_vz
- Real Dxx
- Real \_Dxy
- Real Dxz
- Real Dyx
- Real\_Dyy
- Real \_Dyz
- Real \_Dzx
- Real Dzy
- Real Dzz
- Real \_u\_input

Value of the non-linear variable at the input of the boundary.

#### **Private Attributes**

· Real input temperature

Value of the column temperature at the inlet of the system.

const MaterialProperty< Real > & \_vel

Reference to the velocity material property.

const MaterialProperty< Real > & \_gas\_density

Reference to the gas density material property.

const MaterialProperty < Real > & \_gas\_heat\_capacity

Reference to the gas heat capacity material property.

const MaterialProperty< Real > & \_conductivity

Reference to the thermal conductivity material property.

## 4.23.1 Detailed Description

DGHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

This class object inherits from the DGFluxLimitedBC object (see DGFluxLimitedBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxLimitedBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxLimitedBC functions.

Definition at line 55 of file DGHeatFluxLimitedBC.h.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 DGHeatFluxLimitedBC::DGHeatFluxLimitedBC ( const InputParameters & parameters )

Required constructor for BC objects in MOOSE.

4.23.3 Member Function Documentation

```
4.23.3.1 virtual Real DGHeatFluxLimitedBC::computeQpJacobian() [protected], [virtual]
```

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxLimitedBC.

```
4.23.3.2 virtual Real DGHeatFluxLimitedBC::computeQpResidual() [protected], [virtual]
```

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGFluxLimitedBC.

4.23.4 Member Data Documentation

**4.23.4.1 const MaterialProperty**<**Real**>& **DGHeatFluxLimitedBC::\_conductivity** [private]

Reference to the thermal conductivity material property.

Definition at line 78 of file DGHeatFluxLimitedBC.h.

```
4.23.4.2 RealTensorValue DGFluxLimitedBC::_Diffusion [protected], [inherited]
Diffusivity tensory in the system or at the boundary.
Definition at line 79 of file DGFluxLimitedBC.h.
4.23.4.3 Real DGFluxLimitedBC::_Dxx [protected], [inherited]
Definition at line 85 of file DGFluxLimitedBC.h.
4.23.4.4 Real DGFluxLimitedBC::_Dxy [protected], [inherited]
Definition at line 85 of file DGFluxLimitedBC.h.
4.23.4.5 Real DGFluxLimitedBC::_Dxz [protected], [inherited]
Definition at line 85 of file DGFluxLimitedBC.h.
4.23.4.6 Real DGFluxLimitedBC::_Dyx [protected], [inherited]
Definition at line 86 of file DGFluxLimitedBC.h.
4.23.4.7 Real DGFluxLimitedBC::_Dyy [protected], [inherited]
Definition at line 86 of file DGFluxLimitedBC.h.
4.23.4.8 Real DGFluxLimitedBC::_Dyz [protected], [inherited]
Definition at line 86 of file DGFluxLimitedBC.h.
4.23.4.9 Real DGFluxLimitedBC::_Dzx [protected], [inherited]
Definition at line 87 of file DGFluxLimitedBC.h.
4.23.4.10 Real DGFluxLimitedBC::_Dzy [protected], [inherited]
Definition at line 87 of file DGFluxLimitedBC.h.
4.23.4.11 Real DGFluxLimitedBC::_Dzz [protected], [inherited]
Definition at line 87 of file DGFluxLimitedBC.h.
4.23.4.12 Real DGFluxLimitedBC::_epsilon [protected], [inherited]
Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.
Definition at line 72 of file DGFluxLimitedBC.h.
4.23.4.13 const MaterialProperty<Real>& DGHeatFluxLimitedBC::_gas_density [private]
Reference to the gas density material property.
Definition at line 76 of file DGHeatFluxLimitedBC.h.
4.23.4.14 const MaterialProperty < Real > & DGHeatFluxLimitedBC::_gas_heat_capacity [private]
Reference to the gas heat capacity material property.
Definition at line 77 of file DGHeatFluxLimitedBC.h.
```

**4.23.4.15** Real DGHeatFluxLimitedBC::\_input\_temperature [private]

Value of the column temperature at the inlet of the system.

Definition at line 73 of file DGHeatFluxLimitedBC.h.

4.23.4.16 Real DGFluxLimitedBC::\_sigma [protected], [inherited]

Penalty term based on the size of the element at the boundary.

Definition at line 74 of file DGFluxLimitedBC.h.

4.23.4.17 Real DGFluxLimitedBC::\_u\_input [protected], [inherited]

Value of the non-linear variable at the input of the boundary.

Definition at line 90 of file DGFluxLimitedBC.h.

**4.23.4.18** const MaterialProperty<Real>& DGHeatFluxLimitedBC::\_vel [private]

Reference to the velocity material property.

Definition at line 75 of file DGHeatFluxLimitedBC.h.

4.23.4.19 RealVectorValue DGFluxLimitedBC::\_velocity [protected], [inherited]

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

**4.23.4.20 Real DGFluxLimitedBC::\_vx** [protected], [inherited]

Definition at line 81 of file DGFluxLimitedBC.h.

**4.23.4.21 Real DGFluxLimitedBC::\_vy** [protected], [inherited]

Definition at line 82 of file DGFluxLimitedBC.h.

**4.23.4.22 Real DGFluxLimitedBC::\_vz** [protected], [inherited]

Definition at line 83 of file DGFluxLimitedBC.h.

The documentation for this class was generated from the following file:

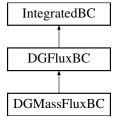
· DGHeatFluxLimitedBC.h

# 4.24 DGMassFluxBC Class Reference

DGMassFluxBC class object inherits from DGFluxBC object.

#include <DGMassFluxBC.h>

Inheritance diagram for DGMassFluxBC:



#### **Public Member Functions**

DGMassFluxBC (const InputParameters &parameters)

Required constructor for BC objects in MOOSE.

#### **Protected Member Functions**

virtual Real computeQpResidual ()

Required function override for BC objects in MOOSE.

• virtual Real computeQpJacobian ()

Required function override for BC objects in MOOSE.

#### **Protected Attributes**

RealVectorValue \_velocity

Velocity vector in the system or at the boundary.

• RealTensorValue \_Diffusion

Diffusivity tensory in the system or at the boundary.

- Real vx
- · Real\_vy
- Real vz
- Real \_Dxx
- Real Dxy
- Real \_Dxz
- Real \_Dyx
- Real Dyy
- Real \_Dyz
- Real Dzx
- Real \_Dzy
- Real \_Dzz
- Real u input

Value of the non-linear variable at the input of the boundary.

### **Private Attributes**

• Real \_input\_temperature

Value of the column temperature at the inlet of the system.

· Real \_input\_pressure

Value of the column total pressure at the inlet of the system.

• Real \_input\_molefraction

Value of the molefraction of the specific species at the inlet of the system.

const MaterialProperty< Real > & \_vel

Reference to the velocity material property.

• unsigned int \_index

Index of the species of interest at the boundary.

· const MaterialProperty

```
< std::vector< Real > > & _dispersion
```

Reference to the dispersion coefficient material property.

· const MaterialProperty

```
< std::vector< Real > > & _molecular_diffusion
```

Reference to the molecular diffusion material property.

# 4.24.1 Detailed Description

DGMassFluxBC class object inherits from DGFluxBC object.

This class object inherits from the DGFluxBC object (see DGFluxBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxBC functions.

Definition at line 62 of file DGMassFluxBC.h.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 DGMassFluxBC::DGMassFluxBC ( const InputParameters & parameters )

Required constructor for BC objects in MOOSE.

4.24.3 Member Function Documentation

**4.24.3.1 virtual Real DGMassFluxBC::computeQpJacobian()** [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxBC.

```
4.24.3.2 virtual Real DGMassFluxBC::computeQpResidual() [protected], [virtual]
```

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGFluxBC.

4.24.4 Member Data Documentation

```
4.24.4.1 RealTensorValue DGFluxBC::_Diffusion [protected], [inherited]
```

Diffusivity tensory in the system or at the boundary.

Definition at line 83 of file DGFluxBC.h.

```
4.24.4.2 const MaterialProperty<std::vector<Real>>& DGMassFluxBC::_dispersion [private]
```

Reference to the dispersion coefficient material property.

Definition at line 88 of file DGMassFluxBC.h.

```
4.24.4.3 Real DGFluxBC::_Dxx [protected], [inherited]
```

Definition at line 89 of file DGFluxBC.h.

```
4.24.4.4 Real DGFluxBC::_Dxy [protected], [inherited]
```

Definition at line 89 of file DGFluxBC.h.

```
4.24.4.5 Real DGFluxBC::_Dxz [protected], [inherited]
```

Definition at line 89 of file DGFluxBC.h.

```
4.24.4.6 Real DGFluxBC::_Dyx [protected], [inherited]
Definition at line 90 of file DGFluxBC.h.
4.24.4.7 Real DGFluxBC::_Dyy [protected], [inherited]
Definition at line 90 of file DGFluxBC.h.
4.24.4.8 Real DGFluxBC::_Dyz [protected], [inherited]
Definition at line 90 of file DGFluxBC.h.
4.24.4.9 Real DGFluxBC::_Dzx [protected], [inherited]
Definition at line 91 of file DGFluxBC.h.
4.24.4.10 Real DGFluxBC::_Dzy [protected], [inherited]
Definition at line 91 of file DGFluxBC.h.
4.24.4.11 Real DGFluxBC::_Dzz [protected], [inherited]
Definition at line 91 of file DGFluxBC.h.
4.24.4.12 unsigned int DGMassFluxBC::_index [private]
Index of the species of interest at the boundary.
Definition at line 87 of file DGMassFluxBC.h.
4.24.4.13 Real DGMassFluxBC::_input_molefraction [private]
Value of the molefraction of the specific species at the inlet of the system.
Definition at line 84 of file DGMassFluxBC.h.
4.24.4.14 Real DGMassFluxBC::_input_pressure [private]
Value of the column total pressure at the inlet of the system.
Definition at line 82 of file DGMassFluxBC.h.
4.24.4.15 Real DGMassFluxBC::_input_temperature [private]
Value of the column temperature at the inlet of the system.
Definition at line 80 of file DGMassFluxBC.h.
4.24.4.16 const MaterialProperty<std::vector<Real>>& DGMassFluxBC::_molecular_diffusion [private]
Reference to the molecular diffusion material property.
Definition at line 89 of file DGMassFluxBC.h.
4.24.4.17 Real DGFluxBC::_u_input [protected], [inherited]
Value of the non-linear variable at the input of the boundary.
Definition at line 94 of file DGFluxBC.h.
4.24.4.18 const MaterialProperty<Real>& DGMassFluxBC::_vel [private]
Reference to the velocity material property.
Definition at line 86 of file DGMassFluxBC.h.
```

4.24.4.19 RealVectorValue DGFluxBC::\_velocity [protected], [inherited]

Velocity vector in the system or at the boundary.

Definition at line 80 of file DGFluxBC.h.

4.24.4.20 Real DGFluxBC::\_vx [protected], [inherited]

Definition at line 85 of file DGFluxBC.h.

4.24.4.21 Real DGFluxBC::\_vy [protected], [inherited]

Definition at line 86 of file DGFluxBC.h.

4.24.4.22 Real DGFluxBC::\_vz [protected], [inherited]

Definition at line 87 of file DGFluxBC.h.

The documentation for this class was generated from the following file:

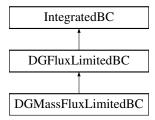
DGMassFluxBC.h

## 4.25 DGMassFluxLimitedBC Class Reference

DGMassFluxLimitedBC class object inherits from DGFluxLimitedBC object.

#include <DGMassFluxLimitedBC.h>

Inheritance diagram for DGMassFluxLimitedBC:



# **Public Member Functions**

DGMassFluxLimitedBC (const InputParameters &parameters)

Required constructor for BC objects in MOOSE.

#### **Protected Member Functions**

virtual Real computeQpResidual ()

Required function override for BC objects in MOOSE.

• virtual Real computeQpJacobian ()

Required function override for BC objects in MOOSE.

# **Protected Attributes**

Real epsilon

Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.

Real \_sigma

Penalty term based on the size of the element at the boundary.

RealVectorValue \_velocity

Velocity vector in the system or at the boundary.

• RealTensorValue Diffusion

Diffusivity tensory in the system or at the boundary.

- Real vx
- Real \_vy
- Real \_vz
- Real Dxx
- Real Dxy
- Real Dxz
- Real Dyx
- Real \_Dyy
- Real \_Dyz
- Real Dzx
- Real Dzy
- Real \_Dzz
- Real u input

Value of the non-linear variable at the input of the boundary.

## **Private Attributes**

· Real \_input\_temperature

Value of the column temperature at the inlet of the system.

Real \_input\_pressure

Value of the column total pressure at the inlet of the system.

· Real input molefraction

Value of the molefraction of the specific species at the inlet of the system.

const MaterialProperty< Real > & \_vel

Reference to the velocity material property.

· unsigned int index

Index of the species of interest at the boundary.

- · const MaterialProperty
  - < std::vector< Real > > & \_dispersion

Reference to the dispersion coefficient material property.

- · const MaterialProperty
  - < std::vector< Real > > & \_molecular\_diffusion

Reference to the molecular diffusion material property.

### 4.25.1 Detailed Description

DGMassFluxLimitedBC class object inherits from DGFluxLimitedBC object.

This class object inherits from the DGFluxLimitedBC object (see DGFluxLimitedBC.h for more details). All public and protected members of this class are required function overrides. The object will take in the given variable and material properties to override some objects declared for the generic DGFluxLimitedBC to fit this particular boundary condition. Then, it just calls the appropriate DGFluxLimitedBC functions.

Definition at line 55 of file DGMassFluxLimitedBC.h.

### 4.25.2 Constructor & Destructor Documentation

4.25.2.1 DGMassFluxLimitedBC::DGMassFluxLimitedBC ( const InputParameters & parameters )

Required constructor for BC objects in MOOSE.

```
4.25.3 Member Function Documentation
```

4.25.3.1 virtual Real DGMassFluxLimitedBC::computeQpJacobian( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a Jacobian contribution for this object. The Jacobian being computed is the associated diagonal element in the overall Jacobian matrix for the system and is used in preconditioning of the linear sub-problem.

Reimplemented from DGFluxLimitedBC.

4.25.3.2 virtual Real DGMassFluxLimitedBC::computeQpResidual( ) [protected], [virtual]

Required function override for BC objects in MOOSE.

This function returns a residual contribution for this object.

Reimplemented from DGFluxLimitedBC.

4.25.4 Member Data Documentation

**4.25.4.1 RealTensorValue DGFluxLimitedBC::\_Diffusion** [protected], [inherited]

Diffusivity tensory in the system or at the boundary.

Definition at line 79 of file DGFluxLimitedBC.h.

**4.25.4.2** const MaterialProperty < std::vector < Real > > & DGMassFluxLimitedBC::\_dispersion [private]

Reference to the dispersion coefficient material property.

Definition at line 81 of file DGMassFluxLimitedBC.h.

**4.25.4.3 Real DGFluxLimitedBC::\_Dxx** [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

4.25.4.4 Real DGFluxLimitedBC::\_Dxy [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

**4.25.4.5 Real DGFluxLimitedBC::\_Dxz** [protected], [inherited]

Definition at line 85 of file DGFluxLimitedBC.h.

4.25.4.6 Real DGFluxLimitedBC::\_Dyx [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

**4.25.4.7 Real DGFluxLimitedBC::\_Dyy** [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

**4.25.4.8 Real DGFluxLimitedBC::\_Dyz** [protected], [inherited]

Definition at line 86 of file DGFluxLimitedBC.h.

**4.25.4.9 Real DGFluxLimitedBC::\_Dzx** [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

**4.25.4.10 Real DGFluxLimitedBC::\_Dzy** [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

4.25.4.11 Real DGFluxLimitedBC::\_Dzz [protected], [inherited]

Definition at line 87 of file DGFluxLimitedBC.h.

4.25.4.12 Real DGFluxLimitedBC::\_epsilon [protected], [inherited]

Penalty term applied to the difference between the solution at the inlet and the value it is supposed to be.

Definition at line 72 of file DGFluxLimitedBC.h.

**4.25.4.13 unsigned int DGMassFluxLimitedBC::\_index** [private]

Index of the species of interest at the boundary.

Definition at line 80 of file DGMassFluxLimitedBC.h.

**4.25.4.14** Real DGMassFluxLimitedBC::\_input\_molefraction [private]

Value of the molefraction of the specific species at the inlet of the system.

Definition at line 77 of file DGMassFluxLimitedBC.h.

**4.25.4.15** Real DGMassFluxLimitedBC::\_input\_pressure [private]

Value of the column total pressure at the inlet of the system.

Definition at line 75 of file DGMassFluxLimitedBC.h.

**4.25.4.16** Real DGMassFluxLimitedBC::\_input\_temperature [private]

Value of the column temperature at the inlet of the system.

Definition at line 73 of file DGMassFluxLimitedBC.h.

**4.25.4.17** const MaterialProperty<std::vector<Real>>& DGMassFluxLimitedBC::\_molecular\_diffusion [private]

Reference to the molecular diffusion material property.

Definition at line 82 of file DGMassFluxLimitedBC.h.

**4.25.4.18 Real DGFluxLimitedBC::\_sigma** [protected], [inherited]

Penalty term based on the size of the element at the boundary.

Definition at line 74 of file DGFluxLimitedBC.h.

**4.25.4.19 Real DGFluxLimitedBC::\_u\_input** [protected], [inherited]

Value of the non-linear variable at the input of the boundary.

Definition at line 90 of file DGFluxLimitedBC.h.

**4.25.4.20** const MaterialProperty<Real>& DGMassFluxLimitedBC::\_vel [private]

Reference to the velocity material property.

Definition at line 79 of file DGMassFluxLimitedBC.h.

**4.25.4.21** RealVectorValue DGFluxLimitedBC::\_velocity [protected], [inherited]

Velocity vector in the system or at the boundary.

Definition at line 77 of file DGFluxLimitedBC.h.

4.25.4.22 Real DGFluxLimitedBC::\_vx [protected], [inherited]

Definition at line 81 of file DGFluxLimitedBC.h.

4.25.4.23 Real DGFluxLimitedBC::\_vy [protected], [inherited]

Definition at line 82 of file DGFluxLimitedBC.h.

4.25.4.24 Real DGFluxLimitedBC::\_vz [protected], [inherited]

Definition at line 83 of file DGFluxLimitedBC.h.

The documentation for this class was generated from the following file:

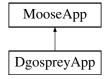
· DGMassFluxLimitedBC.h

# 4.26 DgospreyApp Class Reference

DgospreyApp inherits from the MooseApp object.

#include <DgospreyApp.h>

Inheritance diagram for DgospreyApp:



## **Public Member Functions**

• DgospreyApp (InputParameters parameters)

DgospreyApp constructor (required)

virtual ∼DgospreyApp ()

DgospreyApp destructor (requried)

### Static Public Member Functions

• static void registerApps ()

Function to the DgospreyApp into MOOSE (required)

static void registerObjects (Factory &factory)

Function to register kernels/objects created in DGOSPREY into the application (required)

static void associateSyntax (Syntax &syntax, ActionFactory &action\_factory)

Function to associate syntax with the DgospreyApp (required?)

# 4.26.1 Detailed Description

DgospreyApp inherits from the MooseApp object.

This object defines the required constructors, destructors, and functions that must be a part of every MooseApp based object. All MooseApp objects must be created in this way and override these functions.

Definition at line 54 of file DgospreyApp.h.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 DgospreyApp::DgospreyApp ( InputParameters parameters )

DgospreyApp constructor (required)

```
4.26.2.2 virtual DgospreyApp::~DgospreyApp() [virtual]
```

DgospreyApp destructor (requried)

#### 4.26.3 Member Function Documentation

4.26.3.1 static void DgospreyApp::associateSyntax ( Syntax & syntax, ActionFactory & action\_factory ) [static]

Function to associate syntax with the DgospreyApp (required?)

I don't know what this is or does or what is actually being registered.

```
4.26.3.2 static void DgospreyApp::registerApps( ) [static]
```

Function to the DgospreyApp into MOOSE (required)

```
4.26.3.3 static void DgospreyApp::registerObjects ( Factory & factory ) [static]
```

Function to register kernels/objects created in DGOSPREY into the application (required)

This is the function where the user must register all the kernels and other modules that are to be used in DGOSP-REY. Each time a new kernel or other object is created in DGOSPREY, it must be registered here prior to building and running the application. Otherwise, the new functionallity added will not show up or be utilized.

The documentation for this class was generated from the following file:

· DgospreyApp.h

# 4.27 Dispersion Class Reference

```
#include <Dispersion.h>
```

Inheritance diagram for Dispersion:



## **Public Member Functions**

Dispersion (const InputParameters &parameters)

## **Protected Member Functions**

- virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

## **Private Attributes**

- int index
- · const MaterialProperty

```
< std::vector< Real > > & _dispersion
```

## 4.27.1 Detailed Description

Definition at line 12 of file Dispersion.h.

```
4.27.2 Constructor & Destructor Documentation
```

```
4.27.2.1 Dispersion::Dispersion (const InputParameters & parameters)
```

4.27.3 Member Function Documentation

```
4.27.3.1 virtual Real Dispersion::computeQpJacobian() [protected], [virtual]
```

```
4.27.3.2 virtual Real Dispersion::computeQpResidual() [protected], [virtual]
```

4.27.4 Member Data Documentation

```
4.27.4.1 const MaterialProperty<std::vector<Real>>& Dispersion::_dispersion [private]
```

Definition at line 24 of file Dispersion.h.

```
4.27.4.2 int Dispersion::_index [private]
```

Definition at line 23 of file Dispersion.h.

The documentation for this class was generated from the following file:

Dispersion.h

# 4.28 FINCH\_DATA Struct Reference

Data structure for the FINCH object.

```
#include <finch.h>
```

# **Public Attributes**

```
• int d = 0
```

Dimension of the problem: 0 = cartesian, 1 = cylindrical, 2 = spherical.

• double dt = 0.0125

Time step.

double dt\_old = 0.0125

Previous time step.

• double T = 1.0

Total time.

double dz = 0.1

Space step.

• double L = 1.0

Total space.

• double s = 1.0

Char quantity (spherical = 1, cylindrical = length, cartesian = area)

double t = 0.0

Current Time.

double t old = 0.0

Previous Time.

double uT = 0.0

Total amount of conserved quantity in domain.

• double uT\_old = 0.0

Old Total amount of conserved quantity.

double uAvg = 0.0

Average amount of conserved quantity in domain.

• double uAvg\_old = 0.0

Old Average amount of conserved quantity.

double uIC = 0.0

Initial condition of Conserved Quantity (if constant)

double vIC = 1.0

Initial condition of Velocity (if constant)

double DIC = 1.0

Initial condition of Dispersion (if constant)

• double kIC = 1.0

Initial condition of Reaction (if constant)

double RIC = 1.0

Initial condition of the Time Coefficient (if constant)

• double <u>uo</u> = 1.0

Boundary Value of Conserved Quantity.

• double vo = 1.0

Boundary Value of Velocity.

• double **Do** = 1.0

Boundary Value of Dispersion.

• double **ko** = 1.0

Boundary Value of Reaction.

• double Ro = 1.0

Boundary Value of Time Coefficient.

• double kfn = 1.0

Film mass transfer coefficient Old.

• double kfnp1 = 1.0

Film mass transfer coefficient New.

· double lambda I

Boundary Coefficient for Implicit Neumann (Calculated at Runtime)

• double lambda\_E

Boundary Coefficient for Explicit Neumann (Calculated at Runtime)

• int LN = 10

Number of nodes.

• bool CN = true

True if Crank-Nicholson, false if Implicit, never use explicit.

bool Update = false

Flag to check if the system needs updating.

• bool Dirichlet = false

Flag to indicate use of Dirichlet or Neumann starting boundary.

bool CheckMass = false

Flag to indicate whether or not mass is to be checked.

• bool ExplicitFlux = false

Flag to indicate whether or not to use fully explicit flux limiters.

• bool Iterative = true

Flag to indicate whether to solve directly, or iteratively.

bool SteadyState = false

Flag to determine whether or not to solve the steady-state problem.

```
    bool NormTrack = true

      Flag to determine whether or not to track the norms during simulation.
• double beta = 0.5
      Scheme type indicator: 0.5=CN & 1.0=Implicit; all else NULL.
double tol_rel = 1e-6
      Relative Tolerance for Convergence.
• double tol abs = 1e-6
      Absolute Tolerance for Convergence.

    int max iter = 20

      Maximum number of iterations allowed.
• int total iter = 0
      Total number of iterations made.
int nl_method = FINCH_Picard
      Non-linear solution method - default = FINCH_Picard.

    std::vector< double > CL | I

      Left side, implicit coefficients (Calculated at Runtime)

    std::vector< double > CL_E

      Left side, explicit coefficients (Calculated at Runtime)

    std::vector< double > CC |

      Centered, implicit coefficients (Calculated at Runtime)

    std::vector< double > CC_E

      Centered, explicit coefficients (Calculated at Runtime)

    std::vector< double > CR I

      Right side, implicit coefficients (Calculated at Runtime)

    std::vector< double > CR_E

      Right side, explicit coefficients (Calculated at Runtime)

    std::vector< double > fL | I

      Left side, implicit fluxes (Calculated at Runtime)

    std::vector< double > fL_E

      Left side, explicit fluxes (Calculated at Runtime)

    std::vector< double > fC |

      Centered, implicit fluxes (Calculated at Runtime)

    std::vector< double > fC_E

      Centered, explicit fluxes (Calculated at Runtime)

    std::vector< double > fR | I

      Right side, implicit fluxes (Calculated at Runtime)

    std::vector< double > fR E

      Right side, explicit fluxes (Calculated at Runtime)

    std::vector< double > OI

      Implicit upper diagonal matrix elements (Calculated at Runtime)

    std::vector< double > OE

      Explicit upper diagonal matrix elements (Calculated at Runtime)

    std::vector< double > NI

      Implicit diagonal matrix elements (Calculated at Runtime)

    std::vector< double > NE

      Explicit diagonal matrix elements (Calculated at Runtime)

    std::vector< double > MI

      Implicit lower diagonal matrix elements (Calculated at Runtime)

    std::vector< double > ME

      Explicit lower diagonal matrix elements (Calculated at Runtime)

    std::vector< double > uz_l_l
```

```
std::vector< double > uz_lm1_l
std::vector< double > uz_lp1_l
      Implicit local slopes (Calculated at Runtime)

    std::vector< double > uz | E

 std::vector< double > uz_lm1_E

std::vector< double > uz_lp1_E
      Explicit local slopes (Calculated at Runtime)

    Matrix< double > unm1

      Conserved Quantity Older.

    Matrix< double > un

      Conserved Quantity Old.

    Matrix< double > unp1

      Conserved Quantity New.

    Matrix< double > u_star

      Conserved Quantity Projected New.

    Matrix< double > ubest

      Best found solution if solving iteratively.

    Matrix< double > vn

      Velocity Old.

    Matrix< double > vnp1

      Velocity New.

    Matrix< double > Dn

     Dispersion Old.

    Matrix< double > Dnp1

      Dispersion New.
• Matrix< double > kn
      Reaction Old.

    Matrix< double > knp1

      Reaction New.
• Matrix< double > Sn
      Forcing Function Old.

    Matrix< double > Snp1

      Forcing Function New.
• Matrix< double > Rn
      Time Coeff Old.

    Matrix< double > Rnp1

      Time Coeff New.

    Matrix< double > Fn

     Flux Limiter Old.

    Matrix< double > Fnp1

      Flux Limiter New.

    Matrix< double > gl

      Implicit Side Boundary Conditions.

    Matrix< double > gE

      Explicit Side Boundary Conditions.

    Matrix< double > res

     Current residual.

    Matrix< double > pres

     Current search direction.

    int(* callroutine )(const void *user data)

     Function pointer to executioner (DEFAULT = default_execution)
```

int(\* setic )(const void \*user\_data)

Function pointer to initial conditions (DEFAULT = default\_ic)

int(\* settime )(const void \*user\_data)

Function pointer to set time step (DEFAULT = default\_timestep)

int(\* setpreprocess )(const void \*user\_data)

Function pointer to preprocesses (DEFAULT = default\_preprocess)

int(\* solve )(const void \*user data)

Function pointer to the solver (DEFAULT = default\_solve)

int(\* setparams )(const void \*user\_data)

Function pointer to set parameters (DEFAULT = default\_params)

int(\* discretize )(const void \*user\_data)

Function pointer to discretization (DEFAULT = ospre\_discretization)

- int(\* setbcs )(const void \*user data)
- int(\* evalres )(const Matrix < double > &x, Matrix < double > &res, const void \*user\_data)

Function pointer to the residual function (DEFAULT = default\_res)

• int(\* evalprecon )(const Matrix < double > &b, Matrix < double > &p, const void \*user data)

Function pointer to the preconditioning function (DEFAULT = default\_precon)

int(\* setpostprocess )(const void \*user\_data)

Function pointer to the postprocesses (DEFAULT = default\_postprocess)

int(\* resettime )(const void \*user data)

Function pointer to reset time (DEFAULT = default\_reset)

· PICARD\_DATA picard\_dat

Data structure for PICARD method (no need to use this)

• PJFNK DATA pjfnk dat

Data structure for PJFNK method (more rigours method)

const void \* param data

User's data structure used to evaluate the parameter function (Must override if setparams is overriden)

# 4.28.1 Detailed Description

Data structure for the FINCH object.

C-style object that holds data, functions, and other structures necessary to discretize and solve a FINCH problem. All of this information must be overriden or initialized prior to running a FINCH simulation. Many, many default functions are provided to make it easier to incorporate FINCH into other problems. The main function to override will be the setparams function. This will be a function that the user provides to tell the FINCH simulation how the parameters of the problem vary in time and space and whether or not they are coupled the the variable u. All functions are overridable and several can be skipped entirely, or called directly at different times in the execution of a particular routine. This make FINCH extremely flexible to the user.

Note

All parameters and dimensions do not carry any units with them. The user is required to keep track of all their own units in their particular problem and ensure that units will cancel and be consistent in their own physical model.

Definition at line 81 of file finch.h.

4.28.2 Member Data Documentation

4.28.2.1 double FINCH\_DATA::beta = 0.5

Scheme type indicator: 0.5=CN & 1.0=Implicit; all else NULL.

Definition at line 123 of file finch.h.

4.28.2.2 int(\* FINCH\_DATA::callroutine)(const void \*user\_data)

Function pointer to executioner (DEFAULT = default\_execution)

Definition at line 186 of file finch.h.

4.28.2.3 std::vector<double> FINCH\_DATA::CC\_E

Centered, explicit coefficients (Calculated at Runtime)

Definition at line 134 of file finch.h.

4.28.2.4 std::vector<double> FINCH\_DATA::CC\_I

Centered, implicit coefficients (Calculated at Runtime)

Definition at line 133 of file finch.h.

4.28.2.5 bool FINCH\_DATA::CheckMass = false

Flag to indicate whether or not mass is to be checked.

Definition at line 117 of file finch.h.

4.28.2.6 std::vector<double> FINCH\_DATA::CL\_E

Left side, explicit coefficients (Calculated at Runtime)

Definition at line 132 of file finch.h.

4.28.2.7 std::vector<double> FINCH\_DATA::CL\_I

Left side, implicit coefficients (Calculated at Runtime)

Definition at line 131 of file finch.h.

4.28.2.8 bool FINCH\_DATA::CN = true

True if Crank-Nicholson, false if Implicit, never use explicit.

Definition at line 114 of file finch.h.

4.28.2.9 std::vector<double> FINCH\_DATA::CR\_E

Right side, explicit coefficients (Calculated at Runtime)

Definition at line 136 of file finch.h.

4.28.2.10 std::vector<double> FINCH\_DATA::CR\_I

Right side, implicit coefficients (Calculated at Runtime)

Definition at line 135 of file finch.h.

4.28.2.11 int FINCH\_DATA::d = 0

Dimension of the problem: 0 = cartesian, 1 = cylindrical, 2 = spherical.

Definition at line 84 of file finch.h.

4.28.2.12 double FINCH\_DATA::DIC = 1.0

Initial condition of Dispersion (if constant)

Definition at line 100 of file finch.h.

4.28.2.13 bool FINCH\_DATA::Dirichlet = false

Flag to indicate use of Dirichlet or Neumann starting boundary.

Definition at line 116 of file finch.h.

4.28.2.14 int(\* FINCH\_DATA::discretize)(const void \*user\_data)

Function pointer to discretization (DEFAULT = ospre\_discretization)

Definition at line 192 of file finch.h.

4.28.2.15 Matrix < double > FINCH\_DATA::Dn

Dispersion Old.

Definition at line 166 of file finch.h.

4.28.2.16 Matrix < double > FINCH\_DATA::Dnp1

Dispersion New.

Definition at line 167 of file finch.h.

4.28.2.17 double FINCH\_DATA::Do = 1.0

Boundary Value of Dispersion.

Definition at line 105 of file finch.h.

4.28.2.18 double FINCH\_DATA::dt = 0.0125

Time step.

Definition at line 85 of file finch.h.

4.28.2.19 double FINCH\_DATA::dt\_old = 0.0125

Previous time step.

Definition at line 86 of file finch.h.

4.28.2.20 double FINCH\_DATA::dz = 0.1

Space step.

Definition at line 88 of file finch.h.

4.28.2.21 int(\* FINCH\_DATA::evalprecon)(const Matrix < double > &b, Matrix < double > &p, const void \*user\_data)

Function pointer to the preconditioning function (DEFAULT = default\_precon)

Definition at line 197 of file finch.h.

4.28.2.22 int(\* FINCH\_DATA::evalres)(const Matrix < double > &x, Matrix < double > &res, const void \*user\_data)

Function pointer to the residual function (DEFAULT = default\_res)

Definition at line 195 of file finch.h.

4.28.2.23 bool FINCH\_DATA::ExplicitFlux = false

Flag to indicate whether or not to use fully explicit flux limiters.

Definition at line 118 of file finch.h.

4.28.2.24 std::vector<double> FINCH\_DATA::fC\_E

Centered, explicit fluxes (Calculated at Runtime)

Definition at line 141 of file finch.h.

4.28.2.25 std::vector<double> FINCH\_DATA::fC\_I

Centered, implicit fluxes (Calculated at Runtime)

Definition at line 140 of file finch.h.

4.28.2.26 std::vector<double> FINCH\_DATA::fL\_E

Left side, explicit fluxes (Calculated at Runtime)

Definition at line 139 of file finch.h.

4.28.2.27 std::vector<double> FINCH\_DATA::fL\_I

Left side, implicit fluxes (Calculated at Runtime)

Definition at line 138 of file finch.h.

4.28.2.28 Matrix < double > FINCH\_DATA::Fn

Flux Limiter Old.

Definition at line 175 of file finch.h.

4.28.2.29 Matrix < double > FINCH\_DATA::Fnp1

Flux Limiter New.

Definition at line 176 of file finch.h.

4.28.2.30 std::vector<double> FINCH\_DATA::fR\_E

Right side, explicit fluxes (Calculated at Runtime)

Definition at line 143 of file finch.h.

4.28.2.31 std::vector<double> FINCH\_DATA::fR\_I

Right side, implicit fluxes (Calculated at Runtime)

Definition at line 142 of file finch.h.

4.28.2.32 Matrix < double > FINCH\_DATA::gE

Explicit Side Boundary Conditions.

Definition at line 178 of file finch.h.

4.28.2.33 Matrix<double> FINCH\_DATA::gl

Implicit Side Boundary Conditions.

Definition at line 177 of file finch.h.

4.28.2.34 bool FINCH\_DATA::Iterative = true

Flag to indicate whether to solve directly, or iteratively.

Definition at line 119 of file finch.h.

4.28.2.35 double FINCH\_DATA::kfn = 1.0

Film mass transfer coefficient Old.

Definition at line 108 of file finch.h.

4.28.2.36 double FINCH\_DATA::kfnp1 = 1.0

Film mass transfer coefficient New.

Definition at line 109 of file finch.h.

4.28.2.37 double FINCH\_DATA::kIC = 1.0

Initial condition of Reaction (if constant)

Definition at line 101 of file finch.h.

4.28.2.38 Matrix < double > FINCH\_DATA::kn

Reaction Old.

Definition at line 168 of file finch.h.

4.28.2.39 Matrix<double> FINCH\_DATA::knp1

Reaction New.

Definition at line 169 of file finch.h.

4.28.2.40 double FINCH\_DATA::ko = 1.0

Boundary Value of Reaction.

Definition at line 106 of file finch.h.

4.28.2.41 double FINCH\_DATA::L = 1.0

Total space.

Definition at line 89 of file finch.h.

4.28.2.42 double FINCH\_DATA::lambda\_E

Boundary Coefficient for Explicit Neumann (Calculated at Runtime)

Definition at line 111 of file finch.h.

4.28.2.43 double FINCH\_DATA::lambda\_l

Boundary Coefficient for Implicit Neumann (Calculated at Runtime)

Definition at line 110 of file finch.h.

4.28.2.44 int FINCH\_DATA::LN = 10

Number of nodes.

Definition at line 113 of file finch.h.

4.28.2.45 int FINCH\_DATA::max\_iter = 20

Maximum number of iterations allowed.

Definition at line 126 of file finch.h.

4.28.2.46 std::vector<double> FINCH\_DATA::ME

Explicit lower diagonal matrix elements (Calculated at Runtime)

Definition at line 151 of file finch.h.

4.28.2.47 std::vector<double> FINCH\_DATA::MI

Implicit lower diagonal matrix elements (Calculated at Runtime)

Definition at line 150 of file finch.h.

4.28.2.48 std::vector<double> FINCH\_DATA::NE

Explicit diagonal matrix elements (Calculated at Runtime)

Definition at line 149 of file finch.h.

4.28.2.49 std::vector<double> FINCH\_DATA::NI

Implicit diagonal matrix elements (Calculated at Runtime)

Definition at line 148 of file finch.h.

4.28.2.50 int FINCH\_DATA::nl\_method = FINCH\_Picard

Non-linear solution method - default = FINCH Picard.

Definition at line 128 of file finch.h.

4.28.2.51 bool FINCH\_DATA::NormTrack = true

Flag to determine whether or not to track the norms during simulation.

Definition at line 121 of file finch.h.

4.28.2.52 std::vector<double> FINCH\_DATA::OE

Explicit upper diagonal matrix elements (Calculated at Runtime)

Definition at line 147 of file finch.h.

4.28.2.53 std::vector<double> FINCH\_DATA::OI

Implicit upper diagonal matrix elements (Calculated at Runtime)

Definition at line 146 of file finch.h.

4.28.2.54 const void\* FINCH\_DATA::param\_data

User's data structure used to evaluate the parameter function (Must override if setparams is overriden)

Definition at line 204 of file finch.h.

4.28.2.55 PICARD\_DATA FINCH\_DATA::picard\_dat

Data structure for PICARD method (no need to use this)

Definition at line 202 of file finch.h.

4.28.2.56 PJFNK\_DATA FINCH\_DATA::pjfnk\_dat

Data structure for PJFNK method (more rigours method)

Definition at line 203 of file finch.h.

4.28.2.57 Matrix<double> FINCH\_DATA::pres

Current search direction.

Definition at line 181 of file finch.h.

4.28.2.58 Matrix<double> FINCH\_DATA::res

Current residual.

Definition at line 180 of file finch.h.

4.28.2.59 int(\* FINCH\_DATA::resettime)(const void \*user\_data)

Function pointer to reset time (DEFAULT = default\_reset)

Definition at line 199 of file finch.h.

4.28.2.60 double FINCH\_DATA::RIC = 1.0

Initial condition of the Time Coefficient (if constant)

Definition at line 102 of file finch.h.

4.28.2.61 Matrix < double > FINCH\_DATA::Rn

Time Coeff Old.

Definition at line 172 of file finch.h.

4.28.2.62 Matrix < double > FINCH\_DATA::Rnp1

Time Coeff New.

Definition at line 173 of file finch.h.

4.28.2.63 double FINCH\_DATA::Ro = 1.0

Boundary Value of Time Coefficient.

Definition at line 107 of file finch.h.

4.28.2.64 double FINCH\_DATA::s = 1.0

Char quantity (spherical = 1, cylindrical = length, cartesian = area)

Definition at line 90 of file finch.h.

4.28.2.65 int(\* FINCH\_DATA::setbcs)(const void \*user\_data)

Function pointer to set boundary conditions (DEFAULT = default\_bcs)

Definition at line 193 of file finch.h.

4.28.2.66 int(\* FINCH\_DATA::setic)(const void \*user\_data)

Function pointer to initial conditions (DEFAULT = default\_ic)

Definition at line 187 of file finch.h.

4.28.2.67 int(\* FINCH\_DATA::setparams)(const void \*user\_data)

Function pointer to set parameters (DEFAULT = default\_params)

Definition at line 191 of file finch.h.

4.28.2.68 int(\* FINCH\_DATA::setpostprocess)(const void \*user\_data)

Function pointer to the postprocesses (DEFAULT = default\_postprocess)

Definition at line 198 of file finch.h.

4.28.2.69 int(\* FINCH\_DATA::setpreprocess)(const void \*user\_data)

Function pointer to preprocesses (DEFAULT = default preprocess)

Definition at line 189 of file finch.h.

4.28.2.70 int(\* FINCH\_DATA::settime)(const void \*user\_data)

Function pointer to set time step (DEFAULT = default\_timestep)

Definition at line 188 of file finch.h.

4.28.2.71 Matrix < double > FINCH\_DATA::Sn

Forcing Function Old.

Definition at line 170 of file finch.h.

4.28.2.72 Matrix < double > FINCH\_DATA::Snp1

Forcing Function New.

Definition at line 171 of file finch.h.

4.28.2.73 int(\* FINCH\_DATA::solve)(const void \*user\_data)

Function pointer to the solver (DEFAULT = default\_solve)

Definition at line 190 of file finch.h.

4.28.2.74 bool FINCH\_DATA::SteadyState = false

Flag to determine whether or not to solve the steady-state problem.

Definition at line 120 of file finch.h.

4.28.2.75 double FINCH\_DATA::T = 1.0

Total time.

Definition at line 87 of file finch.h.

4.28.2.76 double FINCH\_DATA::t = 0.0

Current Time.

Definition at line 91 of file finch.h.

4.28.2.77 double FINCH\_DATA::t\_old = 0.0

Previous Time.

Definition at line 92 of file finch.h.

4.28.2.78 double FINCH\_DATA::tol\_abs = 1e-6

Absolute Tolerance for Convergence.

Definition at line 125 of file finch.h.

4.28.2.79 double FINCH\_DATA::tol\_rel = 1e-6

Relative Tolerance for Convergence.

Definition at line 124 of file finch.h.

4.28.2.80 int FINCH\_DATA::total\_iter = 0

Total number of iterations made.

Definition at line 127 of file finch.h.

4.28.2.81 Matrix < double > FINCH\_DATA::u\_star

Conserved Quantity Projected New.

Definition at line 161 of file finch.h.

4.28.2.82 double FINCH\_DATA::uAvg = 0.0

Average amount of conserved quantity in domain.

Definition at line 96 of file finch.h.

4.28.2.83 double FINCH\_DATA::uAvg\_old = 0.0

Old Average amount of conserved quantity.

Definition at line 97 of file finch.h.

4.28.2.84 Matrix<double> FINCH\_DATA::ubest

Best found solution if solving iteratively.

Definition at line 162 of file finch.h.

4.28.2.85 double FINCH\_DATA::uIC = 0.0

Initial condition of Conserved Quantity (if constant)

Definition at line 98 of file finch.h.

4.28.2.86 Matrix < double > FINCH\_DATA::un

Conserved Quantity Old.

Definition at line 159 of file finch.h.

4.28.2.87 Matrix < double > FINCH\_DATA::unm1

Conserved Quantity Older.

Definition at line 158 of file finch.h.

4.28.2.88 Matrix < double > FINCH\_DATA::unp1

Conserved Quantity New.

Definition at line 160 of file finch.h.

4.28.2.89 double FINCH\_DATA::uo = 1.0

Boundary Value of Conserved Quantity.

Definition at line 103 of file finch.h.

4.28.2.90 bool FINCH\_DATA::Update = false

Flag to check if the system needs updating.

Definition at line 115 of file finch.h.

4.28.2.91 double FINCH\_DATA::uT = 0.0

Total amount of conserved quantity in domain.

Definition at line 94 of file finch.h.

4.28.2.92 double FINCH\_DATA::uT\_old = 0.0

Old Total amount of conserved quantity.

Definition at line 95 of file finch.h.

4.28.2.93 std::vector<double> FINCH\_DATA::uz\_I\_E

Definition at line 155 of file finch.h.

4.28.2.94 std::vector<double> FINCH\_DATA::uz\_l\_l

Definition at line 154 of file finch.h.

4.28.2.95 std::vector<double> FINCH\_DATA::uz\_lm1\_E

Definition at line 155 of file finch.h.

4.28.2.96 std::vector<double> FINCH\_DATA::uz\_lm1\_l

Definition at line 154 of file finch.h.

 $4.28.2.97 \quad std::vector < double > FINCH\_DATA::uz\_lp1\_E$ 

Explicit local slopes (Calculated at Runtime)

Definition at line 155 of file finch.h.

 $4.28.2.98 \quad std::vector{<}double{>} FINCH\_DATA::uz\_lp1\_l$ 

Implicit local slopes (Calculated at Runtime)

Definition at line 154 of file finch.h.

4.28.2.99 double FINCH\_DATA::vIC = 1.0

Initial condition of Velocity (if constant)

Definition at line 99 of file finch.h.

4.28.2.100 Matrix < double > FINCH\_DATA::vn

Velocity Old.

Definition at line 164 of file finch.h.

4.28.2.101 Matrix < double > FINCH\_DATA::vnp1

Velocity New.

Definition at line 165 of file finch.h.

4.28.2.102 double FINCH\_DATA::vo = 1.0

Boundary Value of Velocity.

Definition at line 104 of file finch.h.

The documentation for this struct was generated from the following file:

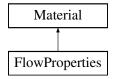
· finch.h

# 4.29 FlowProperties Class Reference

FlowProperties class object inherits from Material object.

```
#include <FlowProperties.h>
```

Inheritance diagram for FlowProperties:



#### **Public Member Functions**

• FlowProperties (const InputParameters &parameters)

Required constructor for objects in MOOSE.

# **Protected Member Functions**

virtual void computeQpProperties ()

Required function override for Material objects in MOOSE.

# **Private Attributes**

std::vector< Real > \_molecular\_wieght

Molecular wieghts for each gas species (g/mol)

std::vector< Real > \_comp\_heat\_capacity

Heat capacities for each gas species (J/g/K)

• std::vector< Real > \_comp\_ref\_viscosity

Sutherland's reference viscosity for each gas species (g/cm/s)

std::vector< Real > \_comp\_ref\_temp

Sutherland's reference temperature for each species (K)

 $\bullet \ \, std::vector < Real > \_comp\_Sutherland\_const$ 

Sutherland's constant for each gas species (K)

Real \_flow\_rate

Inlet flow rate for the fixed-bed column (cm<sup>\(\circ\)</sup>3/hr)

• Real \_column\_length

Length of the fixed-bed column (cm)

• MaterialProperty< Real > & \_velocity

MaterialProperty for the linear velocity in the bed (cm/hr)

MaterialProperty < Real > & \_gas\_density

MaterialProperty for the gas density (g/cm<sup>^</sup>3)

```
\bullet \ \ \mathsf{MaterialProperty} < \mathsf{Real} > \& \ \underline{\mathsf{gas}} \underline{\mathsf{viscosity}}
```

MaterialProperty for the gas viscosity (g/cm/s)

MaterialProperty< Real > & \_gas\_heat\_capacity

MaterialProperty for the gas heat capacity (J/g/K)

MaterialProperty< Real > & \_gas\_molecular\_wieght

MaterialProperty for the gas total molecular wieght (g/mol)

• const MaterialProperty< Real > & inner dia

Coupled material property for bed inner diameter.

const MaterialProperty< Real > & \_porosity

Coupled material property for bed bulk porosity.

const MaterialProperty< Real > & \_pellet\_density

Coupled material property for adsorbent pellet density.

 $\bullet \ \ const \ Material Property < Real > \& \_pellet\_heat\_capacity \\$ 

Coupled material property for adsorbent heat capacity.

MaterialProperty < Real > & \_heat\_retardation

MaterialProperty for energy balance retardation coefficient.

MaterialProperty< std::vector</li>

< Real >> & molecular diffusion

MaterialProperty for each species' molecular diffusion (cm<sup>2</sup>/s)

MaterialProperty < std::vector</li>

< Real > > & \_dispersion

MaterialProperty for each species' dispersion coefficient (cm\^2/hr)

MaterialProperty< std::vector</li>

< Real > > & retardation

MaterialProperty for each species' retardation coefficient.

• VariableValue & \_temperature

Reference to the coupled column temperature.

• VariableValue & \_total\_pressure

Reference to the coupled column pressure.

std::vector< unsigned int > \_index

Indices for the gas species in the system.

std::vector< VariableValue \* > \_gas\_conc

Pointer list to the coupled gases.

std::vector< VariableValue \* > \_solid\_conc

Pointer list to the coupled adsorption concentrations.

std::vector< VariableValue \* > \_solid\_perturb

Pointer list to the coupled adsorption perturbations.

### 4.29.1 Detailed Description

FlowProperties class object inherits from Material object.

This class object inherits from the Material object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will set up and calculate various flow properties including linear velocities, molecular diffusion, mechanical dispersion, gas density, gas viscosity, gas heat capacity, etc. This object also approximates the effective retardation coefficient for each species in the mass balance. The evaluation of that parameter is dependent on the solid phase concentration variable, which will either be calculated by MAGPIE (see magpie.h) or SCOPSOWL (see scopsowl.h) dependending on whether or not we will consider adsorption kinetics in the simulation.

Definition at line 63 of file FlowProperties.h.

```
4.29.2 Constructor & Destructor Documentation
4.29.2.1 FlowProperties::FlowProperties ( const InputParameters & parameters )
Required constructor for objects in MOOSE.
4.29.3 Member Function Documentation
4.29.3.1 virtual void FlowProperties::computeQpProperties() [protected], [virtual]
Required function override for Material objects in MOOSE.
This function computes the material properties when they are needed by other MOOSE objects.
4.29.4 Member Data Documentation
4.29.4.1 Real FlowProperties::_column_length [private]
Length of the fixed-bed column (cm)
Definition at line 81 of file FlowProperties.h.
4.29.4.2 std::vector<Real> FlowProperties::_comp_heat_capacity [private]
Heat capacities for each gas species (J/g/K)
Definition at line 76 of file FlowProperties.h.
4.29.4.3 std::vector<Real> FlowProperties::_comp_ref_temp [private]
Sutherland's reference temperature for each species (K)
Definition at line 78 of file FlowProperties.h.
4.29.4.4 std::vector<Real> FlowProperties::_comp_ref_viscosity [private]
Sutherland's reference viscosity for each gas species (g/cm/s)
Definition at line 77 of file FlowProperties.h.
4.29.4.5 std::vector<Real> FlowProperties::_comp_Sutherland_const [private]
Sutherland's constant for each gas species (K)
Definition at line 79 of file FlowProperties.h.
4.29.4.6 MaterialProperty<std::vector<Real>>& FlowProperties::_dispersion [private]
MaterialProperty for each species' dispersion coefficient (cm<sup>2</sup>/hr)
Definition at line 96 of file FlowProperties.h.
4.29.4.7 Real FlowProperties::_flow_rate [private]
Inlet flow rate for the fixed-bed column (cm<sup>3</sup>/hr)
Definition at line 80 of file FlowProperties.h.
4.29.4.8 std::vector<VariableValue *> FlowProperties::_gas_conc [private]
Pointer list to the coupled gases.
Definition at line 102 of file FlowProperties.h.
```

MaterialProperty<Real>& FlowProperties::\_gas\_density [private] MaterialProperty for the gas density (g/cm<sup>3</sup>) Definition at line 84 of file FlowProperties.h. **4.29.4.10** MaterialProperty < Real > & FlowProperties::\_gas\_heat\_capacity [private] MaterialProperty for the gas heat capacity (J/g/K) Definition at line 86 of file FlowProperties.h. **4.29.4.11** MaterialProperty<Real>& FlowProperties::\_gas\_molecular\_wieght [private] MaterialProperty for the gas total molecular wieght (g/mol) Definition at line 87 of file FlowProperties.h. **4.29.4.12** MaterialProperty<Real>& FlowProperties::\_gas\_viscosity [private] MaterialProperty for the gas viscosity (g/cm/s) Definition at line 85 of file FlowProperties.h. **4.29.4.13** MaterialProperty<Real>& FlowProperties::\_heat\_retardation [private] MaterialProperty for energy balance retardation coefficient. Definition at line 94 of file FlowProperties.h. **4.29.4.14** std::vector<unsigned int> FlowProperties::\_index [private] Indices for the gas species in the system. Definition at line 101 of file FlowProperties.h. **4.29.4.15** const MaterialProperty < Real > & FlowProperties::\_inner\_dia [private] Coupled material property for bed inner diameter. Definition at line 89 of file FlowProperties.h. 4.29.4.16 MaterialProperty < std::vector < Real > > & FlowProperties::\_molecular\_diffusion [private] MaterialProperty for each species' molecular diffusion (cm<sup>2</sup>/s) Definition at line 95 of file FlowProperties.h. **4.29.4.17** std::vector<Real> FlowProperties::\_molecular\_wieght [private] Molecular wieghts for each gas species (g/mol) Definition at line 75 of file FlowProperties.h. **4.29.4.18** const MaterialProperty<Real>& FlowProperties::\_pellet\_density [private] Coupled material property for adsorbent pellet density. Definition at line 91 of file FlowProperties.h. **4.29.4.19** const MaterialProperty<Real>& FlowProperties::\_pellet\_heat\_capacity [private]

Definition at line 92 of file FlowProperties.h.

Coupled material property for adsorbent heat capacity.

**4.29.4.20** const MaterialProperty<Real>& FlowProperties::\_porosity [private]

Coupled material property for bed bulk porosity.

Definition at line 90 of file FlowProperties.h.

**4.29.4.21** MaterialProperty < std::vector < Real > > & FlowProperties::\_retardation [private]

MaterialProperty for each species' retardation coefficient.

Definition at line 97 of file FlowProperties.h.

**4.29.4.22** std::vector<VariableValue \*> FlowProperties::\_solid\_conc [private]

Pointer list to the coupled adsorption concentrations.

Definition at line 103 of file FlowProperties.h.

**4.29.4.23** std::vector<VariableValue \*> FlowProperties::\_solid\_perturb [private]

Pointer list to the coupled adsorption perturbations.

Definition at line 104 of file FlowProperties.h.

**4.29.4.24** VariableValue& FlowProperties::\_temperature [private]

Reference to the coupled column temperature.

Definition at line 99 of file FlowProperties.h.

**4.29.4.25** VariableValue& FlowProperties::\_total\_pressure [private]

Reference to the coupled column pressure.

Definition at line 100 of file FlowProperties.h.

**4.29.4.26** MaterialProperty<Real>& FlowProperties::\_velocity [private]

MaterialProperty for the linear velocity in the bed (cm/hr)

Definition at line 83 of file FlowProperties.h.

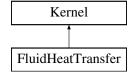
The documentation for this class was generated from the following file:

· FlowProperties.h

# 4.30 FluidHeatTransfer Class Reference

#include <FluidHeatTransfer.h>

Inheritance diagram for FluidHeatTransfer:



**Public Member Functions** 

FluidHeatTransfer (const InputParameters &parameters)

#### **Protected Member Functions**

- virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

#### **Private Attributes**

- const MaterialProperty< Real > & \_bed\_wall\_transfer\_coeff
- const MaterialProperty< Real > & inner dia
- const MaterialProperty < Real > & \_porosity
- VariableValue & \_wall\_temp

## 4.30.1 Detailed Description

Definition at line 11 of file FluidHeatTransfer.h.

- 4.30.2 Constructor & Destructor Documentation
- 4.30.2.1 FluidHeatTransfer::FluidHeatTransfer ( const InputParameters & parameters )
- 4.30.3 Member Function Documentation
- **4.30.3.1 virtual Real FluidHeatTransfer::computeQpJacobian()** [protected], [virtual]
- **4.30.3.2** virtual Real FluidHeatTransfer::computeQpResidual() [protected], [virtual]
- 4.30.4 Member Data Documentation
- 4.30.4.1 const MaterialProperty<Real>& FluidHeatTransfer::bed\_wall\_transfer\_coeff [private]

Definition at line 21 of file FluidHeatTransfer.h.

**4.30.4.2** const MaterialProperty<Real>& FluidHeatTransfer::\_inner\_dia [private]

Definition at line 22 of file FluidHeatTransfer.h.

**4.30.4.3** const MaterialProperty<Real>& FluidHeatTransfer::\_porosity [private]

Definition at line 23 of file FluidHeatTransfer.h.

**4.30.4.4 VariableValue& FluidHeatTransfer::\_wall\_temp** [private]

Definition at line 25 of file FluidHeatTransfer.h.

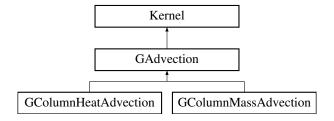
The documentation for this class was generated from the following file:

• FluidHeatTransfer.h

# 4.31 GAdvection Class Reference

#include <GAdvection.h>

Inheritance diagram for GAdvection:



#### **Public Member Functions**

GAdvection (const InputParameters &parameters)

# **Protected Member Functions**

- virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

#### **Protected Attributes**

- RealVectorValue \_velocity
- Real vx
- · Real\_vy
- Real \_vz

### 4.31.1 Detailed Description

Definition at line 13 of file GAdvection.h.

- 4.31.2 Constructor & Destructor Documentation
- 4.31.2.1 GAdvection::GAdvection (const InputParameters & parameters)
- 4.31.3 Member Function Documentation
- 4.31.3.1 virtual Real GAdvection::computeQpJacobian() [protected], [virtual]

Reimplemented in GColumnHeatAdvection, and GColumnMassAdvection.

4.31.3.2 virtual Real GAdvection::computeQpResidual() [protected], [virtual]

Reimplemented in GColumnHeatAdvection, and GColumnMassAdvection.

- 4.31.4 Member Data Documentation
- **4.31.4.1 RealVectorValue GAdvection::\_velocity** [protected]

Definition at line 23 of file GAdvection.h.

**4.31.4.2 Real GAdvection::\_vx** [protected]

Definition at line 25 of file GAdvection.h.

**4.31.4.3 Real GAdvection::\_vy** [protected]

Definition at line 26 of file GAdvection.h.

**4.31.4.4 Real GAdvection::\_vz** [protected]

Definition at line 27 of file GAdvection.h.

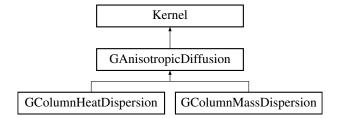
The documentation for this class was generated from the following file:

· GAdvection.h

# 4.32 GAnisotropic Diffusion Class Reference

#include <GAnisotropicDiffusion.h>

Inheritance diagram for GAnisotropicDiffusion:



## **Public Member Functions**

• GAnisotropicDiffusion (const InputParameters &parameters)

# **Protected Member Functions**

- virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

# **Protected Attributes**

- RealTensorValue \_Diffusion
- Real \_Dxx
- Real \_Dxy
- Real\_Dxz
- Real \_Dyx
- Real \_Dyy
- Real \_Dyz
- Real \_Dzx
- Real \_Dzy
- Real \_Dzz

# 4.32.1 Detailed Description

Definition at line 12 of file GAnisotropicDiffusion.h.

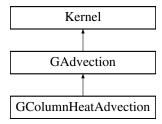
```
4.32.2 Constructor & Destructor Documentation
4.32.2.1 GAnisotropicDiffusion::GAnisotropicDiffusion ( const InputParameters & parameters )
4.32.3 Member Function Documentation
4.32.3.1 virtual Real GAnisotropicDiffusion::computeQpJacobian() [protected], [virtual]
Reimplemented in GColumnHeatDispersion, and GColumnMassDispersion.
4.32.3.2 virtual Real GAnisotropicDiffusion::computeQpResidual() [protected], [virtual]
Reimplemented in GColumnHeatDispersion, and GColumnMassDispersion.
4.32.4 Member Data Documentation
4.32.4.1 RealTensorValue GAnisotropicDiffusion::_Diffusion [protected]
Definition at line 22 of file GAnisotropicDiffusion.h.
4.32.4.2 Real GAnisotropicDiffusion::_Dxx [protected]
Definition at line 24 of file GAnisotropicDiffusion.h.
4.32.4.3 Real GAnisotropicDiffusion::_Dxy [protected]
Definition at line 24 of file GAnisotropicDiffusion.h.
4.32.4.4 Real GAnisotropicDiffusion::_Dxz [protected]
Definition at line 24 of file GAnisotropicDiffusion.h.
4.32.4.5 Real GAnisotropicDiffusion::_Dyx [protected]
Definition at line 25 of file GAnisotropicDiffusion.h.
4.32.4.6 Real GAnisotropicDiffusion::_Dyy [protected]
Definition at line 25 of file GAnisotropicDiffusion.h.
4.32.4.7 Real GAnisotropicDiffusion::_Dyz [protected]
Definition at line 25 of file GAnisotropicDiffusion.h.
4.32.4.8 Real GAnisotropicDiffusion::_Dzx [protected]
Definition at line 26 of file GAnisotropicDiffusion.h.
4.32.4.9 Real GAnisotropicDiffusion::_Dzy [protected]
Definition at line 26 of file GAnisotropicDiffusion.h.
4.32.4.10 Real GAnisotropic Diffusion::_Dzz [protected]
Definition at line 26 of file GAnisotropicDiffusion.h.
The documentation for this class was generated from the following file:
```

• GAnisotropicDiffusion.h

## 4.33 GColumnHeatAdvection Class Reference

#include <GColumnHeatAdvection.h>

Inheritance diagram for GColumnHeatAdvection:



#### **Public Member Functions**

GColumnHeatAdvection (const InputParameters &parameters)

### **Protected Member Functions**

- virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

### **Protected Attributes**

- RealVectorValue \_velocity
- Real vx
- Real \_vy
- Real \_vz

## **Private Attributes**

- const MaterialProperty< Real > & \_vel
- const MaterialProperty< Real > & \_gas\_density
- const MaterialProperty< Real > & \_gas\_heat\_capacity

## 4.33.1 Detailed Description

Definition at line 21 of file GColumnHeatAdvection.h.

- 4.33.2 Constructor & Destructor Documentation
- $4.33.2.1 \quad \textbf{GColumnHeatAdvection::} \textbf{GColumnHeatAdvection ( const InputParameters \& \textit{parameters })}$
- 4.33.3 Member Function Documentation
- 4.33.3.1 virtual Real GColumnHeatAdvection::computeQpJacobian() [protected], [virtual]

Reimplemented from GAdvection.

4.33.3.2 virtual Real GColumnHeatAdvection::computeQpResidual( ) [protected], [virtual]

Reimplemented from GAdvection.

4.33.4 Member Data Documentation

4.33.4.1 const MaterialProperty < Real > & GColumnHeatAdvection::\_gas\_density [private]

Definition at line 33 of file GColumnHeatAdvection.h.

**4.33.4.2** const MaterialProperty<Real>& GColumnHeatAdvection::\_gas\_heat\_capacity [private]

Definition at line 34 of file GColumnHeatAdvection.h.

**4.33.4.3 const MaterialProperty**<**Real**>& **GColumnHeatAdvection**::\_vel [private]

Definition at line 32 of file GColumnHeatAdvection.h.

4.33.4.4 RealVectorValue GAdvection::\_velocity [protected], [inherited]

Definition at line 23 of file GAdvection.h.

**4.33.4.5 Real GAdvection::\_vx** [protected], [inherited]

Definition at line 25 of file GAdvection.h.

**4.33.4.6 Real GAdvection::\_vy** [protected], [inherited]

Definition at line 26 of file GAdvection.h.

4.33.4.7 Real GAdvection::\_vz [protected], [inherited]

Definition at line 27 of file GAdvection.h.

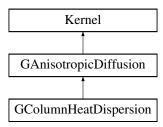
The documentation for this class was generated from the following file:

· GColumnHeatAdvection.h

## 4.34 GColumnHeatDispersion Class Reference

#include <GColumnHeatDispersion.h>

Inheritance diagram for GColumnHeatDispersion:



#### **Public Member Functions**

GColumnHeatDispersion (const InputParameters &parameters)

### **Protected Member Functions**

- virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

### **Protected Attributes**

- RealTensorValue Diffusion
- Real \_Dxx
- Real \_Dxy
- Real Dxz
- Real \_Dyx
- Real \_Dyy
- Real \_Dyz
- Real\_Dzx
- Real \_Dzy
- Real \_Dzz

#### **Private Attributes**

const MaterialProperty< Real > & \_conductivity

### 4.34.1 Detailed Description

Definition at line 20 of file GColumnHeatDispersion.h.

- 4.34.2 Constructor & Destructor Documentation
- 4.34.2.1 GColumnHeatDispersion::GColumnHeatDispersion ( const InputParameters & parameters )
- 4.34.3 Member Function Documentation
- **4.34.3.1** virtual Real GColumnHeatDispersion::computeQpJacobian() [protected], [virtual]

Reimplemented from GAnisotropicDiffusion.

4.34.3.2 virtual Real GColumnHeatDispersion::computeQpResidual() [protected], [virtual]

Reimplemented from GAnisotropicDiffusion.

- 4.34.4 Member Data Documentation
- **4.34.4.1 const MaterialProperty < Real > & GColumnHeatDispersion::\_conductivity** [private]

Definition at line 31 of file GColumnHeatDispersion.h.

4.34.4.2 RealTensorValue GAnisotropicDiffusion::\_Diffusion [protected], [inherited]

Definition at line 22 of file GAnisotropicDiffusion.h.

**4.34.4.3 Real GAnisotropicDiffusion::\_Dxx** [protected], [inherited]

Definition at line 24 of file GAnisotropicDiffusion.h.

4.34.4.4 Real GAnisotropicDiffusion::\_Dxy [protected], [inherited]

Definition at line 24 of file GAnisotropicDiffusion.h.

4.34.4.5 Real GAnisotropicDiffusion::\_Dxz [protected], [inherited]

Definition at line 24 of file GAnisotropicDiffusion.h.

4.34.4.6 Real GAnisotropicDiffusion::\_Dyx [protected], [inherited]

Definition at line 25 of file GAnisotropicDiffusion.h.

**4.34.4.7 Real GAnisotropicDiffusion::\_Dyy** [protected], [inherited]

Definition at line 25 of file GAnisotropicDiffusion.h.

**4.34.4.8 Real GAnisotropic Diffusion:: Dyz** [protected], [inherited]

Definition at line 25 of file GAnisotropicDiffusion.h.

**4.34.4.9 Real GAnisotropicDiffusion::\_Dzx** [protected], [inherited]

Definition at line 26 of file GAnisotropicDiffusion.h.

**4.34.4.10 Real GAnisotropicDiffusion::\_Dzy** [protected], [inherited]

Definition at line 26 of file GAnisotropicDiffusion.h.

**4.34.4.11 Real GAnisotropic Diffusion::\_Dzz** [protected], [inherited]

Definition at line 26 of file GAnisotropicDiffusion.h.

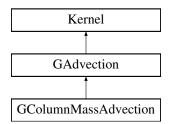
The documentation for this class was generated from the following file:

· GColumnHeatDispersion.h

## 4.35 GColumnMassAdvection Class Reference

#include <GColumnMassAdvection.h>

Inheritance diagram for GColumnMassAdvection:



## **Public Member Functions**

• GColumnMassAdvection (const InputParameters &parameters)

### **Protected Member Functions**

- · virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

## **Protected Attributes**

- RealVectorValue \_velocity
- Real vx
- Real \_vy
- Real \_vz

**Private Attributes** 

```
    const MaterialProperty< Real > & vel
```

4.35.1 Detailed Description

Definition at line 21 of file GColumnMassAdvection.h.

4.35.2 Constructor & Destructor Documentation

4.35.2.1 GColumnMassAdvection::GColumnMassAdvection ( const InputParameters & parameters )

4.35.3 Member Function Documentation

4.35.3.1 virtual Real GColumnMassAdvection::computeQpJacobian() [protected], [virtual]

Reimplemented from GAdvection.

4.35.3.2 virtual Real GColumnMassAdvection::computeQpResidual() [protected], [virtual]

Reimplemented from GAdvection.

4.35.4 Member Data Documentation

**4.35.4.1 const MaterialProperty < Real > & GColumnMassAdvection::\_vel** [private]

Definition at line 32 of file GColumnMassAdvection.h.

**4.35.4.2 RealVectorValue GAdvection::\_velocity** [protected], [inherited]

Definition at line 23 of file GAdvection.h.

**4.35.4.3 Real GAdvection::\_vx** [protected], [inherited]

Definition at line 25 of file GAdvection.h.

**4.35.4.4 Real GAdvection::\_vy** [protected], [inherited]

Definition at line 26 of file GAdvection.h.

**4.35.4.5 Real GAdvection::\_vz** [protected], [inherited]

Definition at line 27 of file GAdvection.h.

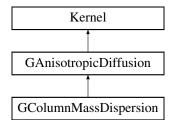
The documentation for this class was generated from the following file:

· GColumnMassAdvection.h

## 4.36 GColumnMassDispersion Class Reference

#include <GColumnMassDispersion.h>

 $Inheritance\ diagram\ for\ GColumn Mass Dispersion:$ 



#### **Public Member Functions**

• GColumnMassDispersion (const InputParameters &parameters)

### **Protected Member Functions**

- · virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

#### **Protected Attributes**

- RealTensorValue \_Diffusion
- Real Dxx
- Real \_Dxy
- Real Dxz
- Real \_Dyx
- Real\_Dyy
- Real\_Dyz
- Real \_Dzx
- Real\_Dzy
- Real \_Dzz

## **Private Attributes**

- unsigned int \_index
- const MaterialProperty

```
< std::vector< Real >> & _dispersion
```

· const MaterialProperty

< std::vector< Real > > & \_molecular\_diffusion

## 4.36.1 Detailed Description

Definition at line 20 of file GColumnMassDispersion.h.

- 4.36.2 Constructor & Destructor Documentation
- 4.36.2.1 GColumnMassDispersion::GColumnMassDispersion ( const InputParameters & parameters )
- 4.36.3 Member Function Documentation
- 4.36.3.1 virtual Real GColumnMassDispersion::computeQpJacobian( ) [protected], [virtual]

Reimplemented from GAnisotropicDiffusion.

```
4.36.3.2 virtual Real GColumnMassDispersion::computeQpResidual( ) [protected], [virtual]
Reimplemented from GAnisotropicDiffusion.
4.36.4 Member Data Documentation
4.36.4.1 RealTensorValue GAnisotropicDiffusion::_Diffusion [protected], [inherited]
Definition at line 22 of file GAnisotropicDiffusion.h.
4.36.4.2 const MaterialProperty<std::vector<Real>>& GColumnMassDispersion::_dispersion [private]
Definition at line 32 of file GColumnMassDispersion.h.
4.36.4.3 Real GAnisotropicDiffusion::_Dxx [protected], [inherited]
Definition at line 24 of file GAnisotropicDiffusion.h.
4.36.4.4 Real GAnisotropicDiffusion::_Dxy [protected], [inherited]
Definition at line 24 of file GAnisotropicDiffusion.h.
4.36.4.5 Real GAnisotropicDiffusion::_Dxz [protected], [inherited]
Definition at line 24 of file GAnisotropicDiffusion.h.
4.36.4.6 Real GAnisotropicDiffusion::_Dyx [protected], [inherited]
Definition at line 25 of file GAnisotropicDiffusion.h.
4.36.4.7 Real GAnisotropicDiffusion::_Dyy [protected], [inherited]
Definition at line 25 of file GAnisotropicDiffusion.h.
4.36.4.8 Real GAnisotropicDiffusion::_Dyz [protected], [inherited]
Definition at line 25 of file GAnisotropicDiffusion.h.
4.36.4.9 Real GAnisotropicDiffusion::_Dzx [protected], [inherited]
Definition at line 26 of file GAnisotropicDiffusion.h.
4.36.4.10 Real GAnisotropic Diffusion::_Dzy [protected], [inherited]
Definition at line 26 of file GAnisotropicDiffusion.h.
4.36.4.11 Real GAnisotropicDiffusion::_Dzz [protected], [inherited]
Definition at line 26 of file GAnisotropicDiffusion.h.
4.36.4.12 unsigned int GColumnMassDispersion::_index [private]
Definition at line 31 of file GColumnMassDispersion.h.
4.36.4.13 const MaterialProperty<std::vector<Real>>& GColumnMassDispersion::_molecular_diffusion [private]
Definition at line 33 of file GColumnMassDispersion.h.
The documentation for this class was generated from the following file:
```

Generated on Tue Nov 24 2015 11:01:41 for DGOSPREY by Doxygen

· GColumnMassDispersion.h

### 4.37 GCR\_DATA Struct Reference

Data structure for the implementation of the GCR algorithm for non-symmetric linear systems.

```
#include <lark.h>
```

#### **Public Attributes**

• int restart = -1

Restart parameter for outer iterations - default = 20.

• int maxit = 0

Maximum allowable outer iterations.

• int iter outer = 0

Number of outer iterations taken.

• int iter\_inner = 0

Number of inner iterations taken.

• int total\_iter = 0

Total number of iterations taken.

• bool breakdown = false

Boolean to determine if a step has failed.

double alpha

Inner iteration step size.

· double beta

Outer iteration step size.

double tol\_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

double tol\_abs = 1e-6

Absolute tolerance for convergence - default = 1e-6.

double res

Absolute residual norm for linear system.

· double relres

Relative residual norm for linear system.

· double relres base

Initial residual norm of the linear system.

double bestres

Best found residual norm of the linear system.

bool Output = true

True = print messages to the console.

Matrix< double > x

Current solution to the linear system.

Matrix< double > bestx

Best found solution to the linear system.

• Matrix< double > r

Residual Vector.

Matrix< double > c\_temp

Temporary c vector to be updated.

Matrix< double > u\_temp

Temporary u vector to be updated.

std::vector< Matrix< double > > u

Vector span for updating x.

std::vector< Matrix< double >> c

Vector span for updating r.

OPTRANS\_DATA transpose\_dat

Data structure for Operator Transposition.

## 4.37.1 Detailed Description

Data structure for the implementation of the GCR algorithm for non-symmetric linear systems.

C-style object used in conjunction with the Generalized Conjugate Residual (GCR) algorithm for solving a non-symmetric linear system of equations. When the linear system in question has a positive-definite-symmetric component to it, then this algorithm is equivalent to GMRESRP. However, it is generally less efficient than GMRESRP and can suffer breakdowns.

Definition at line 336 of file lark.h.

4.37.2 Member Data Documentation

4.37.2.1 double GCR\_DATA::alpha

Inner iteration step size.

Definition at line 345 of file lark.h.

4.37.2.2 double GCR\_DATA::bestres

Best found residual norm of the linear system.

Definition at line 352 of file lark.h.

4.37.2.3 Matrix<double> GCR\_DATA::bestx

Best found solution to the linear system.

Definition at line 357 of file lark.h.

4.37.2.4 double GCR\_DATA::beta

Outer iteration step size.

Definition at line 346 of file lark.h.

4.37.2.5 bool GCR\_DATA::breakdown = false

Boolean to determine if a step has failed.

Definition at line 343 of file lark.h.

4.37.2.6  $std::vector < Matrix < double > > GCR_DATA::c$ 

Vector span for updating r.

Definition at line 362 of file lark.h.

4.37.2.7 Matrix<double> GCR\_DATA::c\_temp

Temporary c vector to be updated.

Definition at line 359 of file lark.h.

4.37.2.8 int GCR\_DATA::iter\_inner = 0

Number of inner iterations taken.

Definition at line 341 of file lark.h.

4.37.2.9 int GCR\_DATA::iter\_outer = 0

Number of outer iterations taken.

Definition at line 340 of file lark.h.

4.37.2.10 int GCR\_DATA::maxit = 0

Maximum allowable outer iterations.

Definition at line 339 of file lark.h.

4.37.2.11 bool GCR\_DATA::Output = true

True = print messages to the console.

Definition at line 354 of file lark.h.

4.37.2.12 Matrix<double> GCR\_DATA::r

Residual Vector.

Definition at line 358 of file lark.h.

4.37.2.13 double GCR\_DATA::relres

Relative residual norm for linear system.

Definition at line 350 of file lark.h.

4.37.2.14 double GCR\_DATA::relres\_base

Initial residual norm of the linear system.

Definition at line 351 of file lark.h.

4.37.2.15 double GCR\_DATA::res

Absolute residual norm for linear system.

Definition at line 349 of file lark.h.

4.37.2.16 int GCR\_DATA::restart = -1

Restart parameter for outer iterations - default = 20.

Definition at line 338 of file lark.h.

4.37.2.17 double GCR\_DATA::tol\_abs = 1e-6

Absolute tolerance for convergence - default = 1e-6.

Definition at line 348 of file lark.h.

4.37.2.18 double GCR\_DATA::tol\_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 347 of file lark.h.

4.37.2.19 int GCR\_DATA::total\_iter = 0

Total number of iterations taken.

Definition at line 342 of file lark.h.

4.37.2.20 OPTRANS\_DATA GCR\_DATA::transpose\_dat

Data structure for Operator Transposition.

Definition at line 364 of file lark.h.

4.37.2.21 std::vector<Matrix<double>> GCR\_DATA::u

Vector span for updating x.

Definition at line 361 of file lark.h.

4.37.2.22 Matrix < double > GCR\_DATA::u\_temp

Temporary u vector to be updated.

Definition at line 360 of file lark.h.

4.37.2.23 Matrix < double > GCR\_DATA::x

Current solution to the linear system.

Definition at line 356 of file lark.h.

The documentation for this struct was generated from the following file:

· lark.h

### 4.38 GMRESLP\_DATA Struct Reference

Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning.

```
#include <lark.h>
```

#### **Public Attributes**

• int restart = -1

Restart parameter - default = min(vector\_size,20)

• int maxit = 0

Maximum allowable iterations - default = min(vector\_size,1000)

• int iter = 0

Number of iterations needed for convergence.

• int steps = 0

Total number of gmres iterations and krylov iterations.

• double tol\_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

• double tol\_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

• double res

Absolution redisual norm of the linear system.

· double relres

Relative residual norm of the linear system.

double relres\_base

Initial residual norm of the linear system.

· double bestres

Best found residual norm of the linear system.

• bool Output = true

True = print messages to console.

Matrix< double > x

Current solution to the linear system.

Matrix< double > bestx

Best found solution to the linear system.

• Matrix< double > r

Residual vector for the linear system.

ARNOLDI\_DATA arnoldi\_dat

Data structure for the kyrlov subspace.

### 4.38.1 Detailed Description

Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning.

C-style object used in conjunction with Generalized Minimum RESidual Left-Precondtioned (GMRESLP) and Full Orthogonalization Method (FOM) algorithms to iteratively or directly solve a linear system of equations. When using with GMRESLP, you can only check/observe the linear residuals before a restart or after the Arnoldi space is constructed. This is because this object uses Left-side Preconditioning. A faster routine may be GMRESRP, which is able to construct residuals after each Arnoldi iteration.

Definition at line 147 of file lark.h.

4.38.2 Member Data Documentation

### 4.38.2.1 ARNOLDI\_DATA GMRESLP\_DATA::arnoldi\_dat

Data structure for the kyrlov subspace.

Definition at line 167 of file lark.h.

4.38.2.2 double GMRESLP\_DATA::bestres

Best found residual norm of the linear system.

Definition at line 159 of file lark.h.

4.38.2.3 Matrix<double> GMRESLP\_DATA::bestx

Best found solution to the linear system.

Definition at line 164 of file lark.h.

4.38.2.4 int GMRESLP\_DATA::iter = 0

Number of iterations needed for convergence.

Definition at line 151 of file lark.h.

4.38.2.5 int GMRESLP\_DATA::maxit = 0

Maximum allowable iterations - default = min(vector\_size,1000)

Definition at line 150 of file lark.h.

4.38.2.6 bool GMRESLP\_DATA::Output = true

True = print messages to console.

Definition at line 161 of file lark.h.

4.38.2.7 Matrix<double> GMRESLP\_DATA::r

Residual vector for the linear system.

Definition at line 165 of file lark.h.

4.38.2.8 double GMRESLP\_DATA::relres

Relative residual norm of the linear system.

Definition at line 157 of file lark.h.

4.38.2.9 double GMRESLP\_DATA::relres\_base

Initial residual norm of the linear system.

Definition at line 158 of file lark.h.

4.38.2.10 double GMRESLP\_DATA::res

Absolution redisual norm of the linear system.

Definition at line 156 of file lark.h.

4.38.2.11 int GMRESLP\_DATA::restart = -1

Restart parameter - default = min(vector size,20)

Definition at line 149 of file lark.h.

4.38.2.12 int GMRESLP\_DATA::steps = 0

Total number of gmres iterations and krylov iterations.

Definition at line 152 of file lark.h.

4.38.2.13 double GMRESLP\_DATA::tol\_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

Definition at line 155 of file lark.h.

4.38.2.14 double GMRESLP\_DATA::tol\_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 154 of file lark.h.

4.38.2.15 Matrix < double > GMRESLP\_DATA::x

Current solution to the linear system.

Definition at line 163 of file lark.h.

The documentation for this struct was generated from the following file:

· lark.h

### 4.39 GMRESR\_DATA Struct Reference

Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMRESR)

```
#include <lark.h>
```

### **Public Attributes**

• int gcr\_restart = -1

Number of GCR restarts (default = 20, max = N)

• int gcr\_maxit = 0

Number of GCR iterations.

• int gmres\_restart = -1

Number of GMRES restarts (max = 20)

• int gmres\_maxit = 1

Number of GMRES iterations (max = 5, default = 1)

int N

Dimension of the linear system.

· int total iter

Total GMRES and GCR iterations.

int iter outer

Total GCR iterations.

· int iter inner

Total GMRES iterations.

• bool GCR\_Output = true

True = print GCR messages.

bool GMRES Output = false

True = print GMRES messages.

• double gmres\_tol = 0.1

Tolerance relative to GCR iterations.

• double gcr\_rel\_tol = 1e-6

Relative outer residual tolerance.

• double gcr\_abs\_tol = 1e-6

Absolute outer residual tolerance.

Matrix< double > arg

Argument matrix passed between preconditioner and iterator.

GCR\_DATA gcr\_dat

Data structure for the outer GCR steps.

GMRESRP\_DATA gmres\_dat

Data structure for the inner GMRES steps.

int(\* matvec )(const Matrix< double > &x, Matrix< double > &Ax, const void \*matvec\_data)

User supplied matrix-vector product function.

int(\* terminal\_precon )(const Matrix< double > &r, Matrix< double > &p, const void \*precon\_data)

Optional user supplied terminal preconditioner.

const void \* matvec\_data

Data structure for the user's matvec function.

const void \* term\_precon

Data structure for the user's terminal preconditioner.

## 4.39.1 Detailed Description

Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMRESR)

C-style object to be used in conjunction with the Generalized Minimum RESidual Recurive (GMRESR) algorithm. Although the name suggests that this method used GMRES recursively, what it is actually doing is nesting GMRE-SRP iterations inside the GCR method to form a preconditioner for GCR. The name GMRESR came from literature (Vorst and Vuik, "GMRESR: A family of nested GMRES methods", 1991).

Definition at line 373 of file lark.h.

# 4.39.2 Member Data Documentation

## 4.39.2.1 Matrix < double > GMRESR\_DATA::arg

Argument matrix passed between preconditioner and iterator.

Definition at line 391 of file lark.h.

4.39.2.2 double GMRESR\_DATA::gcr\_abs\_tol = 1e-6

Absolute outer residual tolerance.

Definition at line 389 of file lark.h.

4.39.2.3 GCR\_DATA GMRESR\_DATA::gcr\_dat

Data structure for the outer GCR steps.

Definition at line 393 of file lark.h.

4.39.2.4 int GMRESR\_DATA::gcr\_maxit = 0

Number of GCR iterations.

Definition at line 376 of file lark.h.

4.39.2.5 bool GMRESR\_DATA::GCR\_Output = true

True = print GCR messages.

Definition at line 384 of file lark.h.

4.39.2.6 double GMRESR\_DATA::gcr\_rel\_tol = 1e-6

Relative outer residual tolerance.

Definition at line 388 of file lark.h.

4.39.2.7 int GMRESR\_DATA::gcr\_restart = -1

Number of GCR restarts (default = 20, max = N)

Definition at line 375 of file lark.h.

4.39.2.8 GMRESRP\_DATA GMRESR\_DATA::gmres\_dat

Data structure for the inner GMRES steps.

Definition at line 394 of file lark.h.

4.39.2.9 int GMRESR\_DATA::gmres\_maxit = 1

Number of GMRES iterations (max = 5, default = 1)

Definition at line 378 of file lark.h.

4.39.2.10 bool GMRESR\_DATA::GMRES\_Output = false

True = print GMRES messages.

Definition at line 385 of file lark.h.

4.39.2.11 int GMRESR\_DATA::gmres\_restart = -1

Number of GMRES restarts (max = 20)

Definition at line 377 of file lark.h.

4.39.2.12 double GMRESR\_DATA::gmres\_tol = 0.1

Tolerance relative to GCR iterations.

Definition at line 387 of file lark.h.

4.39.2.13 int GMRESR\_DATA::iter\_inner

Total GMRES iterations.

Definition at line 382 of file lark.h.

4.39.2.14 int GMRESR\_DATA::iter\_outer

Total GCR iterations.

Definition at line 381 of file lark.h.

4.39.2.15 int(\* GMRESR\_DATA::matvec)(const Matrix < double > &x, Matrix < double > &Ax, const void \*matvec\_data)

User supplied matrix-vector product function.

Definition at line 397 of file lark.h.

4.39.2.16 const void \* GMRESR\_DATA::matvec\_data

Data structure for the user's matvec function.

Definition at line 401 of file lark.h.

4.39.2.17 int GMRESR\_DATA::N

Dimension of the linear system.

Definition at line 379 of file lark.h.

4.39.2.18 const void\* GMRESR\_DATA::term\_precon

Data structure for the user's terminal preconditioner.

Definition at line 402 of file lark.h.

4.39.2.19 int(\* GMRESR\_DATA::terminal\_precon)(const Matrix< double > &r, Matrix< double > &p, const void \*precon\_data)

Optional user supplied terminal preconditioner.

Definition at line 399 of file lark.h.

4.39.2.20 int GMRESR\_DATA::total\_iter

Total GMRES and GCR iterations.

Definition at line 380 of file lark.h.

The documentation for this struct was generated from the following file:

· lark.h

### 4.40 GMRESRP\_DATA Struct Reference

Data structure for the Restarted GMRES algorithm with Right Preconditioning.

```
#include <lark.h>
```

### **Public Attributes**

• int restart = -1

Restart parameter - default = min(20,vector\_size)

• int maxit = 0

Maximum allowable outer iterations.

• int iter\_outer = 0

Total number of outer iterations.

• int iter inner = 0

Total number of inner iterations.

• int iter total = 0

Total number of overall iterations.

• double tol rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

• double tol\_abs = 1e-6

Absolute tolerance for convergence - default = 1e-6.

· double res

Absolute residual norm for linear system.

· double relres

Relative residual norm for linear system.

· double relres base

Initial residual norm of the linear system.

· double bestres

Best found residual norm of the linear system.

• bool Output = true

True = print messages to console.

Matrix< double > x

Current solution to the linear system.

Matrix< double > bestx

Best found solution to the linear system.

• Matrix< double > r

Residual vector for the linear system.

std::vector< Matrix< double >> Vk

(N x k) orthonormal vector basis

std::vector< Matrix< double > > Zk

(N x k) preconditioned vector set

• std::vector< std::vector

< double > > H

(k+1 x k) upper Hessenberg storage matrix

std::vector< std::vector</li>

< double > > H bar

(k+1 x k) Factorized matrix

std::vector< double > y

(k x 1) Vector search direction

• std::vector< double > e0

(k+1 x 1) Normalized vector with residual info

std::vector< double > e0\_bar

(k+1 x 1) Factorized normal vector

Matrix< double > w

(N) x (1) interim result of the matrix\_vector multiplication

Matrix< double > v

(N) x (1) holding cell for the column entries of Vk and other interims

• Matrix< double > sum

(N) x (1) running sum of subspace vectors for use in altering w

## 4.40.1 Detailed Description

Data structure for the Restarted GMRES algorithm with Right Preconditioning.

C-style object used in conjunction with Generalized Minimum RESidual Right Preconditioned (GMRESRP) algorithm to iteratively solve a linear system of equations. Unlike GMRESLP, the GMRESRP method is capable of checking linear residuals at both the inner and outer steps. As a result, this algorithm may terminate earlier than GMRESLP if it has found a suitable solution during one of the inner steps.

Definition at line 177 of file lark.h.

4.40.2 Member Data Documentation

4.40.2.1 double GMRESRP\_DATA::bestres

Best found residual norm of the linear system.

Definition at line 190 of file lark.h.

4.40.2.2 Matrix < double > GMRESRP\_DATA::bestx

Best found solution to the linear system.

Definition at line 195 of file lark.h.

4.40.2.3 std::vector< double > GMRESRP\_DATA::e0

(k+1 x 1) Normalized vector with residual info

Definition at line 203 of file lark.h.

4.40.2.4 std::vector < double > GMRESRP\_DATA::e0\_bar

(k+1 x 1) Factorized normal vector

Definition at line 204 of file lark.h.

4.40.2.5 std::vector< std::vector< double >> GMRESRP\_DATA::H

(k+1 x k) upper Hessenberg storage matrix

Definition at line 200 of file lark.h.

4.40.2.6 std::vector< std::vector< double >> GMRESRP\_DATA::H\_bar

(k+1 x k) Factorized matrix

Definition at line 201 of file lark.h.

4.40.2.7 int GMRESRP\_DATA::iter\_inner = 0

Total number of inner iterations.

Definition at line 182 of file lark.h.

4.40.2.8 int GMRESRP\_DATA::iter\_outer = 0

Total number of outer iterations.

Definition at line 181 of file lark.h.

4.40.2.9 int GMRESRP\_DATA::iter\_total = 0

Total number of overall iterations.

Definition at line 183 of file lark.h.

4.40.2.10 int GMRESRP\_DATA::maxit = 0

Maximum allowable outer iterations.

Definition at line 180 of file lark.h.

4.40.2.11 bool GMRESRP\_DATA::Output = true

True = print messages to console.

Definition at line 192 of file lark.h.

4.40.2.12 Matrix < double > GMRESRP\_DATA::r

Residual vector for the linear system.

Definition at line 196 of file lark.h.

4.40.2.13 double GMRESRP\_DATA::relres

Relative residual norm for linear system.

Definition at line 188 of file lark.h.

4.40.2.14 double GMRESRP\_DATA::relres\_base

Initial residual norm of the linear system.

Definition at line 189 of file lark.h.

4.40.2.15 double GMRESRP\_DATA::res

Absolute residual norm for linear system.

Definition at line 187 of file lark.h.

4.40.2.16 int GMRESRP\_DATA::restart = -1

Restart parameter - default = min(20,vector\_size)

Definition at line 179 of file lark.h.

4.40.2.17 Matrix<double> GMRESRP\_DATA::sum

(N) x (1) running sum of subspace vectors for use in altering w

Definition at line 208 of file lark.h.

4.40.2.18 double GMRESRP\_DATA::tol\_abs = 1e-6

Absolute tolerance for convergence - default = 1e-6.

Definition at line 186 of file lark.h.

4.40.2.19 double GMRESRP\_DATA::tol\_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 185 of file lark.h.

4.40.2.20 Matrix < double > GMRESRP\_DATA::v

(N) x (1) holding cell for the column entries of Vk and other interims

Definition at line 207 of file lark.h.

```
4.40.2.21 std::vector < Matrix < double > > GMRESRP_DATA::Vk

(N x k) orthonormal vector basis

Definition at line 198 of file lark.h.

4.40.2.22 Matrix < double > GMRESRP_DATA::w

(N) x (1) interim result of the matrix_vector multiplication

Definition at line 206 of file lark.h.

4.40.2.23 Matrix < double > GMRESRP_DATA::x

Current solution to the linear system.

Definition at line 194 of file lark.h.

4.40.2.24 std::vector < double > GMRESRP_DATA::y

(k x 1) Vector search direction

Definition at line 202 of file lark.h.

4.40.2.25 std::vector < Matrix < double > > GMRESRP_DATA::Zk
```

The documentation for this struct was generated from the following file:

· lark.h

## 4.41 GPAST\_DATA Struct Reference

```
GPAST Data Structure.
```

```
#include <magpie.h>
```

(N x k) preconditioned vector set Definition at line 199 of file lark.h.

## **Public Attributes**

double x

Adsorbed mole fraction.

double y

Gas phase mole fraction.

• double He

Henry's Coefficient (mol/kg/kPa)

double q

Amount adsorbed for each component (mol/kg)

std::vector< double > gama\_inf

Infinite dilution activities.

• double qo

Pure component capacities (mol/kg)

double Plo

Pure component spreading pressures (mol/kg)

std::vector< double > po

Pure component reference state pressures (kPa)

double poi

Reference state pressures solved for using Recover eval GPAST.

· bool present

If true, then the component is present; if false, then the component is not present.

### 4.41.1 Detailed Description

GPAST Data Structure.

C-style object holding all parameter information associated with the Generalized Predictive Adsorbed Solution Theory (GPAST) system of equations. Each species in the gas phase will have one of these objects.

Definition at line 123 of file magpie.h.

4.41.2 Member Data Documentation

4.41.2.1 std::vector<double> GPAST\_DATA::gama\_inf

Infinite dilution activities.

Definition at line 129 of file magpie.h.

4.41.2.2 double GPAST\_DATA::He

Henry's Coefficient (mol/kg/kPa)

Definition at line 127 of file magpie.h.

4.41.2.3 double GPAST\_DATA::Plo

Pure component spreading pressures (mol/kg)

Definition at line 131 of file magpie.h.

4.41.2.4 std::vector<double> GPAST\_DATA::po

Pure component reference state pressures (kPa)

Definition at line 132 of file magpie.h.

4.41.2.5 double GPAST\_DATA::poi

Reference state pressures solved for using Recover eval GPAST.

Definition at line 133 of file magpie.h.

4.41.2.6 bool GPAST\_DATA::present

If true, then the component is present; if false, then the component is not present.

Definition at line 134 of file magpie.h.

4.41.2.7 double GPAST\_DATA::q

Amount adsorbed for each component (mol/kg)

Definition at line 128 of file magpie.h.

4.41.2.8 double GPAST\_DATA::qo

Pure component capacities (mol/kg)

Definition at line 130 of file magpie.h.

#### 4.41.2.9 double GPAST\_DATA::x

Adsorbed mole fraction.

Definition at line 125 of file magpie.h.

4.41.2.10 double GPAST\_DATA::y

Gas phase mole fraction.

Definition at line 126 of file magpie.h.

The documentation for this struct was generated from the following file:

· magpie.h

### 4.42 GSTA\_DATA Struct Reference

## GSTA Data Structure.

```
#include <magpie.h>
```

#### **Public Attributes**

double qmax

Theoretical maximum capacity of adsorbate-adsorbent pair (mol/kg)

int m

Number of parameters in the GSTA isotherm.

std::vector< double > dHo

Enthalpies for each site (J/mol)

std::vector< double > dSo

Entropies for each site (J/(K\*mol))

## 4.42.1 Detailed Description

## GSTA Data Structure.

C-style object holding all parameter information associated with the Generalized Statistical Thermodynamic Adsorption (GSTA) isotherm model. Each species in the gas phase will have one of these objects.

Definition at line 98 of file magpie.h.

## 4.42.2 Member Data Documentation

```
4.42.2.1 std::vector<double> GSTA_DATA::dHo
```

Enthalpies for each site (J/mol)

Definition at line 102 of file magpie.h.

4.42.2.2 std::vector<double> GSTA\_DATA::dSo

Entropies for each site (J/(K\*mol))

Definition at line 103 of file magpie.h.

4.42.2.3 int GSTA\_DATA::m

Number of parameters in the GSTA isotherm.

Definition at line 101 of file magpie.h.

## 4.42.2.4 double GSTA\_DATA::qmax

Theoretical maximum capacity of adsorbate-adsorbent pair (mol/kg)

Definition at line 100 of file magpie.h.

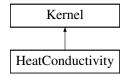
The documentation for this struct was generated from the following file:

· magpie.h

## 4.43 HeatConductivity Class Reference

```
#include <HeatConductivity.h>
```

Inheritance diagram for HeatConductivity:



#### **Public Member Functions**

HeatConductivity (const InputParameters &parameters)

### **Protected Member Functions**

- virtual Real computeQpResidual ()
- · virtual Real computeQpJacobian ()

### **Private Attributes**

- const MaterialProperty< Real > & \_conductivity
- const MaterialProperty < Real > & porosity

#### 4.43.1 Detailed Description

Definition at line 12 of file HeatConductivity.h.

- 4.43.2 Constructor & Destructor Documentation
- 4.43.2.1 HeatConductivity::HeatConductivity ( const InputParameters & parameters )
- 4.43.3 Member Function Documentation
- $\textbf{4.43.3.1} \quad \textbf{virtual Real HeatConductivity::} \textbf{computeQpJacobian()} \quad \textbf{[protected], [virtual]}$
- 4.43.3.2 virtual Real HeatConductivity::computeQpResidual( ) [protected], [virtual]
- 4.43.4 Member Data Documentation
- **4.43.4.1 const MaterialProperty<Real>& HeatConductivity::\_conductivity** [private]

Definition at line 23 of file HeatConductivity.h.

**4.43.4.2 const MaterialProperty** < Real > & HeatConductivity::\_porosity [private]

Definition at line 24 of file HeatConductivity.h.

The documentation for this class was generated from the following file:

· HeatConductivity.h

### 4.44 KMS\_DATA Struct Reference

Data structure for the implemenation of the Krylov Multi-Space (KMS) Method.

```
#include <lark.h>
```

#### **Public Attributes**

• int level = 0

Current level in the recursion.

• int max level = 0

Maximum allowable recursion levels (Default = 0 -> GMRES, Max = 5)

• int restart = -1

Restart parameter for the outer iterates (Default = 20, Max = N)

• int maxit = 0

Maximum allowable iterations for the outer steps.

• int inner\_iter = 0

Number of inner steps taken.

• int outer\_iter = 0

Number of outer steps taken.

• int total\_iter = 0

Total number of iterations in all steps.

• double outer\_reltol = 1e-6

Relative residual tolerance for outer steps (Default = 1e-6)

double outer\_abstol = 1e-6

Absolute residual tolerance for outer steps (Default = 1e-6)

• double inner\_reltol = 0.1

Residual tolerance for inner steps made relative to outer steps (Default = 0.1)

bool Output\_out = true

True = Print the outer steps residuals.

bool Output\_in = false

True = Print the inner steps residuals.

GMRESRP\_DATA gmres\_out

Data structure for the outer steps.

• std::vector< GMRESRP\_DATA > gmres\_in

Data structures for each recursion level.

int(\* matvec )(const Matrix < double > &x, Matrix < double > &Ax, const void \*matvec\_data)

User supplied matrix-vector product function.

int(\* terminal\_precon )(const Matrix< double > &r, Matrix< double > &p, const void \*precon\_data)

Optional user supplied terminal preconditioner.

const void \* matvec\_data

Data structure for the user's matvec function.

const void \* term\_precon

Data structure for the user's terminal preconditioner.

## 4.44.1 Detailed Description

Data structure for the implemenation of the Krylov Multi-Space (KMS) Method.

C-style object to be used in conjunction with the Krylov Multi-Space (KMS) Algorithm to iteratively solve non-symmetric, indefinite linear systems. This method was inspired by the Flexible GMRES (FGMRES) and Recursive GMRES (GMRESR) methods proposed by Saad (1993) and Vorst and Vuik (1991), respectively. The idea behind this method is to recursively call FGMRES to solve a linear system with pregressively smaller Krylov Subspaces built by a Right-Preconditioned GMRES algorithm. Thus creating a "V-cycle" of iteration similar to that seen in Multi-Grid algorithms.

Definition at line 413 of file lark.h.

4.44.2 Member Data Documentation

4.44.2.1 std::vector < GMRESRP\_DATA > KMS\_DATA::gmres\_in

Data structures for each recursion level.

Definition at line 431 of file lark.h.

4.44.2.2 GMRESRP\_DATA KMS\_DATA::gmres\_out

Data structure for the outer steps.

Definition at line 430 of file lark.h.

4.44.2.3 int KMS\_DATA::inner\_iter = 0

Number of inner steps taken.

Definition at line 419 of file lark.h.

4.44.2.4 double KMS\_DATA::inner\_reltol = 0.1

Residual tolerance for inner steps made relative to outer steps (Default = 0.1)

Definition at line 425 of file lark.h.

4.44.2.5 int KMS\_DATA::level = 0

Current level in the recursion.

Definition at line 415 of file lark.h.

4.44.2.6 int(\* KMS\_DATA::matvec)(const Matrix < double > &x, Matrix < double > &Ax, const void \*matvec\_data)

User supplied matrix-vector product function.

Definition at line 434 of file lark.h.

4.44.2.7 const void\* KMS\_DATA::matvec\_data

Data structure for the user's matvec function.

Definition at line 438 of file lark.h.

4.44.2.8 int KMS\_DATA::max\_level = 0

Maximum allowable recursion levels (Default = 0 -> GMRES, Max = 5)

Definition at line 416 of file lark.h.

4.44.2.9 int KMS\_DATA::maxit = 0

Maximum allowable iterations for the outer steps.

Definition at line 418 of file lark.h.

4.44.2.10 double KMS\_DATA::outer\_abstol = 1e-6

Absolute residual tolerance for outer steps (Default = 1e-6)

Definition at line 424 of file lark.h.

4.44.2.11 int KMS\_DATA::outer\_iter = 0

Number of outer steps taken.

Definition at line 420 of file lark.h.

4.44.2.12 double KMS\_DATA::outer\_reltol = 1e-6

Relative residual tolerance for outer steps (Default = 1e-6)

Definition at line 423 of file lark.h.

4.44.2.13 bool KMS\_DATA::Output\_in = false

True = Print the inner steps residuals.

Definition at line 428 of file lark.h.

4.44.2.14 bool KMS\_DATA::Output\_out = true

True = Print the outer steps residuals.

Definition at line 427 of file lark.h.

4.44.2.15 int KMS\_DATA::restart = -1

Restart parameter for the outer iterates (Default = 20, Max = N)

Definition at line 417 of file lark.h.

4.44.2.16 const void\* KMS\_DATA::term\_precon

Data structure for the user's terminal preconditioner.

Definition at line 439 of file lark.h.

 $4.44.2.17 \quad int(*\ KMS\_DATA::terminal\_precon)(const\ Matrix < double > \&r,\ Matrix < double > \&p,\ const\ void\ *precon\_data)$ 

Optional user supplied terminal preconditioner.

Definition at line 436 of file lark.h.

4.44.2.18 int KMS\_DATA::total\_iter = 0

Total number of iterations in all steps.

Definition at line 421 of file lark.h.

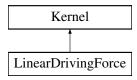
The documentation for this struct was generated from the following file:

· lark.h

## 4.45 LinearDrivingForce Class Reference

#include <LinearDrivingForce.h>

Inheritance diagram for LinearDrivingForce:



### **Public Member Functions**

• LinearDrivingForce (const InputParameters &parameters)

### **Protected Member Functions**

- · virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

### **Private Attributes**

- · bool \_gaining
- Real \_coef
- Real \_driving\_value
- VariableValue & \_var

### 4.45.1 Detailed Description

Definition at line 11 of file LinearDrivingForce.h.

- 4.45.2 Constructor & Destructor Documentation
- 4.45.2.1 LinearDrivingForce::LinearDrivingForce ( const InputParameters & parameters )
- 4.45.3 Member Function Documentation
- 4.45.3.1 virtual Real Linear Driving Force::compute QpJacobian ( ) [protected], [virtual]
- 4.45.3.2 virtual Real LinearDrivingForce::computeQpResidual( ) [protected], [virtual]
- 4.45.4 Member Data Documentation
- **4.45.4.1 Real Linear Driving Force::\_coef** [private]

Definition at line 22 of file LinearDrivingForce.h.

**4.45.4.2 Real Linear Driving Force::\_driving\_value** [private]

Definition at line 23 of file LinearDrivingForce.h.

**4.45.4.3** bool Linear Driving Force::\_gaining [private]

Definition at line 21 of file LinearDrivingForce.h.

**4.45.4.4 VariableValue& LinearDrivingForce::\_var** [private]

Definition at line 24 of file LinearDrivingForce.h.

The documentation for this class was generated from the following file:

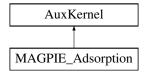
· LinearDrivingForce.h

## 4.46 MAGPIE\_Adsorption Class Reference

Magpie Adsorption class inherits from AuxKernel.

```
#include <MAGPIE_Adsorption.h>
```

Inheritance diagram for MAGPIE\_Adsorption:



#### **Public Member Functions**

• MAGPIE\_Adsorption (const InputParameters &parameters) Standard MOOSE public constructor.

### **Protected Member Functions**

virtual Real computeValue ()
 Required MOOSE function override.

## **Private Attributes**

· unsigned int index

Index of the gaseous species to calculate equilibria for.

· const MaterialProperty

< MAGPIE\_DATA > & \_magpie\_dat

Material Property holding the MAGPIE data structure.

## 4.46.1 Detailed Description

Magpie Adsorption class inherits from AuxKernel.

This class object creates an AuxKernel for use in the MOOSE framework. The AuxKernel will calculate the adsorption equilibria for a given species in the gas phase based on parameters, variables, and constants set in the MAGPIE object. Those values include temperature, pressure, concentration, and associated equilibrium energy constants. The return value is the adsorption equilibrium value in mol/kg.

Definition at line 63 of file MAGPIE\_Adsorption.h.

### 4.46.2 Constructor & Destructor Documentation

4.46.2.1 MAGPIE\_Adsorption::MAGPIE\_Adsorption ( const InputParameters & parameters )

Standard MOOSE public constructor.

#### 4.46.3 Member Function Documentation

4.46.3.1 virtual Real MAGPIE\_Adsorption::computeValue() [protected], [virtual]

Required MOOSE function override.

This is the function that is called by the MOOSE framework when a calculation of the AuxVariable is needed. You are required to override this function for any inherited AuxKernel.

#### 4.46.4 Member Data Documentation

**4.46.4.1 unsigned int MAGPIE\_Adsorption::\_index** [private]

Index of the gaseous species to calculate equilibria for.

Definition at line 76 of file MAGPIE\_Adsorption.h.

4.46.4.2 const MaterialProperty < MAGPIE\_DATA > & MAGPIE\_Adsorption::\_magpie\_dat [private]

Material Property holding the MAGPIE data structure.

Definition at line 77 of file MAGPIE Adsorption.h.

The documentation for this class was generated from the following file:

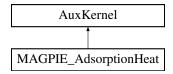
· MAGPIE Adsorption.h

## 4.47 MAGPIE\_AdsorptionHeat Class Reference

Magpie Adsorption Heat class inherits from AuxKernel.

#include <MAGPIE\_AdsorptionHeat.h>

Inheritance diagram for MAGPIE\_AdsorptionHeat:



## **Public Member Functions**

MAGPIE\_AdsorptionHeat (const InputParameters &parameters)
 Standard MOOSE public constructor.

### **Protected Member Functions**

virtual Real computeValue ()
 Required MOOSE function override.

## **Private Attributes**

unsigned int \_index

Index of the gaseous species to calculate adsorbtion heat for.

const MaterialProperty

< MAGPIE\_DATA > & \_magpie\_dat

Material Property holding the MAGPIE data structure.

VariableValue & \_solid\_conc

Reference to the adsorbed amount of the given species (AuxVariable)

### 4.47.1 Detailed Description

Magpie Adsorption Heat class inherits from AuxKernel.

This class object creates an AuxKernel for use in the MOOSE framework. The AuxKernel will calculate the heat of adsorption for a given species in the gas phase based on parameters, variables, and constants set in the MAGPIE object. Those values include temperature, pressure, concentration, and associated equilibrium energy constants. The return value is the heat of adsorption value in J/kg.

Definition at line 63 of file MAGPIE\_AdsorptionHeat.h.

#### 4.47.2 Constructor & Destructor Documentation

## 4.47.2.1 MAGPIE\_AdsorptionHeat::MAGPIE\_AdsorptionHeat ( const InputParameters & parameters )

Standard MOOSE public constructor.

#### 4.47.3 Member Function Documentation

```
4.47.3.1 virtual Real MAGPIE_AdsorptionHeat::computeValue() [protected], [virtual]
```

Required MOOSE function override.

This is the function that is called by the MOOSE framework when a calculation of the AuxVariable is needed. You are required to override this function for any inherited AuxKernel.

## 4.47.4 Member Data Documentation

```
4.47.4.1 unsigned int MAGPIE_AdsorptionHeat::_index [private]
```

Index of the gaseous species to calculate adsorbtion heat for.

Definition at line 76 of file MAGPIE\_AdsorptionHeat.h.

```
4.47.4.2 const MaterialProperty < MAGPIE_DATA > & MAGPIE_AdsorptionHeat::_magpie_dat [private]
```

Material Property holding the MAGPIE data structure.

Definition at line 77 of file MAGPIE\_AdsorptionHeat.h.

```
4.47.4.3 VariableValue& MAGPIE_AdsorptionHeat::_solid_conc [private]
```

Reference to the adsorbed amount of the given species (AuxVariable)

Definition at line 78 of file MAGPIE AdsorptionHeat.h.

The documentation for this class was generated from the following file:

• MAGPIE\_AdsorptionHeat.h

# 4.48 MAGPIE\_DATA Struct Reference

MAGPIE Data Structure.

#include <magpie.h>

#### **Public Attributes**

- std::vector< GSTA\_DATA > gsta\_dat
- std::vector< mSPD\_DATA > mspd\_dat
- std::vector< GPAST\_DATA > gpast\_dat
- SYSTEM\_DATA sys\_dat

### 4.48.1 Detailed Description

# MAGPIE Data Structure.

C-style object holding all information necessary to run a MAGPIE simulation. This is the data structure that will be used in other sub-routines when a mixed gas adsorption simulation needs to be run.

Definition at line 164 of file magpie.h.

#### 4.48.2 Member Data Documentation

4.48.2.1 std::vector<GPAST\_DATA> MAGPIE\_DATA::gpast\_dat

Definition at line 168 of file magpie.h.

4.48.2.2 std::vector < GSTA\_DATA > MAGPIE\_DATA::gsta\_dat

Definition at line 166 of file magpie.h.

4.48.2.3 std::vector<mSPD\_DATA> MAGPIE\_DATA::mspd\_dat

Definition at line 167 of file magpie.h.

## 4.48.2.4 SYSTEM\_DATA MAGPIE\_DATA::sys\_dat

Definition at line 169 of file magpie.h.

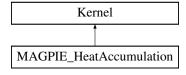
The documentation for this struct was generated from the following file:

· magpie.h

## 4.49 MAGPIE HeatAccumulation Class Reference

#include <MAGPIE\_HeatAccumulation.h>

Inheritance diagram for MAGPIE\_HeatAccumulation:



## **Public Member Functions**

• MAGPIE\_HeatAccumulation (const InputParameters &parameters)

#### **Protected Member Functions**

- virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

### **Private Attributes**

- const MaterialProperty < Real > & porosity
- const MaterialProperty< Real > & \_pellet\_density
- std::vector< VariableValue \* > \_solid\_heat
- std::vector< VariableValue \*> solid heat old

#### 4.49.1 Detailed Description

Definition at line 19 of file MAGPIE HeatAccumulation.h.

- 4.49.2 Constructor & Destructor Documentation
- 4.49.2.1 MAGPIE\_HeatAccumulation::MAGPIE\_HeatAccumulation ( const InputParameters & parameters )
- 4.49.3 Member Function Documentation
- 4.49.3.1 virtual Real MAGPIE\_HeatAccumulation::computeQpJacobian() [protected], [virtual]
- 4.49.3.2 virtual Real MAGPIE\_HeatAccumulation::computeQpResidual() [protected], [virtual]
- 4.49.4 Member Data Documentation
- 4.49.4.1 const MaterialProperty < Real > & MAGPIE\_HeatAccumulation::\_pellet\_density [private]

Definition at line 31 of file MAGPIE HeatAccumulation.h.

**4.49.4.2 const MaterialProperty** < Real > & MAGPIE\_HeatAccumulation::\_porosity [private]

Definition at line 30 of file MAGPIE HeatAccumulation.h.

4.49.4.3 std::vector<VariableValue \*> MAGPIE\_HeatAccumulation::\_solid\_heat [private]

Definition at line 32 of file MAGPIE HeatAccumulation.h.

**4.49.4.4** std::vector<VariableValue \*> MAGPIE\_HeatAccumulation::\_solid\_heat\_old [private]

Definition at line 33 of file MAGPIE\_HeatAccumulation.h.

The documentation for this class was generated from the following file:

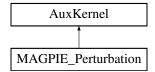
• MAGPIE\_HeatAccumulation.h

## 4.50 MAGPIE\_Perturbation Class Reference

Magpie Perturbation class inherits from AuxKernel.

#include <MAGPIE\_Perturbation.h>

Inheritance diagram for MAGPIE Perturbation:



#### **Public Member Functions**

MAGPIE\_Perturbation (const InputParameters &parameters)
 Standard MOOSE public constructor.

#### **Protected Member Functions**

virtual Real computeValue ()

Required MOOSE function override.

#### **Private Attributes**

· unsigned int index

Index of the gaseous species to calculate equilibria for.

· const MaterialProperty

```
< MAGPIE_DATA > & _magpie_dat
```

Material Property holding the MAGPIE data structure.

### 4.50.1 Detailed Description

Magpie Perturbation class inherits from AuxKernel.

This class object creates an AuxKernel for use in the MOOSE framework. The AuxKernel will calculate the perturbed equilibria for a given species in the gas phase based on parameters, variables, and constants set in the MAGPIE object. Those values include temperature, pressure, concentration, and associated equilibrium energy constants. The return value is the adsorption perturbation value in mol/kg.

Definition at line 72 of file MAGPIE\_Perturbation.h.

4.50.2 Constructor & Destructor Documentation

4.50.2.1 MAGPIE\_Perturbation::MAGPIE\_Perturbation ( const InputParameters & parameters )

Standard MOOSE public constructor.

4.50.3 Member Function Documentation

4.50.3.1 virtual Real MAGPIE\_Perturbation::computeValue() [protected], [virtual]

Required MOOSE function override.

This is the function that is called by the MOOSE framework when a calculation of the AuxVariable is needed. You are required to override this function for any inherited AuxKernel.

4.50.4 Member Data Documentation

**4.50.4.1** unsigned int MAGPIE\_Perturbation::\_index [private]

Index of the gaseous species to calculate equilibria for.

Definition at line 85 of file MAGPIE\_Perturbation.h.

4.50.4.2 const MaterialProperty < MAGPIE\_DATA > & MAGPIE\_Perturbation::\_magpie\_dat [private]

Material Property holding the MAGPIE data structure.

Definition at line 86 of file MAGPIE Perturbation.h.

The documentation for this class was generated from the following file:

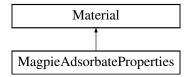
• MAGPIE\_Perturbation.h

## 4.51 MagpieAdsorbateProperties Class Reference

MagpieAdsorbateProperties class object inherits from Material object.

#include <MagpieAdsorbateProperties.h>

Inheritance diagram for MagpieAdsorbateProperties:



#### **Public Member Functions**

• MagpieAdsorbateProperties (const InputParameters &parameters)

Required constructor for objects in MOOSE.

## **Protected Member Functions**

virtual void computeQpProperties ()

Required function override for Material objects in MOOSE.

## **Private Attributes**

std::vector< unsigned int > \_index

Indices for the gas species in the system.

• VariableValue & \_temperature

Reference to the coupled column temperature.

• VariableValue & \_total\_pressure

Reference to the coupled column pressure.

std::vector< VariableValue \* > \_gas\_conc

Pointer list to the coupled gases.

• std::vector< int > num sites

List of the number of sites each gas species' isotherm contains.

std::vector< Real > \_max\_capacity

List of the maximum adsorption capacities of each gas species.

• std::vector< Real > \_molar\_volume

List of the van der Waal's molar volumes of each species.

std::vector< Real > \_enthalpy\_1

List of the site 1 enthalpies for each gas species.

std::vector< Real > \_enthalpy\_2

List of the site 2 enthalpies for each gas species.

```
std::vector< Real > _enthalpy_3
     List of the site 3 enthalpies for each gas species.

    std::vector< Real > enthalpy 4

     List of the site 4 enthalpies for each gas species.
std::vector< Real > _enthalpy_5
     List of the site 5 enthalpies for each gas species.

    std::vector< Real > enthalpy 6

      List of the site 6 enthalpies for each gas species.
std::vector< Real > _entropy_1
      List of the site 1 entropies for each gas species.
std::vector< Real > _entropy_2
      List of the site 2 entropies for each gas species.
std::vector< Real > _entropy_3
      List of the site 3 entropies for each gas species.
std::vector< Real > _entropy_4
     List of the site 4 entropies for each gas species.
std::vector< Real > _entropy_5
      List of the site 5 entropies for each gas species.
std::vector< Real > entropy 6
      List of the site 6 entropies for each gas species.

    MaterialProperty < MAGPIE_DATA > & _magpie_dat

      MaterialProperty object to hold the MAGPIE_DATA structure and all relavent information.
```

#### 4.51.1 Detailed Description

MagpieAdsorbateProperties class object inherits from Material object.

This class object inherits from the Material object in the MOOSE framework. All public and protected members of this class are required function overrides. The object will set up the MAGPIE\_DATA structure (see magpie.h) based on user provied input from the input file. That information will be used to estimate the adsorption of each species in the system based on temperature, pressure, and concentrations of each species.

Note

The GSTA isotherm model for each species allows upto 6 energetically distinct adsorption sites. If those sites are not used by a particular species, then those energies should be left as zeros in the input files and the number of relavent sites for each species needs to be recorded in the input file. Each species is allowed to have a different number of adsorption sites in a particular adsorbent.

Definition at line 62 of file MagpieAdsorbateProperties.h.

```
4.51.2 Constructor & Destructor Documentation
```

4.51.2.1 MagpieAdsorbateProperties::MagpieAdsorbateProperties ( const InputParameters & parameters )

Required constructor for objects in MOOSE.

```
4.51.3 Member Function Documentation
```

4.51.3.1 virtual void MagpieAdsorbateProperties::computeQpProperties() [protected], [virtual]

Required function override for Material objects in MOOSE.

This function computes the material properties when they are needed by other MOOSE objects.

```
4.51.4 Member Data Documentation
```

**4.51.4.1** std::vector<Real> MagpieAdsorbateProperties::\_enthalpy\_1 [private]

List of the site 1 enthalpies for each gas species.

Definition at line 84 of file MagpieAdsorbateProperties.h.

**4.51.4.2** std::vector<Real> MagpieAdsorbateProperties::\_enthalpy\_2 [private]

List of the site 2 enthalpies for each gas species.

Definition at line 85 of file MagpieAdsorbateProperties.h.

**4.51.4.3** std::vector<Real> MagpieAdsorbateProperties::\_enthalpy\_3 [private]

List of the site 3 enthalpies for each gas species.

Definition at line 86 of file MagpieAdsorbateProperties.h.

**4.51.4.4 std::vector**<**Real**> **MagpieAdsorbateProperties::\_enthalpy\_4** [private]

List of the site 4 enthalpies for each gas species.

Definition at line 87 of file MagpieAdsorbateProperties.h.

**4.51.4.5 std::vector**<**Real**> **MagpieAdsorbateProperties::\_enthalpy\_5** [private]

List of the site 5 enthalpies for each gas species.

Definition at line 88 of file MagpieAdsorbateProperties.h.

**4.51.4.6** std::vector<Real> MagpieAdsorbateProperties::\_enthalpy\_6 [private]

List of the site 6 enthalpies for each gas species.

Definition at line 89 of file MagpieAdsorbateProperties.h.

**4.51.4.7** std::vector<Real> MagpieAdsorbateProperties::\_entropy\_1 [private]

List of the site 1 entropies for each gas species.

Definition at line 91 of file MagpieAdsorbateProperties.h.

4.51.4.8 std::vector<Real> MagpieAdsorbateProperties::\_entropy\_2 [private]

List of the site 2 entropies for each gas species.

Definition at line 92 of file MagpieAdsorbateProperties.h.

**4.51.4.9** std::vector<Real> MagpieAdsorbateProperties::\_entropy\_3 [private]

List of the site 3 entropies for each gas species.

Definition at line 93 of file MagpieAdsorbateProperties.h.

 $\textbf{4.51.4.10} \quad \textbf{std::vector} < \textbf{Real} > \textbf{MagpieAdsorbateProperties::\_entropy\_4} \quad \texttt{[private]}$ 

List of the site 4 entropies for each gas species.

Definition at line 94 of file MagpieAdsorbateProperties.h.

**4.51.4.11** std::vector<Real> MagpieAdsorbateProperties::\_entropy\_5 [private]

List of the site 5 entropies for each gas species.

Definition at line 95 of file MagpieAdsorbateProperties.h.

4.51.4.12 std::vector<Real> MagpieAdsorbateProperties::\_entropy\_6 [private]

List of the site 6 entropies for each gas species.

Definition at line 96 of file MagpieAdsorbateProperties.h.

4.51.4.13 std::vector<VariableValue \*> MagpieAdsorbateProperties::\_gas\_conc [private]

Pointer list to the coupled gases.

Definition at line 78 of file MagpieAdsorbateProperties.h.

**4.51.4.14** std::vector<unsigned int> MagpieAdsorbateProperties::\_index [private]

Indices for the gas species in the system.

Definition at line 75 of file MagpieAdsorbateProperties.h.

4.51.4.15 MaterialProperty < MAGPIE\_DATA > & MagpieAdsorbateProperties::\_magpie\_dat [private]

MaterialProperty object to hold the MAGPIE\_DATA structure and all relavent information.

This is the object that needs to interface with the MAGPIE functions in order to solve for variable information such as adsorption capacities, mixed gas adsorption equilibria, and heats of adsorption.

Definition at line 102 of file MagpieAdsorbateProperties.h.

**4.51.4.16** std::vector<Real> MagpieAdsorbateProperties::\_max\_capacity [private]

List of the maximum adsorption capacities of each gas species.

Definition at line 81 of file MagpieAdsorbateProperties.h.

**4.51.4.17** std::vector<Real> MagpieAdsorbateProperties::\_molar\_volume [private]

List of the van der Waal's molar volumes of each species.

Definition at line 82 of file MagpieAdsorbateProperties.h.

**4.51.4.18** std::vector<int> MagpieAdsorbateProperties::\_num\_sites [private]

List of the number of sites each gas species' isotherm contains.

Definition at line 80 of file MagpieAdsorbateProperties.h.

4.51.4.19 VariableValue& MagpieAdsorbateProperties::\_temperature [private]

Reference to the coupled column temperature.

Definition at line 76 of file MagpieAdsorbateProperties.h.

4.51.4.20 VariableValue& MagpieAdsorbateProperties::\_total\_pressure [private]

Reference to the coupled column pressure.

Definition at line 77 of file MagpieAdsorbateProperties.h.

The documentation for this class was generated from the following file:

• MagpieAdsorbateProperties.h

## 4.52 Matrix < T > Class Template Reference

Templated C++ Matrix Class Object (click Matrix to go to function definitions)

#include <macaw.h>

#### **Public Member Functions**

· Matrix (int rows, int columns)

Constructor for matrix with given number of rows and columns.

T & operator() (int i, int j)

Access operator for the matrix element at row i and column j (e.g., aij = A(i,j))

• T operator() (int i, int j) const

Constant access operator for the the matrix element at row i and column j.

Matrix (const Matrix &M)

Copy constructor for constructing a matrix as a copy of another matrix.

Matrix & operator= (const Matrix &M)

Equals operator for setting one matrix equal to another matrix.

Matrix ()

Default constructor for creating an empty matrix.

• ~Matrix ()

Default destructor for clearing out memory.

void set\_size (int i, int j)

Function to set/change the size of a matrix to i rows and j columns.

• void zeros ()

Function to set/change all values in a matrix to zeros.

void edit (int i, int j, T value)

Function to set/change the element of a matrix at row i and column j to given value.

• int rows ()

Function to return the number of rows in a given matrix.

• int columns ()

Function to return the number of columns in a matrix.

• T determinate ()

Function to compute the determinate of a matrix and return that value.

• T norm ()

Function to compute the L2-norm of a matrix and return that value.

• T sum ()

Function to compute the sum of all elements in a matrix and return that value.

T inner\_product (const Matrix &x)

Function to compute the inner product between this matrix and matrix x.

Matrix & cofactor (const Matrix &M)

Function to convert this matrix to a cofactor matrix of the given matrix M.

Matrix operator+ (const Matrix &M)

Operator to add this matrix and matrix M and return the new matrix result.

• Matrix operator- (const Matrix &M)

Operator to subtract this matrix and matrix M and return the new matrix result.

Matrix operator\* (const T)

Operator to multiply this matrix by a scalar T return the new matrix result.

Matrix operator/ (const T)

Operator to divide this matrix by a scalar T and return the new matrix result.

Matrix operator\* (const Matrix &M)

Operator to multiply this matrix and matrix M and return the new matrix result.

Matrix & transpose (const Matrix &M)

Function to convert this matrix to the transpose of the given matrix M.

Matrix & transpose\_multiply (const Matrix &MT, const Matrix &v)

Function to convert this matrix into the result of the given matrix M transposed and multiplied by the other given matrix

Matrix & adjoint (const Matrix &M)

Function to convert this matrix to the adjoint of the given matrix.

Matrix & inverse (const Matrix &M)

Function to convert this matrix to the inverse of the given matrix.

void Display (const std::string Name)

Function to display the contents of this matrix given a Name for the matrix.

Matrix & tridiagonalSolve (const Matrix &A, const Matrix &b)

Function to solve Ax=b for x if A is symmetric, tridiagonal (this->x)

Matrix & ladshawSolve (const Matrix &A, const Matrix &d)

Function to solve Ax=d for x if A is non-symmetric, tridiagonal (this->x)

Matrix & tridiagonalFill (const T A, const T B, const T C, bool Spherical)

Function to fill in this matrix with coefficients A, B, and C to form a tridiagonal matrix.

Matrix & naturalLaplacian3D (int m)

Function to fill out this matrix with coefficients from a 3D Laplacian function.

Matrix & sphericalBCFill (int node, const T coeff, T variable)

Function to fill out a column matrix with spherical specific boundary conditions.

Matrix & ConstantICFill (const T IC)

Function to set all values in a column matrix to a given constant.

Matrix & SolnTransform (const Matrix &A, bool Forward)

Function to transform the values in a column matrix from cartesian to spherical coordinates.

• T sphericalAvg (double radius, double dr, double bound, bool Dirichlet)

Function to compute a spatial average of this column matrix in spherical coordinates.

• T IntegralAvg (double radius, double dr, double bound, bool Dirichlet)

Function to compute a spatial average of this column matrix in spherical coordinates.

• T IntegralTotal (double dr, double bound, bool Dirichlet)

Function to compute a spatial total of this column matrix in spherical coordinates.

Matrix & tridiagonalVectorFill (const std::vector< T > &A, const std::vector< T > &B, const std::vector< T > &C)

Function to fill in this matrix, in tridiagonal fashion, using the vectors of coefficients.

Matrix & columnVectorFill (const std::vector< T > &A)

Function to fill in a column matrix with the values of the given vector object.

Matrix & columnProjection (const Matrix &b, const Matrix &b\_old, const double dt, const double dt\_old)

Function to project a column matrix solution in time based on older state vectors.

Matrix & dirichletBCFill (int node, const T coeff, T variable)

Function to fill in a column matrix with all zeros except at the given node.

Matrix & diagonalSolve (const Matrix &D, const Matrix &v)

Function to solve the system Dx=v for x given that D is diagonal (this->x)

Matrix & upperTriangularSolve (const Matrix &U, const Matrix &v)

Function to solve the system Ux=v for x given that U is upper Triagular (this->x)

Matrix & lowerTriangularSolve (const Matrix &L, const Matrix &v)

Function to solve the system Lx=v for x given that L is lower Triagular (this->x)

Matrix & upperHessenberg2Triangular (Matrix &b)

Function to convert this square matrix to upper Triangular (assuming this is upper Hessenberg)

Matrix & lowerHessenberg2Triangular (Matrix &b)

Function to convert this square matrix to lower Triangular (assuming this is lower Hessenberg)

Matrix & upperHessenbergSolve (const Matrix &H, const Matrix &v)

Function to solve the system Hx=v for x given that H is upper Hessenberg (this->x)

Matrix & lowerHessenbergSolve (const Matrix &H, const Matrix &v)

Function to solve the system Hx=v for x given that H is lower Hessenberg (this->x)

• Matrix & columnExtract (int j, const Matrix &M)

Function to set this column matrix to the jth column of the given matrix M.

Matrix & rowExtract (int i, const Matrix &M)

Function to set this row matrix to the ith row of the given matrix M.

Matrix & columnReplace (int j, const Matrix &v)

Function to this matrices' jth column with the given column matrix v.

Matrix & rowReplace (int i, const Matrix &v)

Function to this matrices' ith row with the given row matrix v.

· void rowShrink ()

Function to delete the last row of this matrix.

void columnShrink ()

Function to delete the last column of this matrix.

void rowExtend (const Matrix &v)

Function to add the row matrix v to the end of this matrix.

void columnExtend (const Matrix &v)

Function to add the column matrix v to the end of this matrix.

#### **Protected Attributes**

· int num rows

Number of rows of the matrix.

int num\_cols

Number of columns of the matrix.

std::vector< T > Data

Storage vector for the elements of the matrix.

## 4.52.1 Detailed Description

```
template < class T> class Matrix < T>
```

Templated C++ Matrix Class Object (click Matrix to go to function definitions)

C++ templated class object containing many different functions, actions, and solver routines associated with Dense Matrices. Operator overloads are also provided to give the user a more natural way of operating matrices on other matrices or scalars. These operator overloads are especially useful for reducing the amount of code needed to be written when working with matrix-based problems.

Definition at line 53 of file macaw.h.

## 4.52.2 Constructor & Destructor Documentation

```
4.52.2.1 template < class T > Matrix < T >::Matrix ( int rows, int columns )
```

Constructor for matrix with given number of rows and columns.

Definition at line 208 of file macaw.h.

```
4.52.2.2 template < class T > Matrix < T >::Matrix ( const Matrix < T > & \it M )
```

Copy constructor for constructing a matrix as a copy of another matrix.

Definition at line 247 of file macaw.h.

References Matrix< T >::Data, Matrix< T >::num\_cols, and Matrix< T >::num\_rows.

```
4.52.2.3 template < class T > Matrix < T >::Matrix ( )
```

Default constructor for creating an empty matrix.

Definition at line 292 of file macaw.h.

References Matrix< T >::num\_cols, and Matrix< T >::num\_rows.

```
4.52.2.4 template < class T > Matrix < T >::\sim Matrix ( )
```

Default destructor for clearing out memory.

Definition at line 302 of file macaw.h.

#### 4.52.3 Member Function Documentation

```
4.52.3.1 template < class T > Matrix < T > & Matrix < T > ::adjoint (const Matrix < T > & M )
```

Function to convert this matrix to the adjoint of the given matrix.

Definition at line 734 of file macaw.h.

References arg\_matrix\_same, mError, non\_square\_matrix, Matrix< T >::num\_cols, and Matrix< T >::num\_rows.

```
4.52.3.2 template < class T > Matrix < T > & Matrix < T > ::cofactor (const Matrix < T > & M
```

Function to convert this matrix to a cofactor matrix of the given matrix M.

Definition at line 489 of file macaw.h.

References arg\_matrix\_same, Matrix< T >::Data, Matrix< T >::determinate(), mError, non\_square\_matrix, Matrix< T >::num cols, and Matrix< T >::num rows.

```
4.52.3.3 template < class T > void Matrix < T > ::columnExtend ( const Matrix < T > & \nu )
```

Function to add the column matrix v to the end of this matrix.

Definition at line 1774 of file macaw.h.

References matvec\_mis\_match, mError, Matrix< T >::num\_cols, and Matrix< T >::num\_rows.

```
4.52.3.4 template < class T > Matrix < T > & Matrix < T >::columnExtract ( int j, const Matrix < T > & M)
```

Function to set this column matrix to the jth column of the given matrix M.

Definition at line 1644 of file macaw.h.

References arg matrix same, mError, and Matrix< T >::num rows.

4.52.3.5 template < class T > Matrix < T > & Matrix < T > ::columnProjection ( const Matrix < T > & b, const Matrix < T > & b, const double dt, const double dt\_old)

Function to project a column matrix solution in time based on older state vectors.

This function is used in finch.h to form Matrix u\_star. It uses the size of the current step and old step, dt and dt\_old respectively, to form an approximation for the next state. The current state and olde state of the variables are passed as b and b\_old respectively.

Definition at line 1344 of file macaw.h.

 $References\ arg\_matrix\_same,\ dim\_mis\_match,\ mError,\ Matrix< T>::num\_cols,\ and\ Matrix< T>::num\_rows.$ 

```
4.52.3.6 template < class T > Matrix < T > & Matrix < T >::columnReplace ( int j, const Matrix < T > & v)
```

Function to this matrices' jth column with the given column matrix v.

Definition at line 1686 of file macaw.h.

References arg\_matrix\_same, matvec\_mis\_match, mError, and Matrix< T >::num\_rows.

```
4.52.3.7 template < class T > int Matrix < T >::columns ( )
```

Function to return the number of columns in a matrix.

Definition at line 351 of file macaw.h.

```
4.52.3.8 template < class T > void Matrix < T >::columnShrink ( )
```

Function to delete the last column of this matrix.

Definition at line 1741 of file macaw.h.

References Matrix< T >::Data, Matrix< T >::num\_cols, and Matrix< T >::num\_rows.

```
4.52.3.9 template < class T > Matrix < T > & Matrix < T > ::columnVectorFill ( const std::vector < T > & A )
```

Function to fill in a column matrix with the values of the given vector object.

Definition at line 1322 of file macaw.h.

References dim\_mis\_match, matvec\_mis\_match, and mError.

```
4.52.3.10 template < class T > Matrix < T > & Matrix < T >::ConstantICFill (const T /C)
```

Function to set all values in a column matrix to a given constant.

Definition at line 1134 of file macaw.h.

References dim\_mis\_match, and mError.

```
4.52.3.11 template < class T > T Matrix < T >::determinate ( )
```

Function to compute the determinate of a matrix and return that value.

Definition at line 358 of file macaw.h.

References Matrix< T >::Data, Matrix< T >::determinate(), mError, non\_square\_matrix, Matrix< T >::num\_cols, and Matrix< T >::num\_rows.

Referenced by Matrix < T >:::cofactor(), Matrix < T >::determinate(), and Matrix < T >::inverse().

```
4.52.3.12 template < class T > Matrix < T > & Matrix < T > ::diagonal Solve ( const Matrix < T > & D, const Matrix < T > & \nu )
```

Function to solve the system Dx=v for x given that D is diagonal (this->x)

Definition at line 1395 of file macaw.h.

References arg\_matrix\_same, dim\_mis\_match, mError, Matrix< T >::num\_cols, Matrix< T >::num\_rows, and singular\_matrix.

```
4.52.3.13 template < class T > Matrix < T > & Matrix < T > ::dirichlet BCFill (int node, const T coeff, T variable)
```

Function to fill in a column matrix with all zeros except at the given node.

Similar to sphericalBCFill, this function will set the values of all elements in the column matrix to zero except at the given node, where the value is set to the product of coeff and variable. This is often used to set BCs in finch.h or other related files/simulations.

Definition at line 1369 of file macaw.h.

References dim\_mis\_match, and mError.

```
4.52.3.14 template < class T > void Matrix < T >::Display ( const std::string Name )
```

Function to display the contents of this matrix given a Name for the matrix.

Definition at line 782 of file macaw.h.

References empty matrix, and mError.

4.52.3.15 template < class T > void Matrix < T > ::edit ( int i, int j, T value )

Function to set/change the element of a matrix at row i and column j to given value.

Definition at line 332 of file macaw.h.

References mError, Matrix< T >::operator()(), and out\_of\_bounds.

Referenced by Matrix< T >::lowerHessenberg2Triangular(), Matrix< T >::operator\*(), and Matrix< T >::upper-Hessenberg2Triangular().

4.52.3.16 template < class T > T Matrix < T >::inner\_product ( const Matrix < T > & x )

Function to compute the inner product between this matrix and matrix x.

Definition at line 463 of file macaw.h.

References dim\_mis\_match, mError, Matrix< T >::num\_cols, and Matrix< T >::num\_rows.

4.52.3.17 template < class T > T Matrix < T >::Integral Avg ( double radius, double dr, double bound, bool Dirichlet )

Function to compute a spatial average of this column matrix in spherical coordinates.

This function is used to compute an average value of a variable, represented in this column matrix, by integrating over the domain of the sphere. (Assumes you DO NOT have variable value at center node)

#### **Parameters**

ſ	radius	radius of the sphere
ſ	dr	space between each node
ſ	bound	value of the variable at the boundary
ſ	Dirichlet	True if problem has a Dirichlet BC, False if Neumann

Definition at line 1182 of file macaw.h.

References dim\_mis\_match, mError, and qo().

4.52.3.18 template < class T > T Matrix < T > ::Integral Total (double dr, double bound, bool Dirichlet)

Function to compute a spatial total of this column matrix in spherical coordinates.

This function is used to compute an average value of a variable, represented in this column matrix, by integrating over the domain of the sphere. (Assumes you DO NOT have variable value at center node)

## **Parameters**

dr	space between each node
bound	value of the variable at the boundary
Dirichlet	True if problem has a Dirichlet BC, False if Neumann

Definition at line 1242 of file macaw.h.

References dim\_mis\_match, M\_PI, mError, and qo().

4.52.3.19 template < class T > Matrix < T > & Matrix < T >::inverse ( const Matrix < T > & M )

Function to convert this matrix to the inverse of the given matrix.

Definition at line 756 of file macaw.h.

References A, arg\_matrix\_same, Matrix< T >::Data, Matrix< T >::determinate(), mError, non\_square\_matrix, Matrix< T >::num\_cols, and Matrix< T >::num\_rows.

4.52.3.20 template < class T > Matrix < T > & Matrix < T > ::ladshawSolve ( const Matrix < T > & A, const Matrix < T > & d)

Function to solve Ax=d for x if A is non-symmetric, tridiagonal (this->x)

Definition at line 876 of file macaw.h.

References A, arg\_matrix\_same, dim\_mis\_match, mError, Matrix< T >::num\_cols, Matrix< T >::num\_rows, singular matrix, and unstable matrix.

4.52.3.21 template < class T > Matrix < T > & Matrix < T > :: lowerHessenberg2Triangular ( Matrix < T > & b )

Function to convert this square matrix to lower Triangular (assuming this is lower Hessenberg)

During this transformation, a column vector (b) is also being transformed to represent the BCs in a linear system. This algorithm uses Givens Rotations to efficiently convert the lower Hessenberg matrix to an lower triangular matrix.

Definition at line 1561 of file macaw.h.

References arg\_matrix\_same, dim\_mis\_match, Matrix< T >::edit(), matrix\_too\_small, mError, Matrix< T >::num\_cols, Matrix< T >::num\_rows, and singular\_matrix.

4.52.3.22 template < class T > Matrix < T > & Matrix < T > ::lowerHessenbergSolve ( const Matrix < T > & H, const Matrix < T > &  $\nu$ )

Function to solve the system Hx=v for x given that H is lower Hessenberg (this->x)

Definition at line 1627 of file macaw.h.

References arg\_matrix\_same, Matrix< T >::Data, and mError.

4.52.3.23 template < class T > Matrix < T > & Matrix < T > ::lowerTriangularSolve ( const Matrix < T > & L, const Matrix < T > &  $\nu$ )

Function to solve the system Lx=v for x given that L is lower Triagular (this->x)

Definition at line 1471 of file macaw.h.

References arg\_matrix\_same, dim\_mis\_match, mError, Matrix< T >::num\_cols, Matrix< T >::num\_rows, and singular\_matrix.

4.52.3.24 template < class T > Matrix < T > & Matrix < T >::naturalLaplacian3D ( int m )

Function to fill out this matrix with coefficients from a 3D Laplacian function.

This function will fill out the coefficients of the matrix with the coefficients that stem from discretizing a 3D Laplacian on a natural grid with 2nd order finite differences.

Definition at line 1031 of file macaw.h.

```
4.52.3.25 template < class T > T Matrix < T >::norm ( )
```

Function to compute the L2-norm of a matrix and return that value.

Definition at line 427 of file macaw.h.

```
4.52.3.26 template < class T > T & Matrix < T >::operator() ( int i, int j )
```

Access operator for the matrix element at row i and column j (e.g., aij = A(i,j))

Definition at line 219 of file macaw.h.

References mError, and out\_of\_bounds.

Referenced by Matrix< T >::edit().

4.52.3.27 template < class T > T Matrix < T >::operator() ( int i, int j ) const

Constant access operator for the the matrix element at row i and column j.

Definition at line 232 of file macaw.h.

References mError, and out of bounds.

4.52.3.28 template < class T > Matrix < T > Matrix < T > ::operator\* ( const T a )

Operator to multiply this matrix by a scalar T return the new matrix result.

Definition at line 612 of file macaw.h.

References Matrix< T >::Data.

4.52.3.29 template < class T> Matrix < T> Matrix < T> :: operator \* ( const Matrix < T> & M )

Operator to multiply this matrix and matrix M and return the new matrix result.

Definition at line 642 of file macaw.h.

References Matrix < T >::Data, dim\_mis\_match, Matrix < T >::edit(), mError, Matrix < T >::num\_cols, and Matrix < T >::num\_rows.

4.52.3.30 template < class T > Matrix < T > Matrix < T > :: operator + ( const Matrix < T > & M )

Operator to add this matrix and matrix M and return the new matrix result.

Definition at line 566 of file macaw.h.

References Matrix < T >::Data, dim\_mis\_match, mError, Matrix < T >::num\_cols, and Matrix < T >::num\_rows.

4.52.3.31 template < class T > Matrix < T > Matrix < T >::operator-( const Matrix < T > & M )

Operator to subtract this matrix and matrix M and return the new matrix result.

Definition at line 589 of file macaw.h.

References Matrix < T >::Data, dim\_mis\_match, mError, Matrix < T >::num\_cols, and Matrix < T >::num\_rows.

4.52.3.32 template < class T> Matrix < T> Matrix < T>::operator/ (const Ta)

Operator to divide this matrix by a scalar T and return the new matrix result.

Definition at line 627 of file macaw.h.

References Matrix< T >::Data.

4.52.3.33 template < class T > Matrix < T > & Matrix < T > ::operator= ( const Matrix < T > & M )

Equals operator for setting one matrix equal to another matrix.

Definition at line 264 of file macaw.h.

References Matrix < T >::num\_cols, and Matrix < T >::num\_rows.

4.52.3.34 template < class T > void Matrix < T >::rowExtend ( const Matrix < T > &  $\nu$  )

Function to add the row matrix v to the end of this matrix.

Definition at line 1758 of file macaw.h.

References matvec\_mis\_match, mError, and Matrix< T >::num\_cols.

4.52.3.35 template < class T > Matrix < T > & Matrix < T >::rowExtract ( int i, const Matrix < T > & M )

Function to set this row matrix to the ith row of the given matrix M.

Definition at line 1665 of file macaw.h.

References arg\_matrix\_same, mError, and Matrix< T >::num\_cols.

4.52.3.36 template < class T > Matrix < T > & Matrix < T > ::rowReplace ( int i, const Matrix < T > & v )

Function to this matrices' ith row with the given row matrix v.

Definition at line 1707 of file macaw.h.

References arg\_matrix\_same, matvec\_mis\_match, mError, and Matrix< T >::num\_cols.

4.52.3.37 template < class T > int Matrix < T >::rows ( )

Function to return the number of rows in a given matrix.

Definition at line 344 of file macaw.h.

4.52.3.38 template < class T > void Matrix < T >::rowShrink ( )

Function to delete the last row of this matrix.

Definition at line 1728 of file macaw.h.

4.52.3.39 template < class T > void Matrix < T > ::set\_size ( int i, int j )

Function to set/change the size of a matrix to i rows and j columns.

Definition at line 309 of file macaw.h.

References invalid\_size, and mError.

4.52.3.40 template < class T > Matrix < T > & Matrix < T >::SolnTransform ( const Matrix < T > & A, bool Forward )

Function to transform the values in a column matrix from cartesian to spherical coordinates.

Definition at line 1153 of file macaw.h.

References arg\_matrix\_same, Matrix< T >::Data, dim\_mis\_match, mError, and Matrix< T >::num\_rows.

4.52.3.41 template < class T > T Matrix < T >::spherical Avg (double radius, double dr, double bound, bool Dirichlet)

Function to compute a spatial average of this column matrix in spherical coordinates.

This function is used to compute an average value of a variable, represented in this column matrix, by integrating over the domain of the sphere. (Assumes you have variable value at center node)

## Parameters

Ī	radius	radius of the sphere
	dr	space between each node
	bound	value of the variable at the boundary
ı	Dirichlet	True if problem has a Dirichlet BC, False if Neumann

Definition at line 1220 of file macaw.h.

References dim\_mis\_match, and mError.

4.52.3.42 template < class T > Matrix < T > & Matrix < T > ::spherical BCFill (int node, const T coeff, T variable)

Function to fill out a column matrix with spherical specific boundary conditions.

This function will fille out a column matrix with zeros at all nodes expect for the node indicated. That node's value will be the product of the node id with the coeff and variable values given.

Definition at line 1108 of file macaw.h.

References dim\_mis\_match, and mError.

```
4.52.3.43 template < class T > T Matrix < T >::sum ( )
```

Function to compute the sum of all elements in a matrix and return that value.

Definition at line 448 of file macaw.h.

```
4.52.3.44 template < class T > Matrix < T > & Matrix < T >::transpose (const Matrix < T > & M)
```

Function to convert this matrix to the transpose of the given matrix M.

Definition at line 676 of file macaw.h.

References arg\_matrix\_same, Matrix < T >::Data, mError, Matrix < T >::num\_cols, and Matrix < T >::num\_rows.

```
4.52.3.45 template < class T > Matrix < T > & Matrix < T > ::transpose_multiply ( const Matrix < T > & MT, const Matrix < T > & V)
```

Function to convert this matrix into the result of the given matrix M transposed and multiplied by the other given matrix v.

Definition at line 699 of file macaw.h.

References arg\_matrix\_same, Matrix< T >::Data, dim\_mis\_match, mError, Matrix< T >::num\_cols, and Matrix< T >::num rows.

```
4.52.3.46 template < class T> Matrix < T> & Matrix < T>::tridiagonal Fill (const TA, const TB, const TC, bool Spherical
```

Function to fill in this matrix with coefficients A, B, and C to form a tridiagonal matrix.

This function fills in the diagonal elements of a square matrix with coefficient B, upper diagonal with C, and lower diagonal with A. The boolean will apply a transformation to those coefficients, if the problem happens to stem from 1-D diffusion in spherical coordinates.

Definition at line 981 of file macaw.h.

References mError, and non\_square\_matrix.

```
4.52.3.47 template < class T > Matrix < T > & Matrix < T > ::tridiagonal Solve ( const Matrix < T > & A, const Matrix < T > & b )
```

Function to solve Ax=b for x if A is symmetric, tridiagonal (this->x)

Definition at line 807 of file macaw.h.

References A, arg\_matrix\_same, dim\_mis\_match, mError, Matrix< T >::num\_cols, and Matrix< T >::num\_rows.

```
4.52.3.48 template < class T > Matrix < T > & Matrix < T > ::tridiagonal Vector Fill ( const std::vector < T > & A, const std::vector < T > & B, const std::vector < T > & C )
```

Function to fill in this matrix, in tridiagonal fashion, using the vectors of coefficients.

Definition at line 1279 of file macaw.h.

References matvec\_mis\_match, mError, and non\_square\_matrix.

```
4.52.3.49 template < class T > Matrix < T > & Matrix < T >::upperHessenberg2Triangular (Matrix < T > & b)
```

Function to convert this square matrix to upper Triangular (assuming this is upper Hessenberg)

During this transformation, a column vector (b) is also being transformed to represent the BCs in a linear system. This algorithm uses Givens Rotations to efficiently convert the upper Hessenberg matrix to an upper triangular matrix.

Definition at line 1512 of file macaw.h.

References arg\_matrix\_same, dim\_mis\_match, Matrix < T >::edit(), matrix\_too\_small, mError, Matrix < T >::num\_rows, and singular\_matrix.

4.52.3.50 template < class T > Matrix < T > & Matrix < T > ::upperHessenbergSolve ( const Matrix < T > & H, const Matrix < T > &  $\nu$  )

Function to solve the system Hx=v for x given that H is upper Hessenberg (this->x)

Definition at line 1610 of file macaw.h.

References arg\_matrix\_same, Matrix< T >::Data, and mError.

4.52.3.51 template < class T > Matrix < T > & Matrix < T > ::upperTriangularSolve ( const Matrix < T > & U, const Matrix < T > & v)

Function to solve the system Ux=v for x given that U is upper Triagular (this->x)

Definition at line 1430 of file macaw.h.

References arg\_matrix\_same, dim\_mis\_match, mError, Matrix< T >::num\_cols, Matrix< T >::num\_rows, and singular\_matrix.

```
4.52.3.52 template < class T > void Matrix < T >::zeros ( )
```

Function to set/change all values in a matrix to zeros.

Definition at line 324 of file macaw.h.

4.52.4 Member Data Documentation

```
4.52.4.1 template < class T > std::vector < T > Matrix < T >::Data [protected]
```

Storage vector for the elements of the matrix.

Definition at line 203 of file macaw.h.

Referenced by Matrix < T >:::cofactor(), Matrix < T >::columnShrink(), Matrix < T >::determinate(), Matrix < T >::inverse(), Matrix < T >::operator\*(), Matrix < T >::operator\*(), Matrix < T >::operator\*(), Matrix < T >::operator\*(), Matrix < T >::transpose(), Matrix < T >::transpose(), Matrix < T >::upperHessenbergSolve().

```
4.52.4.2 template < class T > int Matrix < T >::num_cols [protected]
```

Number of columns of the matrix.

Definition at line 202 of file macaw.h.

Referenced by Matrix T>:::adjoint(), Matrix T>::cofactor(), Matrix T>::columnExtend(), Matrix T>::columnProjection(), Matrix T>::columnShrink(), Matrix T>::determinate(), Matrix T>::diagonalSolve(), Matrix  $T>::inner\_product()$ , Matrix T>::inverse(), Matrix T>::ladshawSolve(), Matrix T>::lower-Hessenberg2Triangular(), Matrix T>::lower-TriangularSolve(), Matrix T>::inverse(), Matrix T>:

```
4.52.4.3 template < class T > int Matrix < T >::num_rows [protected]
```

Number of rows of the matrix.

Definition at line 201 of file macaw.h.

Referenced by Matrix< T >::adjoint(), Matrix< T >::cofactor(), Matrix< T >::columnExtract(), Matrix< T >::columnExtract(), Matrix< T >::columnExtract(), Matrix< T >::columnProjection(), Matrix< T >::columnReplace(), Matrix< T >::columnShrink(), Matrix< T >::determinate(), Matrix< T >::diagonalSolve(), Matrix< T >::inner\_product(), Matrix< T >::inverse(), Matrix< T >::lowerTriangularSolve(), Matrix< T >::lowerTriangularSolve(), Matrix< T >::operator+(), Matrix< T >::operator-(), Matrix< T >::operator-(), Matrix< T >::operator-(), Matrix< T >::transpose\_multiply(), Matrix<

T >::tridiagonalSolve(), Matrix< T >::upperHessenberg2Triangular(), and Matrix< T >::upperTriangularSolve().

The documentation for this class was generated from the following file:

· macaw.h

#### 4.53 MIXED\_GAS Struct Reference

Data structure holding information necessary for computing mixed gas properties.

```
#include <egret.h>
```

#### **Public Attributes**

• int N

Given: Total number of gas species.

bool CheckMolefractions = true

Given: True = Check Molefractions for errors.

• double total\_pressure

Given: Total gas pressure (kPa)

· double gas\_temperature

Given: Gas temperature (K)

· double velocity

Given: Gas phase velocity (cm/s)

· double char length

Given: Characteristic Length (cm)

std::vector< double > molefraction

Given: Gas molefractions of each species (-)

· double total\_density

Calculated: Total gas density  $(g/cm^{\wedge}3)$  {use RE3}.

• double total\_dyn\_vis

Calculated: Total dynamic viscosity (g/cm/s)

· double kinematic\_viscosity

Calculated: Kinematic viscosity (cm<sup>2</sup>/s)

double total\_molecular\_weight

Calculated: Total molecular weight (g/mol)

double total\_specific\_heat

Calculated: Total specific heat (J/g/K)

double Reynolds

Calculated: Value of the Reynold's number (-)

Matrix< double > binary\_diffusion

Calculated: Tensor matrix of binary gas diffusivities (cm\^2/s)

std::vector< PURE\_GAS > species\_dat

Vector of the pure gas info of all species.

## 4.53.1 Detailed Description

Data structure holding information necessary for computing mixed gas properties.

C-style object holding the mixed gas information necessary for performing gas dynamic simulations. This object works in conjunction with the calculate\_variables function and uses the kinetic theory of gases to estimate mixed gas properties.

Definition at line 116 of file egret.h.

4.53.2 Member Data Documentation

4.53.2.1 Matrix<double> MIXED\_GAS::binary\_diffusion

Calculated: Tensor matrix of binary gas diffusivities (cm<sup>2</sup>/s)

Definition at line 136 of file egret.h.

4.53.2.2 double MIXED\_GAS::char\_length

Given: Characteristic Length (cm)

Definition at line 126 of file egret.h.

4.53.2.3 bool MIXED\_GAS::CheckMolefractions = true

Given: True = Check Molefractions for errors.

Definition at line 120 of file egret.h.

4.53.2.4 double MIXED\_GAS::gas\_temperature

Given: Gas temperature (K)

Definition at line 124 of file egret.h.

4.53.2.5 double MIXED\_GAS::kinematic\_viscosity

Calculated: Kinematic viscosity (cm<sup>2</sup>/s)

Definition at line 132 of file egret.h.

4.53.2.6 std::vector<double> MIXED\_GAS::molefraction

Given: Gas molefractions of each species (-)

Definition at line 127 of file egret.h.

4.53.2.7 int MIXED\_GAS::N

Given: Total number of gas species.

Definition at line 119 of file egret.h.

4.53.2.8 double MIXED\_GAS::Reynolds

Calculated: Value of the Reynold's number (-)

Definition at line 135 of file egret.h.

 $\textbf{4.53.2.9} \quad \textbf{std::vector} < \textbf{PURE\_GAS} > \textbf{MIXED\_GAS::species\_dat}$ 

Vector of the pure gas info of all species.

Definition at line 139 of file egret.h.

4.53.2.10 double MIXED\_GAS::total\_density

Calculated: Total gas density (g/cm<sup>3</sup>) {use RE3}.

Definition at line 130 of file egret.h.

4.53.2.11 double MIXED\_GAS::total\_dyn\_vis

Calculated: Total dynamic viscosity (g/cm/s)

Definition at line 131 of file egret.h.

4.53.2.12 double MIXED\_GAS::total\_molecular\_weight

Calculated: Total molecular weight (g/mol)

Definition at line 133 of file egret.h.

4.53.2.13 double MIXED\_GAS::total\_pressure

Given: Total gas pressure (kPa)

Definition at line 123 of file egret.h.

4.53.2.14 double MIXED\_GAS::total\_specific\_heat

Calculated: Total specific heat (J/g/K)

Definition at line 134 of file egret.h.

4.53.2.15 double MIXED\_GAS::velocity

Given: Gas phase velocity (cm/s)

Definition at line 125 of file egret.h.

The documentation for this struct was generated from the following file:

· egret.h

### 4.54 mSPD\_DATA Struct Reference

MSPD Data Structure.

#include <magpie.h>

## **Public Attributes**

• double s

Area shape factor.

double v

van der Waals Volume (cm<sup>\(\circ\)</sup>3/mol)

double eMax

Maximum lateral interaction energy (J/mol)

• std::vector< double > eta

Binary interaction parameter matrix (i,j)

· double gama

Activity coefficient calculated from mSPD.

# 4.54.1 Detailed Description

MSPD Data Structure.

C-Style object holding all parameter information associated with the Modified Spreading Pressure Dependent (SPD) activity model. Each species in the gas phase will have one of these objects.

Definition at line 110 of file magpie.h.

4.54.2 Member Data Documentation

4.54.2.1 double mSPD\_DATA::eMax

Maximum lateral interaction energy (J/mol)

Definition at line 114 of file magpie.h.

4.54.2.2 std::vector<double> mSPD\_DATA::eta

Binary interaction parameter matrix (i,j)

Definition at line 115 of file magpie.h.

4.54.2.3 double mSPD\_DATA::gama

Activity coefficient calculated from mSPD.

Definition at line 116 of file magpie.h.

4.54.2.4 double mSPD\_DATA::s

Area shape factor.

Definition at line 112 of file magpie.h.

4.54.2.5 double mSPD\_DATA::v

van der Waals Volume (cm<sup>3</sup>/mol)

Definition at line 113 of file magpie.h.

The documentation for this struct was generated from the following file:

· magpie.h

## 4.55 NUM\_JAC\_DATA Struct Reference

Data structure to form a numerical jacobian matrix with finite differences.

```
#include <lark.h>
```

## **Public Attributes**

double eps = sqrt(DBL\_EPSILON)

Perturbation value.

Matrix< double > Fx

Vector of function evaluations at x.

Matrix< double > Fxp

Vector of function evaluations at x+eps.

Matrix< double > dxj

Vector of perturbed x values.

## 4.55.1 Detailed Description

Data structure to form a numerical jacobian matrix with finite differences.

C-style object to be used in conjunction with the Numerical Jacobian algorithm. This algorithm will used double-precision finite-differences to formulate an approximate Jacobian matrix at the given variable state for the given residual/non-linear function.

Definition at line 569 of file lark.h.

4.55.2 Member Data Documentation

4.55.2.1 Matrix<double> NUM\_JAC\_DATA::dxj

Vector of perturbed x values.

Definition at line 574 of file lark.h.

4.55.2.2 double NUM\_JAC\_DATA::eps = sqrt(DBL\_EPSILON)

Perturbation value.

Definition at line 571 of file lark.h.

4.55.2.3 Matrix<double> NUM\_JAC\_DATA::Fx

Vector of function evaluations at x.

Definition at line 572 of file lark.h.

4.55.2.4 Matrix<double> NUM\_JAC\_DATA::Fxp

Vector of function evaluations at x+eps.

Definition at line 573 of file lark.h.

The documentation for this struct was generated from the following file:

· lark.h

### 4.56 OPTRANS\_DATA Struct Reference

Data structure for implementation of linear operator transposition.

```
#include <lark.h>
```

## **Public Attributes**

• Matrix< double > li

The ith column vector of the identity operator.

Matrix< double > Ai

The ith column vector of the user's linear operator.

### 4.56.1 Detailed Description

Data structure for implementation of linear operator transposition.

C-style object used in conjunction with the Operator Transpose algorithm to form an action of  $A^{\wedge}T*r$  when A is only available as a linear operator and not a matrix. This is a sub-routine required by GCR and GMRESR to stabilize the outer iterations.

Definition at line 324 of file lark.h.

4.56.2 Member Data Documentation

4.56.2.1 Matrix < double > OPTRANS\_DATA::Ai

The ith column vector of the user's linear operator.

Definition at line 327 of file lark.h.

```
4.56.2.2 Matrix<double> OPTRANS_DATA::li
```

The ith column vector of the identity operator.

Definition at line 326 of file lark.h.

The documentation for this struct was generated from the following file:

· lark.h

## 4.57 PCG\_DATA Struct Reference

Data structure for implementation of the PCG algorithms for symmetric linear systems.

```
#include <lark.h>
```

#### **Public Attributes**

• int maxit = 0

Maximum allowable iterations - default = min(vector\_size,1000)

• int iter = 0

Actual number of iterations taken.

· double alpha

Step size for new solution.

· double beta

Step size for new search direction.

double tol\_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

• double tol\_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

· double res

Absolute residual norm.

double relres

Relative residual norm.

double relres\_base

Initial residual norm.

double bestres

Best found residual norm.

• bool Output = true

True = print messages to console.

Matrix< double > x

Current solution to the linear system.

• Matrix< double > bestx

Best found solution to the linear system.

Matrix< double > r

Residual vector for the linear system.

Matrix< double > r\_old

Previous residual vector.

Matrix< double > z

Preconditioned residual vector (result of precon function)

Matrix< double > z\_old

Previous preconditioned residual vector.

• Matrix < double > p

Search direction.

Matrix< double > Ap

Result of matrix-vector multiplication.

## 4.57.1 Detailed Description

Data structure for implementation of the PCG algorithms for symmetric linear systems.

C-style object used in conjunction with the Preconditioned Conjugate Gradient (PCG) algorithm to iteratively solve a symmetric linear system of equations. This algorithm is optimal if your linear system is symmetric, but will not work at all if your system is asymmetric. For asymmetric systems, use one of the other linear methods.

Definition at line 217 of file lark.h.

4.57.2 Member Data Documentation

4.57.2.1 double PCG\_DATA::alpha

Step size for new solution.

Definition at line 222 of file lark.h.

4.57.2.2 Matrix<double> PCG\_DATA::Ap

Result of matrix-vector multiplication.

Definition at line 240 of file lark.h.

4.57.2.3 double PCG\_DATA::bestres

Best found residual norm.

Definition at line 229 of file lark.h.

4.57.2.4 Matrix < double > PCG\_DATA::bestx

Best found solution to the linear system.

Definition at line 234 of file lark.h.

4.57.2.5 double PCG\_DATA::beta

Step size for new search direction.

Definition at line 223 of file lark.h.

4.57.2.6 int PCG\_DATA::iter = 0

Actual number of iterations taken.

Definition at line 220 of file lark.h.

4.57.2.7 int PCG\_DATA::maxit = 0

Maximum allowable iterations - default = min(vector\_size,1000)

Definition at line 219 of file lark.h.

4.57.2.8 bool PCG\_DATA::Output = true

True = print messages to console.

Definition at line 231 of file lark.h.

4.57.2.9 Matrix<double> PCG\_DATA::p

Search direction.

Definition at line 239 of file lark.h.

4.57.2.10 Matrix<double> PCG\_DATA::r

Residual vector for the linear system.

Definition at line 235 of file lark.h.

4.57.2.11 Matrix<double> PCG\_DATA::r\_old

Previous residual vector.

Definition at line 236 of file lark.h.

4.57.2.12 double PCG\_DATA::relres

Relative residual norm.

Definition at line 227 of file lark.h.

4.57.2.13 double PCG\_DATA::relres\_base

Initial residual norm.

Definition at line 228 of file lark.h.

4.57.2.14 double PCG\_DATA::res

Absolute residual norm.

Definition at line 226 of file lark.h.

4.57.2.15 double PCG\_DATA::tol\_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

Definition at line 225 of file lark.h.

4.57.2.16 double PCG\_DATA::tol\_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 224 of file lark.h.

4.57.2.17 Matrix<double> PCG\_DATA::x

Current solution to the linear system.

Definition at line 233 of file lark.h.

4.57.2.18 Matrix<double> PCG\_DATA::z

Preconditioned residual vector (result of precon function)

Definition at line 237 of file lark.h.

 $\textbf{4.57.2.19} \quad \textbf{Matrix}{<} \textbf{double}{>} \textbf{PCG\_DATA}{::} \textbf{z\_old}$ 

Previous preconditioned residual vector.

Definition at line 238 of file lark.h.

The documentation for this struct was generated from the following file:

· lark.h

#### 4.58 PICARD\_DATA Struct Reference

Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems.

```
#include <lark.h>
```

#### **Public Attributes**

• int maxit = 0

Maximum allowable iterations - default = min(3\*vec size,1000)

• int iter = 0

Actual number of iterations.

• double tol rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

• double tol abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

· double res

Residual norm of the iterate.

· double relres

Relative residual norm of the iterate.

· double relres base

Initial residual norm.

· double bestres

Best found residual norm.

• bool Output = true

True = print messages to console.

Matrix< double > x0

Previous iterate solution vector.

Matrix< double > bestx

Best found solution vector.

• Matrix < double > r

Residual of the non-linear system.

## 4.58.1 Detailed Description

Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems.

C-style object used in conjunction with the Picard algorithm for solving a non-linear system of equations. This is an extradorinarily simple iterative method by which a weak or loose form of the non-linear system is solved based on an initial guess. User must supplied a residual function for the non-linear system and a function representing the weak solution. Generally, this method is less efficient than Newton methods, but is significantly cheaper.

Definition at line 449 of file lark.h.

## 4.58.2 Member Data Documentation

## 4.58.2.1 double PICARD\_DATA::bestres

Best found residual norm.

Definition at line 459 of file lark.h.

4.58.2.2 Matrix<double> PICARD\_DATA::bestx

Best found solution vector.

Definition at line 464 of file lark.h.

4.58.2.3 int PICARD\_DATA::iter = 0

Actual number of iterations.

Definition at line 452 of file lark.h.

4.58.2.4 int PICARD\_DATA::maxit = 0

Maximum allowable iterations - default = min(3\*vec\_size,1000)

Definition at line 451 of file lark.h.

4.58.2.5 bool PICARD\_DATA::Output = true

True = print messages to console.

Definition at line 461 of file lark.h.

4.58.2.6 Matrix<double> PICARD\_DATA::r

Residual of the non-linear system.

Definition at line 465 of file lark.h.

4.58.2.7 double PICARD\_DATA::relres

Relative residual norm of the iterate.

Definition at line 457 of file lark.h.

4.58.2.8 double PICARD\_DATA::relres\_base

Initial residual norm.

Definition at line 458 of file lark.h.

4.58.2.9 double PICARD\_DATA::res

Residual norm of the iterate.

Definition at line 456 of file lark.h.

4.58.2.10 double PICARD\_DATA::tol\_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

Definition at line 455 of file lark.h.

4.58.2.11 double PICARD\_DATA::tol\_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

Definition at line 454 of file lark.h.

4.58.2.12 Matrix<double> PICARD\_DATA::x0

Previous iterate solution vector.

Definition at line 463 of file lark.h.

The documentation for this struct was generated from the following file:

· lark.h

## 4.59 PJFNK\_DATA Struct Reference

Data structure for the implementation of the PJFNK algorithm for non-linear systems.

```
#include <lark.h>
```

#### **Public Attributes**

• int nl iter = 0

Number of non-linear iterations.

• int I iter = 0

Number of linear iterations.

• int fun call = 0

Actual number of function calls made.

int nl maxit = 0

Maximum allowable non-linear steps.

• int linear\_solver = -1

Flag to denote which linear solver to use - default = PJFNK Chooses.

double nl\_tol\_abs = 1e-6

Absolute Convergence tolerance for non-linear system - default = 1e-6.

• double nl\_tol\_rel = 1e-6

Relative Convergence tol for the non-linear system - default = 1e-6.

• double lin\_tol\_rel = 1e-6

Relative tolerance of the linear solver - default = 1e-6.

double lin\_tol\_abs = 1e-6

Absolute tolerance of the linear solver - default = 1e-6.

• double nl\_res

Absolute redidual norm for the non-linear system.

• double nl\_relres

Relative residual for the non-linear system.

double nl\_res\_base

Initial residual norm for the non-linear system.

double nl\_bestres

Best found residual norm.

• double eps =sqrt(DBL\_EPSILON)

Value of epsilon used jacvec - default = sqrt(DBL\_EPSILON)

bool NL\_Output = true

True = print PJFNK messages to console.

• bool L\_Output = false

True = print Linear messages to console.

• bool LineSearch = false

True = use Backtracking Linesearch for global convergence.

• bool Bounce = false

True = allow Linesearch to go outside local well, False = Strict local convergence.

Matrix< double > F

Stored fuction evaluation at x (also the residual)

Matrix< double > Fv

Stored function evaluation at x+eps\*v.

Matrix< double > v

Stored vector of x+eps\*v.

Matrix< double > x

Current solution vector for the non-linear system.

Matrix< double > bestx

Best found solution vector to the non-linear system.

GMRESLP\_DATA gmreslp\_dat

Data structure for the GMRESLP method.

PCG DATA pcg dat

Data structure for the PCG method.

BiCGSTAB\_DATA bicgstab\_dat

Data structure for the BiCGSTAB method.

CGS\_DATA cgs\_dat

Data structure for the CGS method.

GMRESRP\_DATA gmresrp\_dat

Data structure for the GMRESRP method.

• GCR\_DATA gcr\_dat

Data structure for the GCR method.

· GMRESR DATA gmresr dat

Data structure for the GMRESR method.

BACKTRACK\_DATA backtrack\_dat

Data structure for the Backtracking Linesearch algorithm.

const void \* res\_data

Data structure pointer for user's residual data.

const void \* precon\_data

Data structure pointer for user's preconditioning data.

int(\* funeval )(const Matrix< double > &x, Matrix< double > &F, const void \*res\_data)

Function pointer for the user's function F(x) using there data.

int(\* precon )(const Matrix< double > &r, Matrix< double > &p, const void \*precon\_data)

Function pointer for the user's preconditioning function for the linear system.

### 4.59.1 Detailed Description

Data structure for the implementation of the PJFNK algorithm for non-linear systems.

C-style object to be used in conjunction with the Preconditioned Jacobian-Free Newton-Krylov (PJFNK) method for solving a non-linear system of equations. You can use any of the Krylov methods listed in the krylov\_method enum to solve the linear sub-problem. When FOM is specified as the Krylov method, this algorithm becomes equivalent to an exact Newton method. If no Krylov method is specified, then the algorithm will try to pick a method based on the problem size and availability of preconditioning.

Definition at line 511 of file lark.h.

## 4.59.2 Member Data Documentation

## 4.59.2.1 BACKTRACK DATA PJFNK\_DATA::backtrack\_dat

Data structure for the Backtracking Linesearch algorithm.

Definition at line 550 of file lark.h.

## 4.59.2.2 Matrix<double> PJFNK\_DATA::bestx

Best found solution vector to the non-linear system.

Definition at line 538 of file lark.h.

4.59.2.3 BiCGSTAB\_DATA PJFNK\_DATA::bicgstab\_dat

Data structure for the BiCGSTAB method.

Definition at line 543 of file lark.h.

4.59.2.4 bool PJFNK\_DATA::Bounce = false

True = allow Linesearch to go outside local well, False = Strict local convergence.

Definition at line 532 of file lark.h.

4.59.2.5 CGS\_DATA PJFNK\_DATA::cgs\_dat

Data structure for the CGS method.

Definition at line 544 of file lark.h.

4.59.2.6 double PJFNK\_DATA::eps =sqrt(DBL\_EPSILON)

Value of epsilon used jacvec - default = sqrt(DBL\_EPSILON)

Definition at line 527 of file lark.h.

4.59.2.7 Matrix<double> PJFNK\_DATA::F

Stored fuction evaluation at x (also the residual)

Definition at line 534 of file lark.h.

4.59.2.8 int PJFNK\_DATA::fun\_call = 0

Actual number of function calls made.

Definition at line 515 of file lark.h.

4.59.2.9 int(\* PJFNK\_DATA::funeval)(const Matrix < double > &x, Matrix < double > &F, const void \*res\_data)

Function pointer for the user's function F(x) using there data.

Definition at line 559 of file lark.h.

4.59.2.10 Matrix<double> PJFNK\_DATA::Fv

Stored function evaluation at x+eps\*v.

Definition at line 535 of file lark.h.

4.59.2.11 GCR\_DATA PJFNK\_DATA::gcr\_dat

Data structure for the GCR method.

Definition at line 546 of file lark.h.

4.59.2.12 GMRESLP\_DATA PJFNK\_DATA::gmreslp\_dat

Data structure for the GMRESLP method.

Definition at line 541 of file lark.h.

4.59.2.13 GMRESR\_DATA PJFNK\_DATA::gmresr\_dat

Data structure for the GMRESR method.

Definition at line 547 of file lark.h.

4.59.2.14 GMRESRP\_DATA PJFNK\_DATA::gmresrp\_dat

Data structure for the GMRESRP method.

Definition at line 545 of file lark.h.

4.59.2.15 int PJFNK\_DATA::I\_iter = 0

Number of linear iterations.

Definition at line 514 of file lark.h.

4.59.2.16 bool PJFNK\_DATA::L\_Output = false

True = print Linear messages to console.

Definition at line 530 of file lark.h.

4.59.2.17 double PJFNK\_DATA::lin\_tol\_abs = 1e-6

Absolute tolerance of the linear solver - default = 1e-6.

Definition at line 522 of file lark.h.

4.59.2.18 double PJFNK\_DATA::lin\_tol\_rel = 1e-6

Relative tolerance of the linear solver - default = 1e-6.

Definition at line 521 of file lark.h.

4.59.2.19 int PJFNK\_DATA::linear\_solver = -1

Flag to denote which linear solver to use - default = PJFNK Chooses.

Definition at line 517 of file lark.h.

4.59.2.20 bool PJFNK\_DATA::LineSearch = false

True = use Backtracking Linesearch for global convergence.

Definition at line 531 of file lark.h.

4.59.2.21 double PJFNK\_DATA::nl\_bestres

Best found residual norm.

Definition at line 526 of file lark.h.

4.59.2.22 int PJFNK\_DATA::nl\_iter = 0

Number of non-linear iterations.

Definition at line 513 of file lark.h.

4.59.2.23 int PJFNK\_DATA::nl\_maxit = 0

Maximum allowable non-linear steps.

Definition at line 516 of file lark.h.

4.59.2.24 bool PJFNK\_DATA::NL\_Output = true

True = print PJFNK messages to console.

Definition at line 529 of file lark.h.

4.59.2.25 double PJFNK\_DATA::nl\_relres

Relative residual for the non-linear system.

Definition at line 524 of file lark.h.

4.59.2.26 double PJFNK\_DATA::nl\_res

Absolute redidual norm for the non-linear system.

Definition at line 523 of file lark.h.

4.59.2.27 double PJFNK\_DATA::nl\_res\_base

Initial residual norm for the non-linear system.

Definition at line 525 of file lark.h.

4.59.2.28 double PJFNK\_DATA::nl\_tol\_abs = 1e-6

Absolute Convergence tolerance for non-linear system - default = 1e-6.

Definition at line 519 of file lark.h.

4.59.2.29 double PJFNK\_DATA::nl\_tol\_rel = 1e-6

Relative Convergence tol for the non-linear system - default = 1e-6.

Definition at line 520 of file lark.h.

4.59.2.30 PCG\_DATA PJFNK\_DATA::pcg\_dat

Data structure for the PCG method.

Definition at line 542 of file lark.h.

4.59.2.31 int(\* PJFNK\_DATA::precon)(const Matrix < double > &r, Matrix < double > &p, const void \*precon\_data)

Function pointer for the user's preconditioning function for the linear system.

Definition at line 561 of file lark.h.

4.59.2.32 const void\* PJFNK\_DATA::precon\_data

Data structure pointer for user's preconditioning data.

Definition at line 557 of file lark.h.

4.59.2.33 const void\* PJFNK\_DATA::res\_data

Data structure pointer for user's residual data.

Definition at line 555 of file lark.h.

4.59.2.34 Matrix<double> PJFNK\_DATA::v

Stored vector of x+eps\*v.

Definition at line 536 of file lark.h.

4.59.2.35 Matrix<double> PJFNK\_DATA::x

Current solution vector for the non-linear system.

Definition at line 537 of file lark.h.

The documentation for this struct was generated from the following file:

· lark.h

## 4.60 PURE\_GAS Struct Reference

Data structure holding all the parameters for each pure gas spieces.

```
#include <egret.h>
```

#### **Public Attributes**

· double molecular\_weight

Given: molecular weights (g/mol)

double Sutherland Temp

Given: Sutherland's Reference Temperature (K)

double Sutherland Const

Given: Sutherland's Constant (K)

· double Sutherland\_Viscosity

Given: Sutherland's Reference Viscosity (g/cm/s)

· double specific\_heat

Given: Specific heat of the gas (J/g/K)

double molecular\_diffusion

Calculated: molecular diffusivities (cm<sup>2</sup>/s)

· double dynamic viscosity

Calculated: dynamic viscosities (g/cm/s)

· double density

Calculated: gas densities (g/cm<sup>\(\)</sup>3) {use RE3}.

• double Schmidt

Calculated: Value of the Schmidt number (-)

## 4.60.1 Detailed Description

Data structure holding all the parameters for each pure gas spieces.

C-style object that holds the constants and parameters associated with each pure gas species in the overall mixture. This information is used in conjunction with the kinetic theory of gases to produce approximations to many different gas properties needed in simulating gas dynamics, mobility of a gas through porous media, as well as some kinetic adsorption parameters such as diffusivities.

Definition at line 95 of file egret.h.

4.60.2 Member Data Documentation

4.60.2.1 double PURE\_GAS::density

Calculated: gas densities (g/cm<sup>3</sup>) {use RE3}.

Definition at line 107 of file egret.h.

4.60.2.2 double PURE\_GAS::dynamic\_viscosity

Calculated: dynamic viscosities (g/cm/s)

Definition at line 106 of file egret.h.

4.60.2.3 double PURE\_GAS::molecular\_diffusion

Calculated: molecular diffusivities (cm<sup>2</sup>/s)

Definition at line 105 of file egret.h.

4.60.2.4 double PURE\_GAS::molecular\_weight

Given: molecular weights (g/mol)

Definition at line 98 of file egret.h.

4.60.2.5 double PURE\_GAS::Schmidt

Calculated: Value of the Schmidt number (-)

Definition at line 108 of file egret.h.

4.60.2.6 double PURE\_GAS::specific\_heat

Given: Specific heat of the gas (J/g/K)

Definition at line 102 of file egret.h.

4.60.2.7 double PURE\_GAS::Sutherland\_Const

Given: Sutherland's Constant (K)

Definition at line 100 of file egret.h.

4.60.2.8 double PURE\_GAS::Sutherland\_Temp

Given: Sutherland's Reference Temperature (K)

Definition at line 99 of file egret.h.

4.60.2.9 double PURE\_GAS::Sutherland\_Viscosity

Given: Sutherland's Reference Viscosity (g/cm/s)

Definition at line 101 of file egret.h.

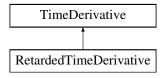
The documentation for this struct was generated from the following file:

• egret.h

# 4.61 RetardedTimeDerivative Class Reference

#include <RetardedTimeDerivative.h>

Inheritance diagram for RetardedTimeDerivative:



**Public Member Functions** 

RetardedTimeDerivative (const InputParameters &parameters)

#### **Protected Member Functions**

- virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

## **Private Attributes**

- int \_index
- · const MaterialProperty

```
< std::vector< Real > > & _retardation
```

#### 4.61.1 Detailed Description

Definition at line 12 of file RetardedTimeDerivative.h.

- 4.61.2 Constructor & Destructor Documentation
- 4.61.2.1 RetardedTimeDerivative::RetardedTimeDerivative ( const InputParameters & parameters )
- 4.61.3 Member Function Documentation
- **4.61.3.1** virtual Real RetardedTimeDerivative::computeQpJacobian() [protected], [virtual]
- **4.61.3.2** virtual Real RetardedTimeDerivative::computeQpResidual() [protected], [virtual]
- 4.61.4 Member Data Documentation
- **4.61.4.1** int RetardedTimeDerivative::\_index [private]

Definition at line 23 of file RetardedTimeDerivative.h.

```
4.61.4.2 const MaterialProperty < std::vector < Real > > & RetardedTimeDerivative::_retardation [private]
```

Definition at line 24 of file RetardedTimeDerivative.h.

The documentation for this class was generated from the following file:

• RetardedTimeDerivative.h

## 4.62 SCOPSOWL\_DATA Struct Reference

Primary data structure for SCOPSOWL simulations.

```
#include <scopsowl.h>
```

### **Public Attributes**

unsigned long int total\_steps

Running total of all calculation steps.

· int coord macro

Coordinate system for large pellet.

int coord\_micro

Coordinate system for small crystal (if any)

• int level = 2

Level of coupling between the different scales (default = 2)

```
    double sim_time

      Stopping time for the simulation (hrs)

    double t old

      Old time of the simulations (hrs)
· double t
      Current time of the simulations (hrs)
• double t counter = 0.0
      Counter for the time output.

    double t print

      Print output at every t_print time (hrs)
• bool Print2File = true
      True = results to .txt; False = no printing.

    bool Print2Console = true

      True = results to console; False = no printing.

    bool SurfDiff = true

      True = includes SKUA simulation if Heterogeneous; False = only uses MAGPIE.
• bool Heterogeneous = true
      True = pellet is made of binder and crystals, False = all one phase.
· double gas_velocity
      Superficial Gas Velocity arount pellet (cm/s)
• double total_pressure
      Gas phase total pressure (kPa)
· double gas_temperature
      Gas phase temperature (K)
· double pellet_radius
      Nominal radius of the pellet - macroscale domain (cm)
· double crystal radius
      Nominal radius of the crystal - microscale domain (um)
· double char_macro
      Characteristic size for macro scale (cm or cm^{\wedge}2) - only if pellet is not spherical.
· double char micro
      Characteristic size for micro scale (um or um^{\wedge}2) - only if crystal is not spherical.
· double binder_fraction
      Volume of binder per total volume of pellet (-)

    double binder_porosity

      Volume of pores per volume of binder (-)
· double binder poresize
      Nominal radius of the binder pores (cm)

    double pellet_density

      Mass of the pellet per volume of pellet (kg/L)

    bool DirichletBC = false

      True = Dirichlet BC; False = Neumann BC.

    bool NonLinear = true

      True = Non-linear solver; False = Linear solver.
std::vector< double > y
      Outside mole fractions of each component (-)

    std::vector< double > tempy

      Temporary place holder for gas mole fractions in other locations (-)

    FILE * OutputFile

      Output file pointer to the output file for postprocesses.
```

double(\* eval\_ads )(int i, int I, const void \*user\_data)

Function pointer for evaluating adsorption (mol/kg)

double(\* eval\_retard )(int i, int I, const void \*user\_data)

Function pointer for evaluating retardation (-)

double(\* eval\_diff)(int i, int I, const void \*user\_data)

Function pointer for evaluating pore diffusion (cm<sup>\(\)</sup>2/hr)

double(\* eval\_surfDiff)(int i, int I, const void \*user\_data)

Function pointer for evaluating surface diffusion (um<sup>2</sup>/hr)

double(\* eval kf )(int i, const void \*user data)

Function pointer for evaluating film mass transfer (cm/hr)

const void \* user data

Data structure for users info to calculate parameters.

MIXED\_GAS \* gas\_dat

Pointer to the MIXED\_GAS data structure (may or may not be used)

MAGPIE DATA magpie dat

Data structure for a magpie problem (to be used if not using skua)

std::vector< FINCH DATA > finch dat

Data structure for pore adsorption kinetics for all species (u in mol/L)

• std::vector < SCOPSOWL PARAM DATA > param dat

Data structure for parameter info for all species.

std::vector < SKUA\_DATA > skua\_dat

Data structure holding a skua object for all nodes (each skua has an object for each species)

#### 4.62.1 Detailed Description

Primary data structure for SCOPSOWL simulations.

C-style object holding necessary information to run a SCOPSOWL simulation. SCOPSOWL is a multi-scale problem involving PDE solution for the macro-scale adsorbent pellet and the micro-scale adsorbent crystals. As such, each SCOPSOWL simulation involves multiple SKUA simulations at the nodes in the macro-scale domain. Alternatively, if the user wishes to specify that the adsorbent is homogeneous, then you can run SCOPSOWL as a single-scale problem. Additionally, you can simplfy the model by assuming that the micro-scale diffusion is very fast, and therefore replace each SKUA simulation with a simpler MAGPIE evaluation. Details on running SCOPSOWL with the various options will be discussed in the SCOPSOWL\_SCENARIOS function.

Definition at line 92 of file scopsowl.h.

#### 4.62.2 Member Data Documentation

4.62.2.1 double SCOPSOWL\_DATA::binder\_fraction

Volume of binder per total volume of pellet (-)

Definition at line 116 of file scopsowl.h.

4.62.2.2 double SCOPSOWL\_DATA::binder\_poresize

Nominal radius of the binder pores (cm)

Definition at line 118 of file scopsowl.h.

4.62.2.3 double SCOPSOWL\_DATA::binder\_porosity

Volume of pores per volume of binder (-)

Definition at line 117 of file scopsowl.h.

4.62.2.4 double SCOPSOWL\_DATA::char\_macro

Characteristic size for macro scale (cm or cm<sup>2</sup>) - only if pellet is not spherical.

Definition at line 114 of file scopsowl.h.

4.62.2.5 double SCOPSOWL\_DATA::char\_micro

Characteristic size for micro scale (um or um<sup>2</sup>) - only if crystal is not spherical.

Definition at line 115 of file scopsowl.h.

4.62.2.6 int SCOPSOWL\_DATA::coord\_macro

Coordinate system for large pellet.

Definition at line 95 of file scopsowl.h.

4.62.2.7 int SCOPSOWL\_DATA::coord\_micro

Coordinate system for small crystal (if any)

Definition at line 96 of file scopsowl.h.

4.62.2.8 double SCOPSOWL\_DATA::crystal\_radius

Nominal radius of the crystal - microscale domain (um)

Definition at line 113 of file scopsowl.h.

4.62.2.9 bool SCOPSOWL\_DATA::DirichletBC = false

True = Dirichlet BC; False = Neumann BC.

Definition at line 121 of file scopsowl.h.

4.62.2.10 double(\* SCOPSOWL\_DATA::eval\_ads)(int i, int I, const void \*user\_data)

Function pointer for evaluating adsorption (mol/kg)

Definition at line 127 of file scopsowl.h.

4.62.2.11 double(\* SCOPSOWL\_DATA::eval\_diff)(int i, int I, const void \*user\_data)

Function pointer for evaluating pore diffusion (cm<sup>2</sup>/hr)

Definition at line 129 of file scopsowl.h.

4.62.2.12 double(\* SCOPSOWL\_DATA::eval\_kf)(int i, const void \*user\_data)

Function pointer for evaluating film mass transfer (cm/hr)

Definition at line 131 of file scopsowl.h.

4.62.2.13 double(\* SCOPSOWL\_DATA::eval\_retard)(int i, int l, const void \*user\_data)

Function pointer for evaluating retardation (-)

Definition at line 128 of file scopsowl.h.

4.62.2.14 double(\* SCOPSOWL\_DATA::eval\_surfDiff)(int i, int I, const void \*user\_data)

Function pointer for evaluating surface diffusion (um<sup>2</sup>/hr)

Definition at line 130 of file scopsowl.h.

4.62.2.15 std::vector<FINCH\_DATA> SCOPSOWL\_DATA::finch\_dat

Data structure for pore adsorption kinetics for all species (u in mol/L)

Definition at line 136 of file scopsowl.h.

4.62.2.16 MIXED\_GAS\* SCOPSOWL\_DATA::gas\_dat

Pointer to the MIXED GAS data structure (may or may not be used)

Definition at line 134 of file scopsowl.h.

4.62.2.17 double SCOPSOWL\_DATA::gas\_temperature

Gas phase temperature (K)

Definition at line 111 of file scopsowl.h.

4.62.2.18 double SCOPSOWL\_DATA::gas\_velocity

Superficial Gas Velocity arount pellet (cm/s)

Definition at line 109 of file scopsowl.h.

4.62.2.19 bool SCOPSOWL\_DATA::Heterogeneous = true

True = pellet is made of binder and crystals, False = all one phase.

Definition at line 107 of file scopsowl.h.

4.62.2.20 int SCOPSOWL\_DATA::level = 2

Level of coupling between the different scales (default = 2)

Definition at line 97 of file scopsowl.h.

4.62.2.21 MAGPIE\_DATA SCOPSOWL\_DATA::magpie\_dat

Data structure for a magpie problem (to be used if not using skua)

Definition at line 135 of file scopsowl.h.

4.62.2.22 bool SCOPSOWL\_DATA::NonLinear = true

True = Non-linear solver; False = Linear solver.

Definition at line 122 of file scopsowl.h.

4.62.2.23 FILE\* SCOPSOWL\_DATA::OutputFile

Output file pointer to the output file for postprocesses.

Definition at line 126 of file scopsowl.h.

 $4.62.2.24 \quad std:: vector < SCOPSOWL\_PARAM\_DATA > SCOPSOWL\_DATA:: param\_dat$ 

Data structure for parameter info for all species.

Definition at line 137 of file scopsowl.h.

4.62.2.25 double SCOPSOWL\_DATA::pellet\_density

Mass of the pellet per volume of pellet (kg/L)

Definition at line 119 of file scopsowl.h.

4.62.2.26 double SCOPSOWL\_DATA::pellet\_radius

Nominal radius of the pellet - macroscale domain (cm)

Definition at line 112 of file scopsowl.h.

4.62.2.27 bool SCOPSOWL\_DATA::Print2Console = true

True = results to console; False = no printing.

Definition at line 105 of file scopsowl.h.

4.62.2.28 bool SCOPSOWL\_DATA::Print2File = true

True = results to .txt; False = no printing.

Definition at line 104 of file scopsowl.h.

4.62.2.29 double SCOPSOWL\_DATA::sim\_time

Stopping time for the simulation (hrs)

Definition at line 98 of file scopsowl.h.

4.62.2.30 std::vector<SKUA\_DATA> SCOPSOWL\_DATA::skua\_dat

Data structure holding a skua object for all nodes (each skua has an object for each species)

Definition at line 139 of file scopsowl.h.

4.62.2.31 bool SCOPSOWL\_DATA::SurfDiff = true

True = includes SKUA simulation if Heterogeneous; False = only uses MAGPIE.

Definition at line 106 of file scopsowl.h.

4.62.2.32 double SCOPSOWL\_DATA::t

Current time of the simulations (hrs)

Definition at line 100 of file scopsowl.h.

4.62.2.33 double SCOPSOWL\_DATA::t\_counter = 0.0

Counter for the time output.

Definition at line 101 of file scopsowl.h.

4.62.2.34 double SCOPSOWL\_DATA::t\_old

Old time of the simulations (hrs)

Definition at line 99 of file scopsowl.h.

4.62.2.35 double SCOPSOWL\_DATA::t\_print

Print output at every t\_print time (hrs)

Definition at line 102 of file scopsowl.h.

4.62.2.36 std::vector<double> SCOPSOWL\_DATA::tempy

Temporary place holder for gas mole fractions in other locations (-)

Definition at line 124 of file scopsowl.h.

4.62.2.37 double SCOPSOWL\_DATA::total\_pressure

Gas phase total pressure (kPa)

Definition at line 110 of file scopsowl.h.

4.62.2.38 unsigned long int SCOPSOWL\_DATA::total\_steps

Running total of all calculation steps.

Definition at line 94 of file scopsowl.h.

4.62.2.39 const void\* SCOPSOWL\_DATA::user\_data

Data structure for users info to calculate parameters.

Definition at line 133 of file scopsowl.h.

4.62.2.40 std::vector<double> SCOPSOWL\_DATA::y

Outside mole fractions of each component (-)

Definition at line 123 of file scopsowl.h.

The documentation for this struct was generated from the following file:

· scopsowl.h

## 4.63 SCOPSOWL\_PARAM\_DATA Struct Reference

Data structure for the species' parameters in SCOPSOWL.

```
#include <scopsowl.h>
```

## **Public Attributes**

Matrix< double > qAvg

Average adsorbed amount for a species at each node (mol/kg)

Matrix< double > qAvg\_old

Old Average adsorbed amount for a species at each node (mol/kg)

Matrix< double > Qst

Heat of adsorption for all nodes (J/mol)

• Matrix< double > Qst\_old

Old Heat of adsorption for all nodes (J/mol)

Matrix< double > dq\_dc

Storage vector for current adsorption slope/strength (dq/dc) (L/kg)

double xIC

Initial conditions for adsorbed molefractions.

double qIntegralAvg

Integral average of adsorption over the entire pellet (mol/kg)

double qIntegralAvg\_old

Old Integral average of adsorption over the entire pellet (mol/kg)

double QstAvg

Integral average heat of adsorption (J/mol)

double QstAvg\_old

Old integral average heat of adsorption (J/mol)

• double qo

Boundary value of adsorption if using Dirichlet BCs (mol/kg)

· double Qsto

Boundary value of adsorption heat if using Dirichlet BCs (J/mol)

· double dq dco

Boundary value of adsorption slope for Dirichelt BCs (L/kg)

• double pore\_diffusion

Value for constant pore diffusion (cm $^{\wedge}$ 2/hr)

· double film transfer

Value for constant film mass transfer (cm/hr)

· double activation\_energy

Activation energy for surface diffusion (J/mol)

· double ref diffusion

Reference state surface diffusivity (um\^2/hr)

double ref temperature

Reference temperature for empirical adjustments (K)

· double affinity

Affinity parameter used in empirical adjustments (-)

- double ref pressure
- · bool Adsorbable

True = species can adsorb; False = species cannot adsorb.

std::string speciesName

String to hold the name of each species.

## 4.63.1 Detailed Description

Data structure for the species' parameters in SCOPSOWL.

C-style object that holds information on all species for a particular SCOPSOWL simulation. Initial conditions, kinetic parameters, and interim matrix objects are stored here for use in various SCOSPSOWL functions.

Definition at line 44 of file scopsowl.h.

## 4.63.2 Member Data Documentation

# 4.63.2.1 double SCOPSOWL\_PARAM\_DATA::activation\_energy

Activation energy for surface diffusion (J/mol)

Definition at line 69 of file scopsowl.h.

# 4.63.2.2 bool SCOPSOWL\_PARAM\_DATA::Adsorbable

True = species can adsorb; False = species cannot adsorb.

Definition at line 75 of file scopsowl.h.

# 4.63.2.3 double SCOPSOWL\_PARAM\_DATA::affinity

Affinity parameter used in empirical adjustments (-)

Definition at line 72 of file scopsowl.h.

# $4.63.2.4 \quad Matrix{<} double{>} SCOPSOWL\_PARAM\_DATA::dq\_dc$

Storage vector for current adsorption slope/strength (dq/dc) (L/kg)

Definition at line 52 of file scopsowl.h.

4.63.2.5 double SCOPSOWL\_PARAM\_DATA::dq\_dco

Boundary value of adsorption slope for Dirichelt BCs (L/kg)

Definition at line 64 of file scopsowl.h.

4.63.2.6 double SCOPSOWL\_PARAM\_DATA::film\_transfer

Value for constant film mass transfer (cm/hr)

Definition at line 67 of file scopsowl.h.

4.63.2.7 double SCOPSOWL\_PARAM\_DATA::pore\_diffusion

Value for constant pore diffusion (cm<sup>2</sup>/hr)

Definition at line 66 of file scopsowl.h.

4.63.2.8 Matrix<double> SCOPSOWL\_PARAM\_DATA::qAvg

Average adsorbed amount for a species at each node (mol/kg)

Definition at line 46 of file scopsowl.h.

 $4.63.2.9 \quad \textbf{Matrix} {<} \textbf{double} {>} \ \textbf{SCOPSOWL\_PARAM\_DATA} :: \textbf{qAvg\_old}$ 

Old Average adsorbed amount for a species at each node (mol/kg)

Definition at line 47 of file scopsowl.h.

4.63.2.10 double SCOPSOWL\_PARAM\_DATA::qIntegralAvg

Integral average of adsorption over the entire pellet (mol/kg)

Definition at line 56 of file scopsowl.h.

4.63.2.11 double SCOPSOWL\_PARAM\_DATA::qIntegralAvg\_old

Old Integral average of adsorption over the entire pellet (mol/kg)

Definition at line 57 of file scopsowl.h.

4.63.2.12 double SCOPSOWL\_PARAM\_DATA::go

Boundary value of adsorption if using Dirichlet BCs (mol/kg)

Definition at line 62 of file scopsowl.h.

4.63.2.13 Matrix < double > SCOPSOWL\_PARAM\_DATA::Qst

Heat of adsorption for all nodes (J/mol)

Definition at line 49 of file scopsowl.h.

4.63.2.14 Matrix<double> SCOPSOWL\_PARAM\_DATA::Qst\_old

Old Heat of adsorption for all nodes (J/mol)

Definition at line 50 of file scopsowl.h.

4.63.2.15 double SCOPSOWL\_PARAM\_DATA::QstAvg

Integral average heat of adsorption (J/mol)

Definition at line 59 of file scopsowl.h.

4.63.2.16 double SCOPSOWL\_PARAM\_DATA::QstAvg\_old

Old integral average heat of adsorption (J/mol)

Definition at line 60 of file scopsowl.h.

4.63.2.17 double SCOPSOWL\_PARAM\_DATA::Qsto

Boundary value of adsorption heat if using Dirichlet BCs (J/mol)

Definition at line 63 of file scopsowl.h.

4.63.2.18 double SCOPSOWL\_PARAM\_DATA::ref\_diffusion

Reference state surface diffusivity (um<sup>2</sup>/hr)

Definition at line 70 of file scopsowl.h.

4.63.2.19 double SCOPSOWL\_PARAM\_DATA::ref\_pressure

Definition at line 73 of file scopsowl.h.

4.63.2.20 double SCOPSOWL\_PARAM\_DATA::ref\_temperature

Reference temperature for empirical adjustments (K)

Definition at line 71 of file scopsowl.h.

4.63.2.21 std::string SCOPSOWL\_PARAM\_DATA::speciesName

String to hold the name of each species.

Definition at line 77 of file scopsowl.h.

4.63.2.22 double SCOPSOWL\_PARAM\_DATA::xIC

Initial conditions for adsorbed molefractions.

Definition at line 54 of file scopsowl.h.

The documentation for this struct was generated from the following file:

· scopsowl.h

## 4.64 SKUA\_DATA Struct Reference

Data structure for all simulation information in SKUA.

#include <skua.h>

# **Public Attributes**

unsigned long int total\_steps

Running total of all calculation steps.

· int coord

Used to determine the coordinates of the problem.

• double sim\_time

Stopping time for the simulation (hrs)

double t\_old

Old time of the simulations (hrs)

• double t

```
Current time of the simulations (hrs)

• double t_counter = 0.0

Counts for print times for output (hrs)

• double t_print

Prints out every t_print time (hrs)

• double qTn

Old total amounts adsorbed (mol/kg)

• double qTnp1

New total amounts adsorbed (mol/kg)

• bool Print2File = true

True = results to .txt; False = no printing.

• bool Print2Console = true
```

True = results to console; False = no printing.

· double gas\_velocity

Superficial Gas Velocity arount pellet (cm/s)

· double pellet radius

Nominal radius of the pellet/crystal (um)

double char\_measure

Length or Area if in Cylindrical or Cartesian coordinates (um or um\(^2\))

• bool DirichletBC = true

True = Dirichlet BC; False = Neumann BC.

• bool NonLinear = true

True = Non-linear solver; False = Linear solver.

std::vector< double > y

Outside mole fractions of each component (-)

FILE \* OutputFile

Output file pointer to the output file.

double(\* eval diff)(int i, int I, const void \*user data)

Function pointer for evaluating surface diffusivity.

double(\* eval\_kf )(int i, const void \*user\_data)

Function pointer for evaluating film mass transfer.

const void \* user\_data

Data structure for user's information needed in parameter functions.

· MAGPIE DATA magpie dat

Data structure for adsorption equilibria (see magpie.h)

• MIXED\_GAS \* gas\_dat

Pointer to the MIXED\_GAS data structure (see egret.h)

std::vector< FINCH\_DATA > finch\_dat

Data structure for adsorption kinetics (see finch.h)

std::vector< SKUA\_PARAM > param\_dat

Data structure for SKUA specific parameters.

## 4.64.1 Detailed Description

Data structure for all simulation information in SKUA.

C-style object holding all data, functions, and other objects needed to successfully run a SKUA simulation. This object holds system information, such as boundary condition type, adsorbent size, and total adsorption, and also contains structure for EGRET (egret.h), FINCH (finch.h), and MAGPIE (magpie.h) calculations. Function pointers for evaluation of the surface diffusivity and film mass transfer coefficients can be overriden by the user to change the behavior of the SKUA simulation. However, defaults are also provided for these functions.

Definition at line 84 of file skua.h.

4.64.2 Member Data Documentation

4.64.2.1 double SKUA\_DATA::char\_measure

Length or Area if in Cylindrical or Cartesian coordinates (um or um<sup>2</sup>)

Definition at line 100 of file skua.h.

4.64.2.2 int SKUA\_DATA::coord

Used to determine the coordinates of the problem.

Definition at line 87 of file skua.h.

4.64.2.3 bool SKUA\_DATA::DirichletBC = true

True = Dirichlet BC; False = Neumann BC.

Definition at line 101 of file skua.h.

4.64.2.4 double(\* SKUA\_DATA::eval\_diff)(int i, int I, const void \*user\_data)

Function pointer for evaluating surface diffusivity.

Definition at line 106 of file skua.h.

4.64.2.5 double(\* SKUA\_DATA::eval\_kf)(int i, const void \*user\_data)

Function pointer for evaluating film mass transfer.

Definition at line 107 of file skua.h.

4.64.2.6 std::vector<FINCH\_DATA> SKUA\_DATA::finch\_dat

Data structure for adsorption kinetics (see finch.h)

Definition at line 111 of file skua.h.

4.64.2.7 MIXED\_GAS\* SKUA\_DATA::gas\_dat

Pointer to the MIXED GAS data structure (see egret.h)

Definition at line 110 of file skua.h.

4.64.2.8 double SKUA\_DATA::gas\_velocity

Superficial Gas Velocity arount pellet (cm/s)

Definition at line 98 of file skua.h.

4.64.2.9 MAGPIE\_DATA SKUA\_DATA::magpie\_dat

Data structure for adsorption equilibria (see magpie.h)

Definition at line 109 of file skua.h.

4.64.2.10 bool SKUA\_DATA::NonLinear = true

True = Non-linear solver: False = Linear solver.

Definition at line 102 of file skua.h.

4.64.2.11 FILE\* SKUA\_DATA::OutputFile

Output file pointer to the output file.

Definition at line 105 of file skua.h.

4.64.2.12 std::vector<SKUA\_PARAM> SKUA\_DATA::param\_dat

Data structure for SKUA specific parameters.

Definition at line 112 of file skua.h.

4.64.2.13 double SKUA\_DATA::pellet\_radius

Nominal radius of the pellet/crystal (um)

Definition at line 99 of file skua.h.

4.64.2.14 bool SKUA\_DATA::Print2Console = true

True = results to console; False = no printing.

Definition at line 96 of file skua.h.

4.64.2.15 bool SKUA\_DATA::Print2File = true

True = results to .txt; False = no printing.

Definition at line 95 of file skua.h.

4.64.2.16 double SKUA\_DATA::qTn

Old total amounts adsorbed (mol/kg)

Definition at line 93 of file skua.h.

4.64.2.17 double SKUA\_DATA::qTnp1

New total amounts adsorbed (mol/kg)

Definition at line 94 of file skua.h.

4.64.2.18 double SKUA\_DATA::sim\_time

Stopping time for the simulation (hrs)

Definition at line 88 of file skua.h.

4.64.2.19 double SKUA\_DATA::t

Current time of the simulations (hrs)

Definition at line 90 of file skua.h.

4.64.2.20 double SKUA\_DATA::t\_counter = 0.0

Counts for print times for output (hrs)

Definition at line 91 of file skua.h.

4.64.2.21 double SKUA\_DATA::t\_old

Old time of the simulations (hrs)

Definition at line 89 of file skua.h.

4.64.2.22 double SKUA\_DATA::t\_print

Prints out every t\_print time (hrs)

Definition at line 92 of file skua.h.

4.64.2.23 unsigned long int SKUA\_DATA::total\_steps

Running total of all calculation steps.

Definition at line 86 of file skua.h.

4.64.2.24 const void\* SKUA\_DATA::user\_data

Data structure for user's information needed in parameter functions.

Definition at line 108 of file skua.h.

4.64.2.25 std::vector<double> SKUA\_DATA::y

Outside mole fractions of each component (-)

Definition at line 103 of file skua.h.

The documentation for this struct was generated from the following file:

• skua.h

## 4.65 SKUA\_PARAM Struct Reference

Data structure for species' parameters in SKUA.

```
#include <skua.h>
```

## **Public Attributes**

- double activation\_energy
- double ref\_diffusion
- double ref\_temperature
- · double affinity
- double ref\_pressure
- double film\_transfer
- double xIC
- double y\_eff
- double Qstn
- double Qstnp1
- double xn
- double xnp1
- bool Adsorbable
- std::string speciesName

## 4.65.1 Detailed Description

Data structure for species' parameters in SKUA.

C-style object holding data and parameters associated with the gas/solid species in the overall SKUA system. These parameters are used in to modify surface diffusivity with temperature, establish film mass transfer coefficients, formulate the initial conditions, and store solution results for heat of adsorption and adsorbed mole fractions. One of these objects will be created for each species in the gas system.

Definition at line 56 of file skua.h.

4.65.2 Member Data Documentation

4.65.2.1 double SKUA\_PARAM::activation\_energy

Definition at line 58 of file skua.h.

4.65.2.2 bool SKUA\_PARAM::Adsorbable

Definition at line 73 of file skua.h.

4.65.2.3 double SKUA\_PARAM::affinity

Definition at line 61 of file skua.h.

4.65.2.4 double SKUA\_PARAM::film\_transfer

Definition at line 63 of file skua.h.

4.65.2.5 double SKUA\_PARAM::Qstn

Definition at line 68 of file skua.h.

4.65.2.6 double SKUA\_PARAM::Qstnp1

Definition at line 69 of file skua.h.

4.65.2.7 double SKUA\_PARAM::ref\_diffusion

Definition at line 59 of file skua.h.

4.65.2.8 double SKUA\_PARAM::ref\_pressure

Definition at line 62 of file skua.h.

4.65.2.9 double SKUA\_PARAM::ref\_temperature

Definition at line 60 of file skua.h.

4.65.2.10 std::string SKUA\_PARAM::speciesName

Definition at line 75 of file skua.h.

4.65.2.11 double SKUA\_PARAM::xIC

Definition at line 65 of file skua.h.

4.65.2.12 double SKUA\_PARAM::xn

Definition at line 70 of file skua.h.

4.65.2.13 double SKUA\_PARAM::xnp1

Definition at line 71 of file skua.h.

4.65.2.14 double SKUA\_PARAM::y\_eff

Definition at line 66 of file skua.h.

The documentation for this struct was generated from the following file:

• skua.h

## 4.66 SYSTEM\_DATA Struct Reference

```
System Data Structure.
```

```
#include <magpie.h>
```

## **Public Attributes**

double T

System Temperature (K)

double PT

Total Pressure (kPa)

double qT

Total Amount adsorbed (mol/kg)

double PI

Total Lumped Spreading Pressure (mol/kg)

· double pi

Actual Spreading pressure  $(J/m^{\wedge}2)$ 

· double As

Specific surface area of adsorbent ( $m^2/kg$ )

int N

Total Number of Components.

- int |
- int J
- int K

Special indices used to keep track of sub-systems.

· unsigned long int total\_eval

Counter to keep track of total number of non-linear steps.

double avg\_norm

Used to store all norms from evaluations then average at end of run.

· double max norm

Used to store the maximum e.norm calculated from non-linear iterations.

• int Sys

Number of sub-systems to solve.

• int Par

Number of binary parameters to solve for.

bool Recover

If Recover == false, standard GPAST using y's as knowns.

· bool Carrier

If there is an inert carrier gas, Carrier == true.

bool Ideal

If the behavior of the system is determined to be ideal, then Ideal == true.

bool Output

Boolean to suppress output if desired (true = display, false = no display.

# 4.66.1 Detailed Description

System Data Structure.

C-style object holding all the data associated with the overall system to be modeled.

Definition at line 139 of file magpie.h.

4.66.2 Member Data Documentation

4.66.2.1 double SYSTEM\_DATA::As

Specific surface area of adsorbent (m<sup>2</sup>/kg)

Definition at line 146 of file magpie.h.

4.66.2.2 double SYSTEM\_DATA::avg\_norm

Used to store all norms from evaluations then average at end of run.

Definition at line 150 of file magpie.h.

4.66.2.3 bool SYSTEM\_DATA::Carrier

If there is an inert carrier gas, Carrier == true.

Definition at line 155 of file magpie.h.

4.66.2.4 int SYSTEM\_DATA::I

Definition at line 148 of file magpie.h.

4.66.2.5 bool SYSTEM\_DATA::Ideal

If the behavior of the system is determined to be ideal, then Ideal == true.

Definition at line 156 of file magpie.h.

4.66.2.6 int SYSTEM\_DATA::J

Definition at line 148 of file magpie.h.

4.66.2.7 int SYSTEM\_DATA::K

Special indices used to keep track of sub-systems.

Definition at line 148 of file magpie.h.

4.66.2.8 double SYSTEM\_DATA::max\_norm

Used to store the maximum e.norm calculated from non-linear iterations.

Definition at line 151 of file magpie.h.

4.66.2.9 int SYSTEM\_DATA::N

Total Number of Components.

Definition at line 147 of file magpie.h.

4.66.2.10 bool SYSTEM\_DATA::Output

Boolean to suppress output if desired (true = display, false = no display.

Definition at line 157 of file magpie.h.

4.66.2.11 int SYSTEM\_DATA::Par

Number of binary parameters to solve for.

Definition at line 153 of file magpie.h.

4.66.2.12 double SYSTEM\_DATA::PI

Total Lumped Spreading Pressure (mol/kg)

Definition at line 144 of file magpie.h.

4.66.2.13 double SYSTEM\_DATA::pi

Actual Spreading pressure (J/m<sup>2</sup>)

Definition at line 145 of file magpie.h.

4.66.2.14 double SYSTEM\_DATA::PT

Total Pressure (kPa)

Definition at line 142 of file magpie.h.

4.66.2.15 double SYSTEM\_DATA::qT

Total Amount adsorbed (mol/kg)

Definition at line 143 of file magpie.h.

4.66.2.16 bool SYSTEM\_DATA::Recover

If Recover == false, standard GPAST using y's as knowns.

Definition at line 154 of file magpie.h.

4.66.2.17 int SYSTEM\_DATA::Sys

Number of sub-systems to solve.

Definition at line 152 of file magpie.h.

4.66.2.18 double SYSTEM\_DATA::T

System Temperature (K)

Definition at line 141 of file magpie.h.

4.66.2.19 unsigned long int SYSTEM\_DATA::total\_eval

Counter to keep track of total number of non-linear steps.

Definition at line 149 of file magpie.h.

The documentation for this struct was generated from the following file:

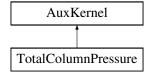
• magpie.h

# 4.67 TotalColumnPressure Class Reference

Total Column Pressure class inherits from AuxKernel.

#include <TotalColumnPressure.h>

Inheritance diagram for TotalColumnPressure:



#### **Public Member Functions**

TotalColumnPressure (const InputParameters &parameters)
 Standard MOOSE public constructor.

## **Protected Member Functions**

virtual Real computeValue ()
 Required MOOSE function override.

## **Private Attributes**

• VariableValue & \_temperature

Reference to the temperature non-linear variable.

std::vector< unsigned int > \_index

Indices of the gaseous species coupled to the object.

std::vector< VariableValue \* > \_gas\_conc

Pointer list for the non-linear concentration variables.

## 4.67.1 Detailed Description

Total Column Pressure class inherits from AuxKernel.

This class object creates an AuxKernel for use in the MOOSE framework. The AuxKernel will calculate the total column pressure (in kPa) based on the non-linear variables of temperature and concentration of each species in the gas phase. Total pressure is calculated based on the ideal gas law.

Definition at line 54 of file TotalColumnPressure.h.

# 4.67.2 Constructor & Destructor Documentation

4.67.2.1 TotalColumnPressure::TotalColumnPressure ( const InputParameters & parameters )

Standard MOOSE public constructor.

4.67.3 Member Function Documentation

4.67.3.1 virtual Real TotalColumnPressure::computeValue() [protected], [virtual]

Required MOOSE function override.

This is the function that is called by the MOOSE framework when a calculation of the total system pressure is needed. You are required to override this function for any inherited AuxKernel.

4.67.4 Member Data Documentation

**4.67.4.1** std::vector<VariableValue \*> TotalColumnPressure::\_gas\_conc [private]

Pointer list for the non-linear concentration variables.

Definition at line 70 of file TotalColumnPressure.h.

**4.67.4.2** std::vector<unsigned int> TotalColumnPressure::\_index [private]

Indices of the gaseous species coupled to the object.

Definition at line 69 of file TotalColumnPressure.h.

**4.67.4.3 VariableValue& TotalColumnPressure::\_temperature** [private]

Reference to the temperature non-linear variable.

Definition at line 68 of file TotalColumnPressure.h.

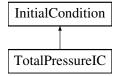
The documentation for this class was generated from the following file:

• TotalColumnPressure.h

## 4.68 TotalPressureIC Class Reference

#include <TotalPressureIC.h>

Inheritance diagram for TotalPressureIC:



## **Public Member Functions**

- TotalPressureIC (const InputParameters &parameters)
- virtual Real value (const Point &p)

## **Private Attributes**

· Real PT IC

## 4.68.1 Detailed Description

Definition at line 10 of file TotalPressureIC.h.

- 4.68.2 Constructor & Destructor Documentation
- 4.68.2.1 TotalPressurelC::TotalPressurelC ( const InputParameters & parameters )
- 4.68.3 Member Function Documentation
- 4.68.3.1 virtual Real TotalPressurelC::value ( const Point & p ) [virtual]
- 4.68.4 Member Data Documentation
- **4.68.4.1 Real TotalPressurelC::\_PT\_IC** [private]

Definition at line 17 of file TotalPressureIC.h.

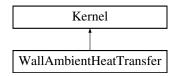
The documentation for this class was generated from the following file:

TotalPressureIC.h

## 4.69 WallAmbientHeatTransfer Class Reference

#include <WallAmbientHeatTransfer.h>

Inheritance diagram for WallAmbientHeatTransfer:



## **Public Member Functions**

WallAmbientHeatTransfer (const InputParameters &parameters)

## **Protected Member Functions**

- virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

#### **Private Attributes**

- const MaterialProperty< Real > & \_wall\_exterior\_transfer\_coeff
- const MaterialProperty< Real > & \_inner\_dia
- const MaterialProperty< Real > & \_outer\_dia
- VariableValue & \_ambient\_temp

## 4.69.1 Detailed Description

Definition at line 11 of file WallAmbientHeatTransfer.h.

- 4.69.2 Constructor & Destructor Documentation
- 4.69.2.1 WallAmbientHeatTransfer::WallAmbientHeatTransfer ( const InputParameters & parameters )
- 4.69.3 Member Function Documentation
- 4.69.3.1 virtual Real WallAmbientHeatTransfer::computeQpJacobian() [protected], [virtual]
- 4.69.3.2 virtual Real WallAmbientHeatTransfer::computeQpResidual() [protected], [virtual]
- 4.69.4 Member Data Documentation
- **4.69.4.1 VariableValue& WallAmbientHeatTransfer::\_ambient\_temp** [private]

Definition at line 25 of file WallAmbientHeatTransfer.h.

**4.69.4.2** const MaterialProperty<Real>& WallAmbientHeatTransfer::\_inner\_dia [private]

Definition at line 22 of file WallAmbientHeatTransfer.h.

4.69.4.3 const Material Property < Real > & Wall Ambient Heat Transfer::\_outer\_dia [private]

Definition at line 23 of file WallAmbientHeatTransfer.h.

4.69.4.4 const MaterialProperty<Real>& WallAmbientHeatTransfer::\_wall\_exterior\_transfer\_coeff [private]

Definition at line 21 of file WallAmbientHeatTransfer.h.

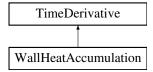
The documentation for this class was generated from the following file:

· WallAmbientHeatTransfer.h

## 4.70 WallHeatAccumulation Class Reference

#include <WallHeatAccumulation.h>

Inheritance diagram for WallHeatAccumulation:



## **Public Member Functions**

• WallHeatAccumulation (const InputParameters &parameters)

## **Protected Member Functions**

- · virtual Real computeQpResidual ()
- virtual Real computeQpJacobian ()

## **Private Attributes**

- const MaterialProperty< Real > & \_wall\_density
- const MaterialProperty< Real > & \_wall\_heat\_capacity

## 4.70.1 Detailed Description

Definition at line 12 of file WallHeatAccumulation.h.

- 4.70.2 Constructor & Destructor Documentation
- 4.70.2.1 WallHeatAccumulation::WallHeatAccumulation ( const InputParameters & parameters )
- 4.70.3 Member Function Documentation
- 4.70.3.1 virtual Real WallHeatAccumulation::computeQpJacobian( ) [protected], [virtual]
- 4.70.3.2 virtual Real WallHeatAccumulation::computeQpResidual( ) [protected], [virtual]
- 4.70.4 Member Data Documentation
- **4.70.4.1 const MaterialProperty < Real > & WallHeatAccumulation::\_wall\_density** [private]

Definition at line 23 of file WallHeatAccumulation.h.

5 File Documentation 185

4.70.4.2 const MaterialProperty < Real > & WallHeatAccumulation::\_wall\_heat\_capacity [private]

Definition at line 24 of file WallHeatAccumulation.h.

The documentation for this class was generated from the following file:

· WallHeatAccumulation.h

# 5 File Documentation

## 5.1 AdsorbentProperties.h File Reference

Material Properties kernel that will setup and hold all information associated with the adsorbent.

```
#include "Material.h"
```

## Classes

· class AdsorbentProperties

AdsorbentProperties class object inherits from Material object.

## **Functions**

template<>
 InputParameters validParams< AdsorbentProperties > ()

# 5.1.1 Detailed Description

Material Properties kernel that will setup and hold all information associated with the adsorbent. Material Properties kernel that will setup and hold all information associated with the fixed-bed.

This file creates a material property object for various properties of a given adsorbent. These properties are then used in other material property files and/or kernels to calculate information such as linear velocity, mechanical dispersion, or any adsorption kinetic parameters.

## Note

Currently, we do not couple with adsorption kinetics, so this file is only used in conjunction with the linear velocity and mechanical dispersion properties.

# Warning

THIS KERNEL IS INCOMPLETE! ONLY USED FOR DATA STORAGE FOR PELLET DENSITY AND HEAT CAPACITY!

## Author

Austin Ladshaw

## Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

This file creates a material property object for various properties of the fixed bed. Those properties are used in conjunction with other kernels and materials to establish information such as heat transfer coefficients, conductivities, and size parameters.

## Warning

THIS KERNEL IS INCOMPLETE! ONLY USED FOR DATA STORAGE FOR VARIOUS BED PARAMETERS!

## **Author**

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file AdsorbentProperties.h.

## 5.1.2 Function Documentation

5.1.2.1 template<> InputParameters validParams< AdsorbentProperties > ( )

## 5.2 BedHeatAccumulation.h File Reference

```
#include "TimeDerivative.h"
```

## Classes

· class BedHeatAccumulation

## **Functions**

template<>
 InputParameters validParams< BedHeatAccumulation > ()

```
5.2.1 Function Documentation
5.2.1.1 template<> InputParameters validParams< BedHeatAccumulation > ( )
5.3 BedProperties.h File Reference
#include "Material.h"
Classes
   • class BedProperties
         BedProperties class object inherits from Material object.
Functions
    template<>
      InputParameters validParams< BedProperties > ()
5.3.1 Function Documentation
5.3.1.1 template<> InputParameters validParams< BedProperties > ( )
5.4 BedWallHeatTransfer.h File Reference
#include "Kernel.h"
Classes
    · class BedWallHeatTransfer
Functions
    template<>
      InputParameters validParams< BedWallHeatTransfer > ()
5.4.1 Function Documentation
5.4.1.1 template<> InputParameters validParams< BedWallHeatTransfer > ( )
5.5 ColumnTemperaturelC.h File Reference
#include "InitialCondition.h"
```

# Classes

• class ColumnTemperatureIC

## **Functions**

```
    template<>
        InputParameters validParams< ColumnTemperatureIC > ()
```

## 5.5.1 Function Documentation

```
5.5.1.1 template<> InputParameters validParams< ColumnTemperatureIC > ( )
```

## 5.6 ConcentrationIC.h File Reference

```
#include "InitialCondition.h"
```

## Classes

class ConcentrationIC

## **Functions**

template<>
 InputParameters validParams< ConcentrationIC > ()

## 5.6.1 Function Documentation

```
5.6.1.1 template <> InputParameters validParams < ConcentrationIC > ( )
```

# 5.7 DGAdvection.h File Reference

Discontinous Galerkin kernel for advection.

```
#include "DGKernel.h"
#include <cmath>
```

# Classes

class DGAdvection

DGAdvection class object inherits from DGKernel object.

## **Functions**

template<>
 InputParameters validParams< DGAdvection > ()

## 5.7.1 Detailed Description

Discontinous Galerkin kernel for advection. This file creates a discontinuous Galerkin kernel for advection physics in a given domain. It is a generic advection kernel that is meant to be inherited from to make a more specific kernel for a given problem. The physical parameter in this kernel's formulation is a velocity vector. That vector can be built piecewise by the respective x, y, and z components of a velocity field at a given quadrature point.

#### Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

#### **Author**

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGAdvection.h.

## 5.7.2 Function Documentation

```
5.7.2.1 template<> InputParameters validParams< DGAdvection > ( )
```

## 5.8 DGAnisotropicDiffusion.h File Reference

Discontinous Galerkin kernel for anisotropic diffusion.

```
#include "DGKernel.h"
#include <cmath>
```

## Classes

class DGAnisotropicDiffusion

DGAnisotropicDiffusion class object inherits from DGKernel object.

## **Functions**

template<>
 InputParameters validParams< DGAnisotropicDiffusion > ()

## 5.8.1 Detailed Description

Discontinous Galerkin kernel for anisotropic diffusion. This file creates a discontinous Galerkin kernel for anisotropic diffusion in a given domain. It is a generic diffusion kernel that is meant to be inherited from to make a more specific kernel for a given problem. The physical parameter in this kernel's formulation is a diffusion tensor. That tensor can be built piecewise by the respective components of the tensor at a given quadrature point.

#### Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

## **Author**

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGAnisotropicDiffusion.h.

## 5.8.2 Function Documentation

5.8.2.1 template<> InputParameters validParams< DGAnisotropicDiffusion > ( )

## 5.9 DGColumnHeatAdvection.h File Reference

Discontinous Galerkin kernel for energy advection in a fixed-bed column.

```
#include "DGAdvection.h"
```

## Classes

class DGColumnHeatAdvection

DGColumnHeatAdvection class object inherits from DGAdvection object.

## Macros

#define DGCOLUMNHEATADVECTION\_H

# **Functions**

template<>
 InputParameters validParams< DGColumnHeatAdvection > ()

## 5.9.1 Detailed Description

Discontinous Galerkin kernel for energy advection in a fixed-bed column. This file creates a discontinous Galerkin kernel for the advective heat transfer in a fixed-bed column. The advection portion of the energy transport equations involves the linear velocity in the system, as well as the gas density and gas heat capacity. Those parameters are given as material properties for the system.

Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

## **Author**

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGColumnHeatAdvection.h.

- 5.9.2 Macro Definition Documentation
- 5.9.2.1 #define DGCOLUMNHEATADVECTION\_H

Definition at line 47 of file DGColumnHeatAdvection.h.

- 5.9.3 Function Documentation
- 5.9.3.1 template<> InputParameters validParams< DGColumnHeatAdvection > ( )
- 5.10 DGColumnHeatDispersion.h File Reference

Discontinous Galerkin kernel for energy dispersion in a fixed-bed column.

```
#include "DGAnisotropicDiffusion.h"
```

## Classes

class DGColumnHeatDispersion

DGColumnHeatDispersion class object inherits from DGAnisotropicDiffusion object.

#### **Macros**

#define DGCOLUMNHEATDISPERSION H

## **Functions**

template<>
 InputParameters validParams< DGColumnHeatDispersion > ()

## 5.10.1 Detailed Description

Discontinous Galerkin kernel for energy dispersion in a fixed-bed column. This file creates a discontinous Galerkin kernel for the dispersive heat transfer in a fixed-bed column. The dispersion portion of the energy transport equations involves the thermal conductivity in the system. That parameter is calculated in a material properties file and passed into this object for use the construction of residuals and Jacobians.

#### Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

## **Author**

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGColumnHeatDispersion.h.

5.10.2 Macro Definition Documentation

5.10.2.1 #define DGCOLUMNHEATDISPERSION\_H

Definition at line 47 of file DGColumnHeatDispersion.h.

5.10.3 Function Documentation

5.10.3.1 template <> InputParameters validParams < DGColumnHeatDispersion > ( )

## 5.11 DGColumnMassAdvection.h File Reference

Discontinous Galerkin kernel for mass advection in a fixed-bed column.

```
#include "DGAdvection.h"
```

#### Classes

· class DGColumnMassAdvection

DGColumnMassAdvection class object inherits from DGAdvection object.

## **Functions**

template<>
 InputParameters validParams< DGColumnMassAdvection > ()

## 5.11.1 Detailed Description

Discontinous Galerkin kernel for mass advection in a fixed-bed column. This file creates a discontinous Galerkin kernel for the advective mass transfer in a fixed-bed column. The advection portion of the mass transport equations involves the linear velocity in the system. That parameter is given as material property.

## Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

## **Author**

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGColumnMassAdvection.h.

## 5.11.2 Function Documentation

5.11.2.1 template <> InputParameters validParams < DGColumnMassAdvection > ( )

## 5.12 DGColumnMassDispersion.h File Reference

Discontinous Galerkin kernel for mass dispersion in a fixed-bed column.

#include "DGAnisotropicDiffusion.h"

#### Classes

· class DGColumnMassDispersion

DGColumnMassDispersion class object inherits from DGAnisotropicDiffusion object.

## **Functions**

template<>
 InputParameters validParams< DGColumnMassDispersion > ()

## 5.12.1 Detailed Description

Discontinous Galerkin kernel for mass dispersion in a fixed-bed column. This file creates a discontinous Galerkin kernel for the dispersive mass transfer in a fixed-bed column. The dispersion portion of the mass transport equations involves the molecular diffusivity of a species in the system, as well as the overall axial dispersion of material caused by mechanical mixing and molecular diffusion. Those parameters are calculated in a material properties file and passed into this object for use the construction of residuals and Jacobians.

#### Note

Any DG kernel under DGOSPREY will have a cooresponding G kernel (usually of same name) that must be included with the DG kernel in the input file. This is because the DG finite element method breaks into several different residual pieces, only a handful of which are handled by the DG kernel system and the other parts must be handled by the standard Galerkin system. This my be due to some legacy code in MOOSE. I am not sure if it is possible to lump all of these actions into a single DG kernel.

## Author

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGColumnMassDispersion.h.

## 5.12.2 Function Documentation

5.12.2.1 template<> InputParameters validParams< DGColumnMassDispersion > ( )

## 5.13 DGColumnWallHeatFluxBC.h File Reference

Boundary Condition kernel for the heat flux across the wall of the fixed-bed column.

```
#include "DGFluxBC.h"
```

## Classes

class DGColumnWallHeatFluxBC

DGColumnWallHeatFluxBC class object inherits from DGFluxBC object.

#### **Functions**

template<>
 InputParameters validParams< DGColumnWallHeatFluxBC > ()

## 5.13.1 Detailed Description

Boundary Condition kernel for the heat flux across the wall of the fixed-bed column. This file creates a boundary condition kernel for the heat flux across the boundary of the walls of the column in the fixed-bed adsorber. It inherits from the DGFluxBC, which acts as a generic flux BC module. This kernel is coupled to the wall temperature variable, as well as the material properties for thermal conductivity and the bed-wall heat transfer coefficient.

This type of boundary condition for DG kernels applies the true flux boundary condition. Alternatively, you can use the "FluxLimitedBC" to impose a Dirichlet boundary condition on the system. Although, in true finite volumes or DG methods, there is no Dirichlet boundary conditions, because the solutions are based on fluxes into and out of cells in a domain.

## Author

Austin Ladshaw

Date

11/20/2015

# Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGColumnWallHeatFluxBC.h.

## 5.13.2 Function Documentation

 $5.13.2.1 \quad template <> Input Parameters\ valid Params < DGColumn Wall Heat Flux BC> (\quad)$ 

## 5.14 DGColumnWallHeatFluxLimitedBC.h File Reference

Boundary Condition kernel for a dirichlet-like boundary condition of heat on the column wall.

```
#include "DGFluxLimitedBC.h"
```

#### Classes

class DGColumnWallHeatFluxLimitedBC

DGColumnWallHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

## **Functions**

template<>
 InputParameters validParams< DGColumnWallHeatFluxLimitedBC > ()

## 5.14.1 Detailed Description

Boundary Condition kernel for a dirichlet-like boundary condition of heat on the column wall. This file creates a boundary condition that mimics a Dirichlet boundary condition for the heat of the column at the wall. A true Dirichlet boundary condition does not exist in DG methods. However, this will create a weak form that imposes a constraint that the solution must be of a certain value at the boundary.

#### **Author**

Austin Ladshaw

## Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGColumnWallHeatFluxLimitedBC.h.

## 5.14.2 Function Documentation

5.14.2.1 template <> InputParameters validParams < DGColumnWallHeatFluxLimitedBC > ( )

## 5.15 DGConcentrationIC.h File Reference

```
#include "InitialCondition.h"
```

## Classes

· class DGConcentrationIC

## **Functions**

template<>
 InputParameters validParams< DGConcentrationIC > ()

#### 5.15.1 Function Documentation

5.15.1.1 template <> InputParameters validParams < DGConcentrationIC > ( )

## 5.16 DGFluxBC.h File Reference

Boundary Condition kernel for the flux across a boundary of the domain.

```
#include "IntegratedBC.h"
#include "libmesh/vector_value.h"
```

#### Classes

class DGFluxBC

DGFluxBC class object inherits from IntegratedBC object.

## **Functions**

template<>
 InputParameters validParams< DGFluxBC > ()

## 5.16.1 Detailed Description

Boundary Condition kernel for the flux across a boundary of the domain. This file creates a generic boundary condition kernel for the flux of material accross a boundary. The flux is based on a diffusivity tensor and a velocity vector and is valid in all directions and all boundaries of a DG method. Since the DG method's flux boundary conditions are essitially the same for input and ouput boundaries, this kernel will check the sign of the flux normal to the boundary and determine automattically whether it is an output or input boundary, then apply the appropriate conditions.

This type of boundary condition for DG kernels applies the true flux boundary condition. Alternatively, you can use the "FluxLimitedBC" to impose a Dirichlet boundary condition on the system. Although, in true finite volumes or DG methods, there is no Dirichlet boundary conditions, because the solutions are based on fluxes into and out of cells in a domain.

## **Author**

Austin Ladshaw

## Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGFluxBC.h.

#### 5.16.2 Function Documentation

```
5.16.2.1 template <> InputParameters validParams < DGFluxBC > ( )
```

## 5.17 DGFluxLimitedBC.h File Reference

Boundary Condition kernel to mimic a Dirichlet BC for DG methods.

```
#include "IntegratedBC.h"
#include "libmesh/vector_value.h"
```

## Classes

· class DGFluxLimitedBC

DGFluxLimitedBC class object inherits from IntegratedBC object.

#### **Functions**

template<>
 InputParameters validParams< DGFluxLimitedBC > ()

## 5.17.1 Detailed Description

Boundary Condition kernel to mimic a Dirichlet BC for DG methods. This file creates a boundary condition kernel to impose a dirichlet-like boundary condition in DG methods. True DG methods do not have Dirichlet boundary conditions, so this kernel seeks to impose a constraint on the inlet of a boundary that is met if the value of a variable at the inlet boundary is equal to the finite element solution at that boundary. When the condition is not met, the residuals get penalyzed until the condition is met.

## **Author**

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGFluxLimitedBC.h.

## 5.17.2 Function Documentation

5.17.2.1 template <> InputParameters validParams < DGFluxLimitedBC > ( )

## 5.18 DGHeatFluxBC.h File Reference

Boundary Condition kernel for the heat flux in and out of the ends of the fixed-bed column.

```
#include "DGFluxBC.h"
```

## Classes

class DGHeatFluxBC

DGHeatFluxBC class object inherits from DGFluxBC object.

#### **Functions**

template<>
 InputParameters validParams< DGHeatFluxBC > ()

## 5.18.1 Detailed Description

Boundary Condition kernel for the heat flux in and out of the ends of the fixed-bed column. This file creates a boundary condition kernel for the heat flux across the boundary of the ends of the column in the fixed-bed adsorber. It inherits from the DGFluxBC, which acts as a generic flux BC module. This kernel is coupled to the column temperature variable, as well as the material properties for thermal conductivity, gas density and heat capacity, and the velocity in the domain.

This type of boundary condition for DG kernels applies the true flux boundary condition. Alternatively, you can use the "FluxLimitedBC" to impose a Dirichlet boundary condition on the system. Although, in true finite volumes or DG methods, there is no Dirichlet boundary conditions, because the solutions are based on fluxes into and out of cells in a domain.

## Author

Austin Ladshaw

## Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGHeatFluxBC.h.

## 5.18.2 Function Documentation

5.18.2.1 template <> InputParameters validParams < DGHeatFluxBC > ( )

## 5.19 DGHeatFluxLimitedBC.h File Reference

Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet.

```
#include "DGFluxLimitedBC.h"
```

## Classes

· class DGHeatFluxLimitedBC

DGHeatFluxLimitedBC class object inherits from DGFluxLimitedBC object.

## **Functions**

template<>
 InputParameters validParams< DGHeatFluxLimitedBC > ()

## 5.19.1 Detailed Description

Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet. This file creates a dirichlet-like boundary condition kernel for the column temperature at the inlet of the system. The outlet boundary condition would remain unchanged from the standard DG form of the boundaries. Only the inlet BC is affected by this file. See FluxLimitedBC.h for more details.

## Author

Austin Ladshaw

## Date

11/20/2015

# Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGHeatFluxLimitedBC.h.

## 5.19.2 Function Documentation

5.19.2.1 template <> InputParameters validParams < DGHeatFluxLimitedBC > ( )

## 5.20 DGMassFluxBC.h File Reference

Boundary Condition kernel for the mass flux in and out of the ends of the fixed-bed column.

```
#include "DGFluxBC.h"
```

## Classes

class DGMassFluxBC

DGMassFluxBC class object inherits from DGFluxBC object.

## **Functions**

template<>
 InputParameters validParams< DGMassFluxBC > ()

## 5.20.1 Detailed Description

Boundary Condition kernel for the mass flux in and out of the ends of the fixed-bed column. This file creates a boundary condition kernel for the mass flux across the boundary of the ends of the column in the fixed-bed adsorber. It inherits from the DGFluxBC, which acts as a generic flux BC module. This kernel is coupled to the column temperature variable, as well as the material properties for thermal conductivity, gas density and heat capacity, and the velocity in the domain.

This type of boundary condition for DG kernels applies the true flux boundary condition. Alternatively, you can use the "FluxLimitedBC" to impose a Dirichlet boundary condition on the system. Although, in true finite volumes or DG methods, there is no Dirichlet boundary conditions, because the solutions are based on fluxes into and out of cells in a domain.

#### **Author**

Austin Ladshaw

Date

11/20/2015

# Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGMassFluxBC.h.

## 5.20.2 Function Documentation

5.20.2.1 template <> InputParameters validParams < DGMassFluxBC > ( )

## 5.21 DGMassFluxLimitedBC.h File Reference

Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet.

```
#include "DGFluxLimitedBC.h"
```

## Classes

class DGMassFluxLimitedBC

DGMassFluxLimitedBC class object inherits from DGFluxLimitedBC object.

## **Functions**

template<>
 InputParameters validParams< DGMassFluxLimitedBC > ()

## 5.21.1 Detailed Description

Boundary Condition kernel to mimic a dirichlet boundary condition at the column inlet. This file creates a dirichlet-like boundary condition kernel for the gas species concentration at the inlet of the system. The outlet boundary condition would remain unchanged from the standard DG form of the boundaries. Only the inlet BC is affected by this file. See FluxLimitedBC.h for more details.

## **Author**

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DGMassFluxLimitedBC.h.

## 5.21.2 Function Documentation

```
5.21.2.1 template <> InputParameters validParams < DGMassFluxLimitedBC > ( )
```

## 5.22 DgospreyApp.h File Reference

Registration object for creating a registering DGOSPREY kernels.

```
#include "MooseApp.h"
```

## Classes

class DgospreyApp

DgospreyApp inherits from the MooseApp object.

## **Functions**

template<>
 InputParameters validParams< DgospreyApp > ()

## 5.22.1 Detailed Description

Registration object for creating a registering DGOSPREY kernels. This file is responsible for registering all DGOSP-REY kernels in the MOOSE framework. Any additional kernel developed under DGOSPREY must be included and registered in this structure. This structure is required by MOOSE in order to have the DGOSPREY objects interact with the underlying MOOSE solvers and finite element framework.

**Author** 

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file DgospreyApp.h.

- 5.22.2 Function Documentation
- 5.22.2.1 template <> InputParameters validParams < DgospreyApp > ( )
- 5.23 DgospreyRevision.h File Reference

Macros

- #define DGOSPREY\_REVISION "git commit 9890b6c on 2014-09-24"
- 5.23.1 Macro Definition Documentation
- 5.23.1.1 #define DGOSPREY\_REVISION "git commit 9890b6c on 2014-09-24"

Definition at line 6 of file base/DgospreyRevision.h.

## 5.24 DgospreyRevision.h File Reference

**Macros** 

- #define DGOSPREY\_REVISION "git commit 5a076c0 on 2015-08-31"
- 5.24.1 Macro Definition Documentation
- 5.24.1.1 #define DGOSPREY\_REVISION "git commit 5a076c0 on 2015-08-31"

Definition at line 6 of file DgospreyRevision.h.

# 5.25 Dispersion.h File Reference

```
#include "Kernel.h"
```

## Classes

class Dispersion

## **Functions**

template<>
 InputParameters validParams< Dispersion > ()

## 5.25.1 Function Documentation

5.25.1.1 template <> InputParameters validParams < Dispersion > ( )

## 5.26 egret.h File Reference

Estimation of Gas-phase pRopErTies.

```
#include "macaw.h"
```

## Classes

struct PURE\_GAS

Data structure holding all the parameters for each pure gas spieces.

struct MIXED GAS

Data structure holding information necessary for computing mixed gas properties.

## Macros

- #define EGRET\_HPP\_
- #define Rstd 8.3144621

Gas Constant in J/K/mol (or) L\*kPa/K/mol (Standard Units)

#define RE3 8.3144621E+3

Gas Constant in cm<sup>\(\)</sup> 3\*kPa/K/mol (Convenient for density calculations)

#define Po 100.0

Standard state pressure (kPa)

#define Cstd(p, T) ((p)/(Rstd\*T))

Calculation of concentration/density from partial pressure (Cstd = mol/L)

#define CE3(p, T) ((p)/(RE3\*T))

Calculation of concentration/density from partial pressure (CE3 = mol/cm<sup>\( \)</sup>3)

#define Pstd(c, T) ((c)\*Rstd\*T)

Calculation of partial pressure from concentration/density (c = mol/L)

#define PE3(c, T) ((c)\*RE3\*T)

Calculation of partial pressure from concentration/density ( $c = mol/cm^{3}$ )

#define Nu(mu, rho) ((mu)/(rho))

Calculation of kinematic viscosity from dynamic viscosity and density (cm\^2/s)

#define PSI(T) (0.873143 + (0.000072375\*T))

Calculation of temperature correction factor for dynamic viscosity.

#define Dp\_ij(Dij, PT) ((PT\*Dij)/Po)

Calculation of the corrected binary diffusivity (cm $^{\land}$ 2/s)

#define D\_ij(MWi, MWj, rhoi, rhoj, mui, muj) ( (4.0 / sqrt(2.0)) \* pow(((1/MWi)+(1/MWj)),0.5) ) / pow((pow((rhoi/(1.385\*mui)),2.0)/MWi),0.25)+ pow((pow((rhoi/(1.385\*mui)),2.0)/MWj),0.25)),2.0 )

Calculation of binary diffusion based on MW, density, and viscosity info (cm\2/s)

#define Mu(muo, To, C, T) (muo \* ((To + C)/(T + C)) \* pow((T/To),1.5) )

Calculation of single species viscosity from Sutherland's Equ. (g/cm/s)

#define D\_ii(rhoi, mui) (1.385\*mui/rhoi)

Calculation of self-diffusivity (cm\^2/s)

• #define ReNum(u, L, nu) (u\*L/nu)

Calculation of the Reynold's Number (-)

#define ScNum(nu, D) (nu/D)

Calculation of the Schmidt Number (-)

• #define FilmMTCoeff(D, L, Re, Sc) ((D/L)\*(2.0 + (1.1\*pow(Re,0.6)\*pow(Sc,0.3))))

Calculation of film mass transfer coefficient (cm/s)

#### **Functions**

int initialize\_data (int N, MIXED\_GAS \*gas\_dat)

Function to initialize the MIXED\_GAS structure based on number of gas species.

int set\_variables (double PT, double T, double us, double L, std::vector< double > &y, MIXED\_GAS \*gas\_-dat)

Function to set the values of the parameters in the gas phase.

int calculate\_properties (MIXED\_GAS \*gas\_dat)

Function to calculate the gas properties based on information in MIXED\_GAS.

## 5.26.1 Detailed Description

Estimation of Gas-phase pRopErTies. egret.cpp

This file is responsible for estimating various temperature, pressure, and concentration dependent parameters to be used in other models for gas phase adsorption, mass transfer, and or mass transport. The goal of this file is to eliminate redundancies in code such that the higher level programs operate more efficiently and cleanly. Calculations made here are based on kinetic theory of gases, ideal gas law, and some emperical models that were developed to account for changes in density and viscosity with changes in temperature between standard temperatures and up to 1000 K.

# Author

Austin Ladshaw

Date

01/29/2015

## Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

Definition in file egret.h.

5.26.2 Macro Definition Documentation

5.26.2.1 #define CE3( p, T) ((p)/(RE3\*T))

Calculation of concentration/density from partial pressure (CE3 = mol/cm<sup>3</sup>)

Definition at line 42 of file egret.h.

5.26.2.2 #define Cstd( p, T) ((p)/(Rstd\*T))

Calculation of concentration/density from partial pressure (Cstd = mol/L)

Definition at line 38 of file egret.h.

5.26.2.3 #define D\_ii( rhoi, mui ) (1.385\*mui/rhoi)

Calculation of self-diffusivity (cm<sup>2</sup>/s)

Definition at line 74 of file egret.h.

5.26.2.4 #define D\_ij( *MWi, MWj, rhoi, rhoj, mui, muj*) ( (4.0 / sqrt(2.0)) \* pow(((1/MWi)+(1/MWj)),0.5)) / pow( (pow((rhoi/(1.385\*mui)),2.0)/MWi),0.25)+ pow((pow((rhoj/(1.385\*muj)),2.0)/MWj),0.25)),2.0)

Calculation of binary diffusion based on MW, density, and viscosity info (cm<sup>2</sup>/s)

Definition at line 66 of file egret.h.

5.26.2.5 #define Dp\_ij(  $\textit{Dij},\ \textit{PT}$  ) ((PT\*Dij)/Po)

Calculation of the corrected binary diffusivity (cm<sup>2</sup>/s)

Definition at line 62 of file egret.h.

5.26.2.6 #define EGRET\_HPP\_

Definition at line 23 of file egret.h.

5.26.2.7 #define FilmMTCoeff( D, L, Re, Sc ) ((D/L)\*(2.0 + (1.1\*pow(Re,0.6)\*pow(Sc,0.3))))

Calculation of film mass transfer coefficient (cm/s)

Definition at line 86 of file egret.h.

5.26.2.8 #define Mu( muo, To, C, T) (muo \* ((To + C)/(T + C)) \* pow((T/To),1.5))

Calculation of single species viscosity from Sutherland's Equ. (g/cm/s)

Definition at line 70 of file egret.h.

5.26.2.9 #define Nu( mu, rho) ((mu)/(rho))

Calculation of kinematic viscosity from dynamic viscosity and density (cm<sup>^2</sup>/2/s)

Definition at line 54 of file egret.h.

5.26.2.10 #define PE3( c, T) ((c)\*RE3\*T)

Calculation of partial pressure from concentration/density (c =  $mol/cm^3$ )

Definition at line 50 of file egret.h.

5.26.2.11 #define Po 100.0

Standard state pressure (kPa)

Definition at line 34 of file egret.h.

5.26.2.12 #define PSI(T) (0.873143 + (0.000072375\*T))

Calculation of temperature correction factor for dynamic viscosity.

Definition at line 58 of file egret.h.

5.26.2.13 #define Pstd( c, T) ((c)\*Rstd\*T)

Calculation of partial pressure from concentration/density (c = mol/L)

Definition at line 46 of file egret.h.

5.26.2.14 #define RE3 8.3144621E+3

Gas Constant in cm<sup>3</sup>\*kPa/K/mol (Convenient for density calculations)

Definition at line 30 of file egret.h.

5.26.2.15 #define ReNum( u, L, nu ) (u\*L/nu)

Calculation of the Reynold's Number (-)

Definition at line 78 of file egret.h.

5.26.2.16 #define Rstd 8.3144621

Gas Constant in J/K/mol (or) L\*kPa/K/mol (Standard Units)

Definition at line 26 of file egret.h.

5.26.2.17 #define ScNum( nu, D) (nu/D)

Calculation of the Schmidt Number (-)

Definition at line 82 of file egret.h.

5.26.3 Function Documentation

5.26.3.1 int calculate\_properties ( MIXED\_GAS \* gas\_dat )

Function to calculate the gas properties based on information in MIXED\_GAS.

This function uses the kinetic theory of gases, combined with other semi-empirical models, to predict and approximate several properties of the mixed gas phase that might be necessary when running any gas dynamical simulation. This includes mass and energy transfer equations, as well as adsorption kinetics in porous adsorbents.

5.26.3.2 int initialize\_data ( int N, MIXED\_GAS \* gas\_dat )

Function to initialize the MIXED\_GAS structure based on number of gas species.

This function will initialize the sizes of all vector objects in the MIXED\_GAS structure based on the number of gas species indicated by N.

5.26.3.3 int set\_variables ( double PT, double T, double us, double L, std::vector < double > & y, MIXED\_GAS \* gas\_dat )

Function to set the values of the parameters in the gas phase.

The gas phase properties are a function of total pressure, gas temperature, gas velocity, characteristic length, and the mole fractions of each species in the gas phase. Prior to calculating the gas phase properties, these parameters must be set and updated as they change.

# **Parameters**

PT	total gas pressure in kPa
T	gas temperature in K

us	gas velocity in cm/s
L	characteristic length in cm (this depends on the particular system)
У	vector of gas mole fractions of each species in the mixture
gas_dat	pointer to the MIXED_GAS data structure

#### 5.27 error.h File Reference

All error types are defined here.

#include <iostream>

#### Macros

#define mError(i)

#### **Enumerations**

enum error\_type {
 generic\_error, file\_dne, indexing\_error, magpie\_reverse\_error,
 simulation\_fail, invalid\_components, invalid\_boolean, invalid\_molefraction,
 invalid\_gas\_sum, invalid\_solid\_sum, scenario\_fail, out\_of\_bounds,
 non\_square\_matrix, dim\_mis\_match, empty\_matrix, opt\_no\_support,
 invalid\_fraction, ortho\_check\_fail, unstable\_matrix, no\_diffusion,
 negative\_mass, negative\_time, matvec\_mis\_match, arg\_matrix\_same,
 singular\_matrix, matrix\_too\_small, invalid\_size, nullptr\_func,
 invalid\_norm, vector\_out\_of\_bounds, zero\_vector, tensor\_out\_of\_bounds,
 non\_real\_edge, nullptr\_error, invalid\_atom, invalid\_proton,
 invalid\_neutron, invalid\_electron, invalid\_valence, string\_parse\_error,
 unregistered\_name, rxn\_rate\_error, invalid\_species, duplicate\_variable,
 missing\_information, invalid\_type, key\_not\_found, anchor\_alias\_dne,
 initial\_error, not\_a\_token, read\_error, invalid\_console\_input }

# **Functions**

· void error (int flag)

Error function customizes output message based on flag.

# 5.27.1 Detailed Description

All error types are defined here. error.cpp

List of names for error type.

This file defines all the different errors that may occur in any simulation in any file. Those errors are recognized by an enum with is then passed through to the error.cpp file that customizes the error message to the console. A macro will also print out the file name and line number where the error occured.

#### **Author**

Austin Ladshaw

Date

04/28/2014

## Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

Definition in file error.h.

```
5.27.2 Macro Definition Documentation
```

```
5.27.2.1 #define mError( i )
```

# Value:

```
{error(i);
std::cout << "Source: " << __FILE__ << "\nLine: " << __LINE__ << std::endl;}</pre>
```

Definition at line 22 of file error.h.

Referenced by Matrix T>:::adjoint(), Matrix T>::cofactor(), Matrix T>::columnExtend(), Matrix T>::columnExtend(), Matrix T>::columnExtend(), Matrix T>::columnProjection(), Matrix T>::columnReplace(), Matrix T>::columnVector-Fill(), Matrix T>::columnProjection(), Matrix T>::columnReplace(), Matrix T>::columnVector-Fill(), Matrix T>:columnVector-Fill(), M

## 5.27.3 Enumeration Type Documentation

```
5.27.3.1 enum error_type
```

List of names for error type.

#### Enumerator

```
generic_error
file_dne
indexing_error
magpie_reverse_error
simulation_fail
invalid_components
invalid_boolean
invalid_molefraction
invalid_gas_sum
invalid_solid_sum
scenario_fail
out_of_bounds
non_square_matrix
dim_mis_match
empty_matrix
```

opt\_no\_support invalid\_fraction ortho\_check\_fail unstable\_matrix no\_diffusion negative\_mass negative\_time matvec\_mis\_match arg\_matrix\_same singular\_matrix matrix\_too\_small invalid\_size nullptr\_func invalid\_norm vector\_out\_of\_bounds zero\_vector tensor\_out\_of\_bounds non\_real\_edge nullptr\_error invalid\_atom invalid\_proton invalid\_neutron invalid\_electron invalid\_valence string\_parse\_error unregistered\_name rxn\_rate\_error invalid\_species duplicate\_variable missing\_information invalid\_type key\_not\_found anchor\_alias\_dne initial\_error not\_a\_token read\_error invalid\_console\_input

Definition at line 28 of file error.h.

# 5.27.4 Function Documentation

# 5.27.4.1 void error ( int flag )

Error function customizes output message based on flag.

This error function is reference in the error.cpp file, but is not called by any other file. Instead, all other files call the mError(i) macro that expands into this error function call plus prints out the file name and line number where the error occured.

#### 5.28 finch.h File Reference

Flux-limiting Implicit Non-oscillatory Conservative High-resolution scheme.

```
#include "macaw.h"
#include "lark.h"
```

#### Classes

struct FINCH DATA

Data structure for the FINCH object.

#### **Enumerations**

enum finch\_solve\_type { FINCH\_Picard, LARK\_Picard, LARK\_PJFNK }

List of enum options to define the solver type in FINCH.

enum finch\_coord\_type { Cartesian, Cylindrical, Spherical }

List of enum options to define the coordinate system in FINCH.

#### **Functions**

double max (std::vector< double > &values)

Function returns the maximum in a list of values.

double min (std::vector< double > &values)

Function returns the minimum in a list of values.

double minmod (std::vector< double > &values)

Function returns the result of the minmod function acting on a list of values.

int uTotal (FINCH\_DATA \*dat)

Function integrates the conserved quantity to return it's total in the domain.

int uAverage (FINCH\_DATA \*dat)

Function integrates the conserved quantity to reture it's average in the domain.

int check\_Mass (FINCH\_DATA \*dat)

Function checks the unp1 vector for negative values and will adjust if needed.

int I\_direct (FINCH\_DATA \*dat)

Function solves the discretized FINCH problem directly by assuming it is linear.

int lark\_picard\_step (const Matrix< double > &x, Matrix< double > &G, const void \*data)

Function to perform the necessary LARK Picard iterative method (not typically used)

int nl picard (FINCH DATA \*dat)

Function to solve the discretized FINCH problem iteratively by assuming it is non-linear.

int setup\_FINCH\_DATA (int(\*user\_callroutine)(const void \*user\_data), int(\*user\_setic)(const void \*user\_data), int(\*user\_timestep)(const void \*user\_data), int(\*user\_preprocess)(const void \*user\_data), int(\*user\_solve)(const void \*user\_data), int(\*user\_setparams)(const void \*user\_data), int(\*user\_discretize)(const void \*user\_data), int(\*user\_bcs)(const void \*user\_data), int(\*user\_res)(const Matrix< double > &x, Matrix< double > &res, const void \*user\_data), int(\*user\_precon)(const Matrix< double > &b, Matrix< double > &p, const void \*user\_data), int(\*user\_postprocess)(const void \*user\_data), int(\*user\_reset)(const void \*user\_data), FINCH\_DATA \*dat, const void \*param\_data)

Function to setup memory and set user defined functions into the FINCH object.

void print2file\_dim\_header (FILE \*Output, FINCH\_DATA \*dat)

Function will print out a dimension header for FINCH output.

void print2file\_time\_header (FILE \*Output, FINCH\_DATA \*dat)

Function will print out a time header for FINCH output.

void print2file\_result\_old (FILE \*Output, FINCH\_DATA \*dat)

Function will print out the old results to the variable u.

void print2file\_result\_new (FILE \*Output, FINCH\_DATA \*dat)

Function will print out the new results to the variable u.

void print2file\_newline (FILE \*Output, FINCH\_DATA \*dat)

Function will force print out a blank line.

void print2file\_tab (FILE \*Output, FINCH\_DATA \*dat)

Function will force print out a tab.

int default\_execution (const void \*user\_data)

Default executioner function for FINCH.

int default ic (const void \*user data)

Default initial conditions function for FINCH.

int default\_timestep (const void \*user\_data)

Default time step function for FINCH.

int default\_preprocess (const void \*user\_data)

Default preprocesses function for FINCH.

int default\_solve (const void \*user\_data)

Default solve function for FINCH.

• int default params (const void \*user data)

Default params function for FINCH.

int minmod\_discretization (const void \*user\_data)

Minmod Discretization function for FINCH.

int vanAlbada\_discretization (const void \*user\_data)

Van Albada Discretization function for FINCH.

int ospre discretization (const void \*user data)

Ospre Discretization function for FINCH.

int default\_bcs (const void \*user\_data)

Default boundary conditions function for FINCH.

int default res (const Matrix < double > &x, Matrix < double > &res, const void \*user data)

Default residual function for FINCH.

int default\_precon (const Matrix< double > &b, Matrix< double > &p, const void \*user\_data)

Default preconditioning function for FINCH.

- int default\_postprocess (const void \*user\_data)
- int default\_reset (const void \*user\_data)

Default reset function for FINCH.

# 5.28.1 Detailed Description

Flux-limiting Implicit Non-oscillatory Conservative High-resolution scheme. finch.cpp

This is a conservative finite differences scheme based on the Kurganov and Tadmoor (2000) MUSCL scheme for non-linear conservation laws. It can solve 1-D conservation law problems in three different coordinate systems: (i) Cartesian - axial, (ii) Cylindrical - radial, and (iii) Spherical - radial. It is the backbone algorithm behind all 1-D PDE problems in the ecosystem software.

The form of the general conservation law problem that FINCH solves is...

```
z^{\wedge}d*R*du/dt = d/dz(z^{\wedge}d*D*du/dz) - d/dz(z^{\wedge}d*v*u) - z^{\wedge}d*k*u + z^{\wedge}d*S
```

where R, D, v, k, and S are the parameters of the problem and d, z, and u are the coordinates, spatial dimension, and conserved quantities, respectively. The parameter R is a retardation coefficient, D is a diffusion coefficient, v is a velocity, k is a reaction coefficient, and S is a forcing function or source/sink term.

FINCH supports the use of both Dirichlet and Neuman boundary conditions as the input/inlet condition and uses the No Flux (or Natural) boundary condition for the output/outlet of the domain. For radial problems, the outlet is always

taken to the the center of the cylindrical or spherical particle. This enforces the symmetry of the problem. For axial problems, the outlet is determined by the sign of the velocity term and is therefore choosen by the routine based on the actual flow direction in the domain.

Parameters of the problem can be coupled to the variable u and also be functions of space and time. The coupling of the parameters with the variable forces the problem to become non-linear, which requires iteration to solve. The default iterative method is a built-in Picard's method. This method is equivalent to an inexact Newton method, because we use the Linear Solve of this system as a weak approximation to the non-linear solve. Generally, this method is sufficient and is the most efficient. However, if a problem is particularly difficult to solve, then we can call some of the non-linear solvers developed in LARK. If PJFNK is used, then the Linear Solve for the FINCH problem is used as the Preconditioner for the Linear Solve in PJFNK.

This algorithm comes packaged with three different slope limiter functions to stabilize the velocity term for highly advectively dominate problems. The available slope limiters are: (i) minmod, (ii) van Albada, and (iii) ospre. By default, the FINCH setup function will set the slope limiter to ospre, because this method provides a reasonable compromise between accuracy and efficiency.

**Slope Limiter Stats:** 

minmod -> Highest Accuracy, Lowest Efficiency van Albada -> Lowest Accuracy, Highest Efficiency

ospre -> Average Accuracy, Average Efficiency

**Author** 

Austin Ladshaw

Date

01/29/2015

# Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

Definition in file finch.h.

5.28.2 Enumeration Type Documentation

5.28.2.1 enum finch\_coord\_type

List of enum options to define the coordinate system in FINCH.

Enumerator

Cartesian
Cylindrical
Spherical

Definition at line 65 of file finch.h.

5.28.2.2 enum finch\_solve\_type

List of enum options to define the solver type in FINCH.

**Enumerator** 

FINCH\_Picard

# LARK\_Picard LARK\_PJFNK

Definition at line 62 of file finch.h.

5.28.3 Function Documentation

5.28.3.1 int check\_Mass ( FINCH\_DATA \* dat )

Function checks the unp1 vector for negative values and will adjust if needed.

This function can be turned off or on in the FINCH\_DATA structure. Typically, you will want to leave this on so that the routine does not return negative values for u. However, if you want to get negative values of u, then turn this option off.

5.28.3.2 int default\_bcs ( const void \* user\_data )

Default boundary conditions function for FINCH.

The default boundary conditions function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and sets the boundary conditions according to the type of problem requested. The input BCs will always be either Neumann or Dirichlet and the output BC will always be a zero flux Neumann BC.

5.28.3.3 int default\_execution ( const void \* user\_data )

Default executioner function for FINCH.

The default executioner function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and calls the preprocesses, solve, postprocesses, checkMass, uTotal, and uAverage functions in that order.

5.28.3.4 int default\_ic ( const void \* user\_data )

Default initial conditions function for FINCH.

The default initial condition function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and sets the initial values of all system parameters according to the given constants in that structure.

5.28.3.5 int default\_params ( const void \* user\_data )

Default params function for FINCH.

The default params function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and sets the values of all parameters at all nodes equal to the values of those parameters at the boundaries.

5.28.3.6 int default\_postprocess ( const void \* user\_data )

The default postprocesses function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and does nothing.

5.28.3.7 int default\_precon ( const Matrix< double > & b, Matrix< double > & p, const void \* user\_data )

Default preconditioning function for FINCH.

The default preconditioning function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and performs a tridiagonal linear solve using a Modified Thomas Algorithm. This preconditioner will solve the linear problem exactly if there is no advective portion of the physics. Additionally, this preconditioner is also used as the basis for forming the default FINCH non-linear iterations and is sufficient for solving most problems.

5.28.3.8 int default\_preprocess ( const void \* user\_data )

Default preprocesses function for FINCH.

The default preprocesses function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and

does nothing.

```
5.28.3.9 int default_res ( const Matrix < double > & x, Matrix < double > & res, const void * user_data )
```

Default residual function for FINCH.

The default residual function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and calls the setparams function (passing the param\_data structure), the discretization function, and the set BCs functions, in that order. It then forms the implicit and explicit side residuals that go into the iterative solver.

```
5.28.3.10 int default_reset ( const void * user_data )
```

Default reset function for FINCH.

The default reset function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and sets all old state parameters and variables to the new state.

```
5.28.3.11 int default_solve ( const void * user_data )
```

Default solve function for FINCH.

The default solve function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and calls the corresponding solution method depending on the users conditions.

```
5.28.3.12 int default_timestep ( const void * user_data )
```

Default time step function for FINCH.

The default time step function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and sets the time step to 1/2 the mesh size or bases the time step off of the CFL condition if the problem is not being solved iteratively and involves an advective portion.

```
5.28.3.13 int l_direct ( FINCH_DATA * dat )
```

Function solves the discretized FINCH problem directly by assuming it is linear.

```
5.28.3.14 int lark_picard_step ( const Matrix < double > & x, Matrix < double > & G, const void * data )
```

Function to perform the necessary LARK Picard iterative method (not typically used)

```
5.28.3.15 double max ( std::vector< double > & values )
```

Function returns the maximum in a list of values.

```
5.28.3.16 double min ( std::vector < double > & values )
```

Function returns the minimum in a list of values.

```
5.28.3.17 double minmod ( std::vector< double > & values )
```

Function returns the result of the minmod function acting on a list of values.

```
5.28.3.18 int minmod_discretization ( const void * user_data )
```

Minmod Discretization function for FINCH.

The minmod discretization function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and discretizes the time and space portion of the problem with 2nd order finite differences and uses the minmod slope limiter function to stabilize the advective physics.

```
5.28.3.19 int nl_picard ( FINCH_DATA * dat )
```

Function to solve the discretized FINCH problem iteratively by assuming it is non-linear.

Note

If the problem is actually linear, then this will solve it in one iteration. So it may be best to always assume the problem is non-linear.

5.28.3.20 int ospre\_discretization ( const void \* user\_data )

Ospre Discretization function for FINCH.

The ospre discretization function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and discretizes the time and space portion of the problem with 2nd order finite differences and uses the ospre slope limiter function to stabilize the advective physics. This is the default discretization function.

5.28.3.21 void print2file\_dim\_header ( FILE \* Output, FINCH\_DATA \* dat )

Function will print out a dimension header for FINCH output.

5.28.3.22 void print2file\_newline ( FILE \* Output, FINCH\_DATA \* dat )

Function will force print out a blank line.

5.28.3.23 void print2file\_result\_new ( FILE \* Output, FINCH\_DATA \* dat )

Function will print out the new results to the variable u.

5.28.3.24 void print2file\_result\_old ( FILE \* Output, FINCH\_DATA \* dat )

Function will print out the old results to the variable u.

5.28.3.25 void print2file\_tab ( FILE \* Output, FINCH\_DATA \* dat )

Function will force print out a tab.

5.28.3.26 void print2file\_time\_header ( FILE \* Output, FINCH\_DATA \* dat )

Function will print out a time header for FINCH output.

5.28.3.27 int setup\_FINCH\_DATA ( int(\*)(const void \*user\_data) user\_callroutine, int(\*)(const void \*user\_data) user\_setic, int(\*)(const void \*user\_data) user\_setic, int(\*)(const void \*user\_data) user\_preprocess, int(\*)(const void \*user\_data) user\_setic, int(\*)(const Matrix< double > &x, Matrix< double

Function to setup memory and set user defined functions into the FINCH object.

This function MUST be called prior to running any FINCH based simulation. However, you are only every required to provide this function with the FINCH\_DATA pointer. It is recommended, however, that you do provide the user\_setparams and param\_data pointers, as these will likely vary significantly from problem to problem.

After the problem is setup in memory, you do not technically have to have FINCH call all of it's own functions. You can write your own executioner, initial conditions, and other functions and decided how and when everything is called. Then just call the solve function in FINCH\_DATA when you want to use the FINCH solver. This is how FINCH is used in SKUA, SCOPSOWL, DOGFISH, and MONKFISH.

## **Parameters**

user_callroutine	function pointer the the call routine function
user_setic	function pointer to set initial conditions for problem
user_timestep	function pointer to set the next time step
user_preprocess	function pointer to setup a preprocess operation
user_solve	function pointer to solve the system of equations

function pointer to set the parameters in the problem (always override this)
function pointer to select discretization scheme for the problem
function pointer to evaluate boundary conditions for the problem
function pointer to evaluate non-linear residuals for the problem
function pointer to perform a linear preconditioning operation
function pointer to setup a postprocess operation
function pointer to reset stateful data for next simulation
pointer to the FINCH_DATA structure
user supplied pointer to a data structure needed in user_setparams

```
5.28.3.28 int uAverage ( FINCH_DATA * dat )
```

Function integrates the conserved quantity to reture it's average in the domain.

```
5.28.3.29 int uTotal ( FINCH_DATA * dat )
```

Function integrates the conserved quantity to return it's total in the domain.

```
5.28.3.30 int vanAlbada_discretization ( const void * user_data )
```

Van Albada Discretization function for FINCH.

The van Albada discretization function for FINCH assumes the user\_data parameter is the FINCH\_DATA structure and discretizes the time and space portion of the problem with 2nd order finite differences and uses the van Albada slope limiter function to stabilize the advective physics.

#### 5.29 flock.h File Reference

FundamentaL Off-gas Collection of Kernels.

```
#include "macaw.h"
#include "egret.h"
#include "finch.h"
#include "lark.h"
#include "skua.h"
#include "scopsowl.h"
#include "magpie.h"
```

# 5.29.1 Detailed Description

FundamentaL Off-gas Collection of Kernels. This is just a .h file that holds all the includes necessary to develop and run simulations for adsorption and/or mass/energy transfer problems for gaseous systems. Include this file into any other project or source code that needs the methods below.

Files Included in FLOCK

macaw.h egret.h finch.h lark.h skua.h scopsowl.h magpie.h

**Author** 

Austin Ladshaw

Date

04/28/2014

# Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

Definition in file flock.h.

# 5.30 FlowProperties.h File Reference

Material Properties kernel that will setup and calculate gas flow properties based on physical characteristics.

```
#include "Material.h"
#include "flock.h"
```

#### Classes

class FlowProperties

FlowProperties class object inherits from Material object.

#### Macros

```
    #define _gas_const 8.3144621
    Gas Law Constant - J/K/mol.
```

#### **Functions**

template<>
 InputParameters validParams< FlowProperties > ()

# 5.30.1 Detailed Description

Material Properties kernel that will setup and calculate gas flow properties based on physical characteristics. This file creates a material property object for the flow properties in the fixed-bed column. The flow properties are calculated based on some dimensional analysis, empirical relationships, and kinetic theory of gases. Those properties are then coupled the with mass and energy kernels in the domain to simulate the dynamic and non-linear behavior in the system.

# Author

Austin Ladshaw

# Date

11/20/2015

# Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file FlowProperties.h.

```
5.30.2 Macro Definition Documentation
5.30.2.1 #define _gas_const 8.3144621
Gas Law Constant - J/K/mol.
Definition at line 45 of file FlowProperties.h.
5.30.3 Function Documentation
5.30.3.1 \quad template <> Input Parameters \ valid Params < Flow Properties > ( \ \ )
5.31 FluidHeatTransfer.h File Reference
#include "Kernel.h"
Classes
    · class FluidHeatTransfer
Functions
    • template<>
      InputParameters validParams< FluidHeatTransfer > ()
5.31.1 Function Documentation
5.31.1.1 \quad template <> Input Parameters \ valid Params < Fluid Heat Transfer > ( \ \ )
5.32 GAdvection.h File Reference
#include "Kernel.h"
Classes

    class GAdvection

Functions
    template<>
      InputParameters validParams < GAdvection > ()
5.32.1 Function Documentation
5.32.1.1 template<> InputParameters validParams< GAdvection > ( )
5.33 GAnisotropicDiffusion.h File Reference
#include "Kernel.h"
```

#### Classes

class GAnisotropicDiffusion

#### **Functions**

```
    template<>
        InputParameters validParams< GAnisotropicDiffusion > ()
```

```
5.33.1 Function Documentation
```

```
5.33.1.1 template <> InputParameters validParams < GAnisotropicDiffusion > ( )
```

# 5.34 GColumnHeatAdvection.h File Reference

```
#include "GAdvection.h"
```

#### Classes

· class GColumnHeatAdvection

#### **Functions**

```
    template<>
        InputParameters validParams< GColumnHeatAdvection > ()
```

#### 5.34.1 Function Documentation

```
5.34.1.1 template<> InputParameters validParams< GColumnHeatAdvection > ( )
```

# 5.35 GColumnHeatDispersion.h File Reference

```
#include "GAnisotropicDiffusion.h"
```

## Classes

· class GColumnHeatDispersion

# **Functions**

```
    template<>
        InputParameters validParams< GColumnHeatDispersion > ()
```

# 5.35.1 Function Documentation

```
5.35.1.1 template<> InputParameters validParams< GColumnHeatDispersion > ( )
```

# 5.36 GColumnMassAdvection.h File Reference

```
#include "GAdvection.h"
```

#### Classes

· class GColumnMassAdvection

# **Functions**

```
    template<>
        InputParameters validParams< GColumnMassAdvection > ()
```

```
5.36.1 Function Documentation
```

```
5.36.1.1 template<> InputParameters validParams< GColumnMassAdvection > ( )
```

# 5.37 GColumnMassDispersion.h File Reference

```
#include "GAnisotropicDiffusion.h"
```

#### Classes

• class GColumnMassDispersion

#### **Functions**

```
    template<>
        InputParameters validParams< GColumnMassDispersion > ()
```

```
5.37.1 Function Documentation
```

```
5.37.1.1 template<> InputParameters validParams< GColumnMassDispersion > ( )
```

# 5.38 HeatConductivity.h File Reference

```
#include "Kernel.h"
```

# Classes

· class HeatConductivity

# **Functions**

```
    template<>
        InputParameters validParams< HeatConductivity > ()
```

## 5.38.1 Function Documentation

```
5.38.1.1 template <> InputParameters validParams < HeatConductivity > ( )
```

# 5.39 lark.h File Reference

Linear Algebra Residual Kernels.

5.39 lark.h File Reference 222

```
#include "macaw.h"
#include <float.h>
```

#### Classes

struct ARNOLDI\_DATA

Data structure for the construction of the Krylov subspaces for a linear system.

struct GMRESLP DATA

Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning.

struct GMRESRP\_DATA

Data structure for the Restarted GMRES algorithm with Right Preconditioning.

struct PCG DATA

Data structure for implementation of the PCG algorithms for symmetric linear systems.

struct BiCGSTAB DATA

Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems.

struct CGS\_DATA

Data structure for the implementation of the CGS algorithm for non-symmetric linear systems.

• struct OPTRANS DATA

Data structure for implementation of linear operator transposition.

struct GCR DATA

Data structure for the implementation of the GCR algorithm for non-symmetric linear systems.

struct GMRESR\_DATA

Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMRESR)

struct KMS\_DATA

Data structure for the implemenation of the Krylov Multi-Space (KMS) Method.

struct PICARD\_DATA

Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems.

struct BACKTRACK\_DATA

Data structure for the implementation of Backtracking Linesearch.

struct PJFNK\_DATA

Data structure for the implementation of the PJFNK algorithm for non-linear systems.

struct NUM\_JAC\_DATA

Data structure to form a numerical jacobian matrix with finite differences.

# Macros

#define MIN\_TOL 1e-15

Minimum Allowable Tolerance for linear and non-linear problems.

# **Enumerations**

enum krylov\_method {
 GMRESLP, PCG, BiCGSTAB, CGS,
 FOM, GMRESRP, GCR, GMRESR }

Enum of definitions for linear solver types in PJFNK.

#### **Functions**

- int update\_arnoldi\_solution (Matrix< double > &x, Matrix< double > &x0, ARNOLDI\_DATA \*arnoldi\_dat)

  Function to update the linear vector x based on the Arnoldi Krylov subspace.
- int arnoldi (int(\*matvec)(const Matrix< double > &v, Matrix< double > &w, const void \*data), int(\*precon)(const Matrix< double > &b, Matrix< double > &p, const void \*data), Matrix< double > &ro, ARNOLDI\_DATA \*arnoldi\_dat, const void \*matvec\_data, const void \*precon\_data)

Function to factor a linear operator into an orthonormal basis and upper Hessenberg matrix.

int gmresLeftPreconditioned (int(\*matvec)(const Matrix< double > &v, Matrix< double > &w, const void \*data), int(\*precon)(const Matrix< double > &b, Matrix< double > &p, const void \*data), Matrix< double > &b, GMRESLP DATA \*gmreslp dat, const void \*matvec data, const void \*precon data)

Function to iteratively solve a non-symmetric, indefinite linear system with GMRESLP.

int fom (int(\*matvec)(const Matrix< double > &v, Matrix< double > &w, const void \*data), int(\*precon)(const Matrix< double > &b, Matrix< double > &b, GMRESLP\_DATA \*gmreslp\_dat, const void \*matvec\_data, const void \*precon\_data)

Function to directly solve a non-symmetric, indefinite linear system with FOM.

int gmresRightPreconditioned (int(\*matvec)(const Matrix< double > &v, Matrix< double > &w, const void \*data), int(\*precon)(const Matrix< double > &b, Matrix< double > &p, const void \*data), Matrix< double > &b, GMRESRP\_DATA \*gmresrp\_dat, const void \*matvec\_data, const void \*precon\_data)

Function to iteratively solve a non-symmetric, indefinite linear system with GMRESRP.

int pcg (int(\*matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void \*data), int(\*precon)(const Matrix< double > &r, Matrix< double > &z, const void \*data), Matrix< double > &b, PCG\_DATA \*pcg\_dat, const void \*matvec\_data, const void \*precon\_data)

Function to iteratively solve a symmetric, definite linear system with PCG.

int bicgstab (int(\*matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void \*data), int(\*precon)(const Matrix< double > &r, Matrix< double > &z, const void \*data), Matrix< double > &b, BiCGSTAB\_DATA \*bicg\_dat, const void \*matvec\_data, const void \*precon\_data)

Function to iteratively solve a non-symmetric, definite linear system with BiCGSTAB.

• int cgs (int(\*matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void \*data), int(\*precon)(const Matrix< double > &r, Matrix< double > &z, const void \*data), Matrix< double > &b, CGS\_DATA \*cgs\_dat, const void \*matvec\_data, const void \*precon\_data)

Function to iteratively solve a non-symmetric, definite linear system with CGS.

int operatorTranspose (int(\*matvec)(const Matrix< double > &v, Matrix< double > &Av, const void \*data),
 Matrix< double > &r, Matrix< double > &u, OPTRANS\_DATA \*transpose\_dat, const void \*matvec\_data)

Function that is used to perform transposition of a linear operator and results in a new vector  $A^{\wedge}T*r=u$ .

int gcr (int(\*matvec)(const Matrix < double > &x, Matrix < double > &Ax, const void \*data), int(\*precon)(const Matrix < double > &r, Matrix < double > &Mr, const void \*data), Matrix < double > &b, GCR\_DATA \*gcr\_dat, const void \*matvec data, const void \*precon data)

Function to iteratively solve a non-symmetric, definite linear system with GCR.

int gmresrPreconditioner (const Matrix < double > &r, Matrix < double > &Mr, const void \*data)

Function used in conjunction with GMRESR to apply GMRESRP iterations as a preconditioner.

int gmresr (int(\*matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void \*data), int(\*terminal\_precon)(const Matrix< double > &r, Matrix< double > &Mr, const void \*data), Matrix< double > &b, GMRESR\_DATA \*gmresr\_dat, const void \*matvec\_data, const void \*term\_precon\_data)

Function to iteratively solve a non-symmetric, indefinite linear system with GMRESR.

int kmsPreconditioner (const Matrix< double > &r, Matrix< double > &Mr, const void \*data)

Preconditioner function for the Krylov Multi-Space.

• int krylovMultiSpace (int(\*matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void \*data), int(\*terminal\_precon)(const Matrix< double > &r, Matrix< double > &Mr, const void \*data), Matrix< double > &b, KMS\_DATA \*kms\_dat, const void \*matvec\_data, const void \*term\_precon\_data)

Function to iteratively solve a non-symmetric, indefinite linear system with KMS.

• int picard (int(\*res)(const Matrix< double > &x, Matrix< double > &r, const void \*data), int(\*evalx)(const Matrix< double > &x0, Matrix< double > &x, const void \*data), Matrix< double > &x, PICARD\_DATA \*picard dat, const void \*res data, const void \*evalx data)

Function to iteratively solve a non-linear system using the Picard or Fixed-Point method.

int jacvec (const Matrix< double > &v, Matrix< double > &Jv, const void \*data)

Function to form a linear operator of a Jacobian matrix used along with the PJFNK method.

int backtrackLineSearch (int(\*feval)(const Matrix< double > &x, Matrix< double > &F, const void \*data),
 Matrix< double > &Fkp1, Matrix< double > &xkp1, Matrix< double > &pk, double normFk, BACKTRACK-DATA \*backtrack\_dat, const void \*feval\_data)

Function to perform a Backtracking Line Search operation to smooth out convergence of PJFNK.

int pjfnk (int(\*res)(const Matrix< double > &x, Matrix< double > &F, const void \*data), int(\*precon)(const Matrix< double > &r, Matrix< double > &x, PJFNK\_DATA \*pjfnk\_dat, const void \*res\_data, const void \*precon\_data)

Function to perform the PJFNK algorithm to solve a non-linear system of equations.

• int NumericalJacobian (int(\*Func)(const Matrix< double > &x, Matrix< double > &F, const void \*user\_data), const Matrix< double > &x, Matrix< double > &J, int Nx, int Nf, NUM\_JAC\_DATA \*jac\_dat, const void \*user\_data)

Function to form a full numerical Jacobian matrix from a given non-linear function.

#### 5.39.1 Detailed Description

Linear Algebra Residual Kernels. lark.cpp

The functions contained within are designed to solve generic linear and non-linear square systems of equations given a function argument and data from the user. Optionally, the user can also provide a function to return a preconditioning result that will be applied to the system.

Having the user define how the preconditioning is carried out provides two major advantages: (1) we do not need to store and large, sparse preconditioning matrices and instead only store the preconditioned vector result and (2) this allows the user to use any kind of preconditioner they see fit for their problem.

The Arnoldi function is typically not called by the user, but can be if desired. It accepts the function arguments and a residual vector to form an orthonormal basis of the Krylov subspace using the Modified Gram-Schmidt process (aka Arnoldi Iteration). This function is called by GMRES to iteratively solve a linear system of equations. Note that you can use this function to directly solve the linear system as long as that system is not too large. Construction of the basis is expensive, which is why this is used as a sub-function of an iterative method.

The Restarted GMRES function will accept function arguments for a linear system and attempt to solve said system iteratively by constructing an orthonormal basis from the Krylov function. Note that this GMRES function does support restarting and will use restarting by default if the linear system is too large.

Also included is a GMRES algorithm without restarting. This will directly solve the linear system within residual tolerance using a Full Orthogonal basis set of that system. It is equivalent to calling the Krylov method with the k parameter equal to N (i.e. the number of equations). This method is nick-named the Full Othogonalization Method (FOM), although the true FOM algorithm in literature is slightly different.

The PJFNK function will accept function arguments for a square, non-linear system of equations and attempt to solve it iteratively using both the GMRES and Krylov functions with Newton's method to convert the non-linear system into a linear system.

Also built here is a PCG implementation for solving symmetric linear systems. Can also be called by PJFNK if we know that the linear system (i.e. the Jacobian) is symmetric. This algorithm is significantly more efficient than GMRES, but is only valid if the system of equations is symmetric.

Other linear solvers implemented in this work are the BiCGSTAB and CGS algorithms for non-symmetric, positive definite matrices. These algorithms are significantly more computationally efficient than GMRES or FOM. However, they can both break down if the linear system is poorly conditioned. In general, you only want to use these methods if you have preconditioning available and your linear system is very, very large. Otherwise, you will be better suited to using GMRES or FOM.

There is also an implementation of the Generalized Conjugate Residual (GCR) method with and without restarting. This is a GMRES-like method that should give the exact solution within N iterations, where N is the original size of the matrix. Built ontop of the GCR method is a GMRESR (or GMRES Recursive) algorithm that uses GCR as the base method and performs GMRESRP iterations as a preconditioner at each iteration of GCR. This is the only linear

solver that has built-in preconditioning. As a result, it may be slower than other algorithms for simple problems, but generally will have much better convergence behavior and will almost always give better residual reduction, even for hard to solve problems.

We have also developed a novel/experimental iterative method based on the idea of recursively preconditioning a Krylov Subspace with more Krylov Subspaces. We have called with algorithm the Krylov Multi-Space (KMS) method. This algorithm is based on publications from Vorst and Vuik (1991) and Saad (1993). The idea is too use the FGMRES algorithm developed by Saad (1993) and precondition it with more FGMRES steps, i.e., nesting the iterations as Vorst and Vuik (1991) had proposed. In this way, we have created a generalized Krylov Subspace method that has it's own variable preconditioner that can be adjusted depending on the user's desired complexity and convergence rate. If the levels of recursion requested is zero, then this algorithm is exactly equal to GMRES with right preconditioning. If the level is one, then it is FGMRES with a GMRES preconditioner. However, we allow the levels of recursion to reach up to 5, thus allowing us to precondition the preconitioners with more GMRES steps. This can result is significantly faster convergence rates, but is typically only necessary for very large or difficult to solve problems.

NOTE: There are three GMRES implementations: (i) gmresLP, (ii) fom, and (iii) gmresRP. GMRESLP is a restarted GMRES implementation that is left preconditioned and only checks the residual on the outer loops. This may be less efficient than GMRESRP, which can check both outer and inner loop residuals. However, GMRESRP has to use right preconditioning, which also slightly changes the convergence behavior of the linear system. GMRES with left preconditioning and without restarting will just build the full subspace by default, thus solving the system exactly, but may require too much memory. You can do a GMRESRP unrestarted by specifying that the restart parameter be equal to the size of the problem.

**Basic Implementation Details:** 

Linear Solvers -> Solve Ax=b for x

Non-Linear Solvers -> Solve F(x)=0 for x

All implementations require system size to be 2 or greater

Author

Austin Ladshaw

Date

10/14/2014

# Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

Definition in file lark.h.

5.39.2 Macro Definition Documentation

5.39.2.1 #define MIN\_TOL 1e-15

Minimum Allowable Tolerance for linear and non-linear problems.

Definition at line 111 of file lark.h.

5.39.3 Enumeration Type Documentation

5.39.3.1 enum krylov\_method

Enum of definitions for linear solver types in PJFNK.

Enum delineates the available Krylov Subspace methods that can be used to solve the linear sub-problem at each non-linear iteration in a Newton method.

#### Enumerator

**GMRESLP** 

**PCG** 

**BICGSTAB** 

CGS

**FOM** 

**GMRESRP** 

**GCR** 

**GMRESR** 

Definition at line 492 of file lark.h.

#### 5.39.4 Function Documentation

```
5.39.4.1 int arnoldi ( int(*)(const Matrix< double > &v, Matrix< double > &w, const void *data) matvec, int(*)(const Matrix< double > &b, Matrix< double > &p, const void *data) precon, Matrix< double > & r0,

ARNOLDI_DATA * arnoldi_dat, const void * matvec_data, const void * precon_data)
```

Function to factor a linear operator into an orthonormal basis and upper Hessenberg matrix.

This function performs the Arnoldi algorithm to factor a linear operator into an orthonormal basis and upper Hessenberg matrix. Each orthonormal vector is formed using a Modified Gram-Schmidt procedure. When used in conjunction with GMRESLP, user may supply a preconditioning operator to improve convergence of the linear system. However, this function can be used by itself to factor the user's linear operator.

#### **Parameters**

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
r0	user supplied vector to serve as the first basis vector in the orthonormal basis
arnoldi_dat	pointer to the ARNOLDI_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

# Note

int (\*matvec) (const Matrix<double>& v, Matrix<double> &Av, const void \*data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

 $int \ (*precon) \ (const \ {\tt Matrix} < {\tt double} > \& \ b, \ {\tt Matrix} < {\tt double} > \& {\tt Mb}, \ const \ void \ *data)$ 

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

-----

\_\_\_\_\_\_

5.39 lark.h File Reference 227

5.39.4.2 int backtrackLineSearch ( int(\*)(const Matrix< double > &x, Matrix< double > &F, const void \*data)

feval, Matrix< double > & Fkp1, Matrix< double > & xkp1, Matrix< double > & pk, double normFk,

BACKTRACK\_DATA \* backtrack\_dat, const void \* feval\_data )

Function to perform a Backtracking Line Search operation to smooth out convergence of PJFNK.

This function performs a simple backtracking line search operation on the residuals from the PJFNK method. The step size of the non-linear iteration is checked against a level of tolerance for residual reduction, then adjusted down if necessary. This method always starts out with the maximum allowable step size. If the largest step size is fine, then the algorithm does nothing. Otherwise, it iteratively adjusts the step size down, until a suitable step is found. In the case that no suitable step is found, this algorithm will report failure to the PJFNK method and PJFNK will decide whether to continue trying to find a global minimum or report that it is stuck in a local minimum.

#### **Parameters**

feval	user supplied residual function for the non-linear system
Fkp1	vector holding the residuals for the next non-linear step
xkp1	vector holding the solution for the next non-linear step
pk	vector holding the current non-linear search direction
normFk	value of the current non-linear residual
backtrack_dat	pointer to the BACKTRACK_DATA data structure
feval_data	user supplied void pointer to the data structure needed for residual evaluation

#### Note

int (\*feval) (const Matrix<double>& x, Matrix<double> &F, const void \*data)

\_\_\_\_\_

This is a user supplied function for the non-linear residuals. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix x representing the current non-linear variables. Those variables are used to evaluate the users functions and return the residuals in the matrix F. The void pointer data is a data structure provided by the user to hold information the function may need in order to form the residuals.

\_\_\_\_\_

5.39.4.3 int bicgstab ( int(\*)(const Matrix < double > &p, Matrix < double > &Ap, const void \*data) matvec, int(\*)(const Matrix < double > &r, Matrix < double > &z, const void \*data) precon, Matrix < double > & b, BiCGSTAB DATA \* bicg\_dat, const void \* matvec\_data, const void \* precon\_data)

Function to iteratively solve a non-symmetric, definite linear system with BiCGSTAB.

This function iteratively solves a non-symmetric, definite linear system using the Bi-Conjugate Gradient STABilized (BiCGSTAB) method. This is a highly efficient algorithm for solving non-symmetric problems, but will occassionally breakdown and fail. Most common failures are caused by poor preconditioning. Works very well for grid-based linear systems.

#### **Parameters**

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
bicg_dat	pointer to the BiCGSTAB_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

#### Note

int (\*matvec) (const Matrix<double>& v, Matrix<double> &Av, const void \*data)

\_\_\_\_

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified

the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

int (\*precon) (const Matrix<double>& b, Matrix<double> &Mb, const void \*data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

-----

\_\_\_\_\_

5.39.4.4 int cgs ( int(\*)(const Matrix < double > &p, Matrix < double > &Ap, const void \*data) matvec, int(\*)(const Matrix < double > &r, Matrix < double > &z, const void \*data) precon, Matrix < double > & b, CGS\_DATA \* cgs\_dat, const void \* matvec\_data, const void \* precon\_data)

Function to iteratively solve a non-symmetric, definite linear system with CGS.

This function iteratively solves a non-symmetric, definite linear system using the Conjugate Gradient Squared (CGS) method. This is an extremely efficient algorithm for solving non-symmetric problems, but will often breakdown and fail. Most common failures are caused by poor or no preconditioning. Works very will for grid-based linear systems.

#### **Parameters**

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
cgs_dat	pointer to the CGS_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

#### Note

int (\*matvec) (const Matrix<double>& v, Matrix<double> &Av, const void \*data)

\_\_\_\_\_

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

int (\*precon) (const Matrix<double>& b, Matrix<double> &Mb, const void \*data)

\_\_\_\_\_

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

\_\_\_\_\_

5.39.4.5 int fom ( int(\*)(const Matrix < double > &v, Matrix < double > &w, const void \*data) matvec, int(\*)(const Matrix < double > &b, Matrix < double > &b, GMRESLP\_DATA \* gmreslp\_dat, const void \* matvec\_data, const void \* precon\_data )

Function to directly solve a non-symmetric, indefinite linear system with FOM.

This function directly solves a non-symmetric, indefinite linear system using the Full Orthogonalization Method (F-OM). This algorithm is exactly equivalent to GMRESLP without restarting. Therefore, it uses the GMRESLP\_DATA structure and calls the GMRESLP algorithm without using restarts. As a result, it never checks linear residuals. However, this should give the exact solution upon completion, assuming the linear operator is not singular.

#### **Parameters**

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
gmreslp_dat	pointer to the GMRESLP_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

#### Note

int (\*matvec) (const Matrix<double>& v, Matrix<double> &Av, const void \*data)

\_\_\_\_\_

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

int (\*precon) (const Matrix<double>& b, Matrix<double> &Mb, const void \*data)

-----

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

\_\_\_\_\_

5.39.4.6 int gcr ( int(\*)(const Matrix< double > &x, Matrix< double > &Ax, const void \*data) matvec, int(\*)(const Matrix< double > &r, Matrix< double > &Mr, const void \*data) precon, Matrix< double > & b, GCR\_DATA \* gcr\_dat, const void \* matvec\_data, const void \* precon\_data)

Function to iteratively solve a non-symmetric, definite linear system with GCR.

This function iteratively solves a non-symmetric, definite linear system using the Generalized Conjugate Residual (GCR) method. Similar to GMRESRP, this algorithm will construct a growing orthonormal basis set that will eventually form the exact solution to the linear system. However, this algorithm is less efficient than GMRESRP and can suffer breakdowns if the linear system is indefinite.

# Parameters

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
gcr_dat	pointer to the GCR_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

#### Note

int (\*matvec) (const Matrix<double>& v, Matrix<double> &Av, const void \*data)

\_\_\_\_\_

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

int (\*precon) (const Matrix<double>& b, Matrix<double> &Mb, const void \*data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear

operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

\_\_\_\_\_

5.39.4.7 int gmresLeftPreconditioned ( int(\*)(const Matrix< double > &v, Matrix< double > &w, const void \*data) matvec, int(\*)(const Matrix< double > &b, Matrix< double > &b, const void \*data) precon, Matrix< double > & b, GMRESLP\_DATA \* gmreslp\_dat, const void \* matvec\_data, const void \* precon\_data )

Function to iteratively solve a non-symmetric, indefinite linear system with GMRESLP.

This function iteratively solves a non-symmetric, indefinite linear system using the Generalized Minimum RESidual method with Left Preconditioning (GMRESLP). It calls the Arnoldi algorithm to factor a linear operator into an orthonormal basis and upper Hessenberg matrix, then uses that factorization to form an approximation to the linear system. Because this algorithm uses left-side preconditioning, it can only check the linear residuals at the outer iterations.

#### **Parameters**

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
gmreslp_dat	pointer to the GMRESLP_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

#### Note

int (\*matvec) (const Matrix<double>& v, Matrix<double> &Av, const void \*data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

int (\*precon) (const Matrix<double>& b, Matrix<double> &Mb, const void \*data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

\_\_\_\_\_

\_\_\_\_\_

5.39.4.8 int gmresr ( int(\*)(const Matrix< double > &x, Matrix< double > &Ax, const void \*data) matvec, int(\*)(const Matrix< double > &r, Matrix< double > &Mr, const void \*data) terminal\_precon, Matrix< double > & b, GMRESR\_DATA \* gmresr\_dat, const void \* matvec\_data, const void \* term\_precon\_data )

Function to iteratively solve a non-symmetric, indefinite linear system with GMRESR.

This function iteratively solves a non-symmetric, indefinite linear system using the Generalized Minimum RESidual Recursive (GMRESR) method. This algorithm actually uses GCR at the outer iterations, but stabilizes GCR with GMRESRP inner iterations to implicitly form a variable preconditioner to the linear system. As such, this is one of only two methods that inherently includes preconditioning (the other is KMS), without any user supplied preconditioning operator. However, this algorithms is signficantly more computationally expensive than GCR or GMRESRP separately. It should only be used for solving very large or very hard to solve linear systems.

#### **Parameters**

matvec	user supplied linear operator given as an int function
terminal_precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
gmresr_dat	pointer to the GMRESR_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
term_precon	user supplied void pointer to a data structure needed for the precondtioning operator
data	

#### Note

int (\*matvec) (const Matrix<double>& v, Matrix<double> &Av, const void \*data)

\_\_\_\_\_

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

int (\*terminal\_precon) (const Matrix<double>& b, Matrix<double> &Mb, const void \*data)

\_\_\_\_\_

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

-----

5.39.4.9 int gmresRightPreconditioned ( int(\*)(const Matrix< double > &v, Matrix< double > &w, const void \*data) matvec, int(\*)(const Matrix< double > &b, Matrix< double > &b, Const void \*data) precon, Matrix< double > &b, GMRESRP\_DATA \* gmresrp\_dat, const void \* matvec\_data, const void \* precon\_data)

Function to iteratively solve a non-symmetric, indefinite linear system with GMRESRP.

This function iteratively solves a non-symmetric, indefinite linear system using the Generalized Minimum RESidual method with Right Preconditioning (GMRESRP). Because this algorithm uses right preconditioning, it is able to check the linear residuals at both the outer and inner iterations. This may be much for efficient compared to G-MRESLP. In order to check inner residuals, this algorithm has to perform it's own internal Modified Gram-Schmidt procedure and will not call the Arnoldi algorithm.

#### **Parameters**

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
gmresrp_dat	pointer to the GMRESRP_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

#### Note

int (\*matvec) (const Matrix < double > & v, Matrix < double > & Av, const void \*data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

int (\*precon) (const Matrix<double>& b, Matrix<double> &Mb, const void \*data)

.....

5.39 lark.h File Reference 232

\_\_\_\_\_

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

\_\_\_\_\_

5.39.4.10 int gmresrPreconditioner (const Matrix < double > & r, Matrix < double > & Mr, const void \* data)

Function used in conjunction with GMRESR to apply GMRESRP iterations as a preconditioner.

This function is required to take the form of the user supplied preconditioning functions for other iterative methods. However, it cannot be used in conjunction with any other Krylov method. It is only called by the GMRESR function when the preconditioner needs to be applied.

#### **Parameters**

r	vector supplied to the preconditioner to operate on
Mr	vector to hold the result of the preconditioning operation
data	void pointer to the GMRESR_DATA data structure

5.39.4.11 int jacvec ( const Matrix < double > & v, Matrix < double > & Jv, const void \* data )

Function to form a linear operator of a Jacobian matrix used along with the PJFNK method.

This function is used in conjunction with the PJFNK routine to form a linear operator that a Krylov method can operate on. This linear operator is formed from the current residual vector of the non-linear iteration in PJFNK using a finite difference approximation.

Jacobian Linear Operator:  $J*v = (F(x_k + eps*v) - F(x_k)) / eps$ 

# **Parameters**

V	vector to be multiplied by the Jacobian matrix
Jv	storage vector for the result of the Jacobi-vector product
data	void pointer to the PJFNK_DATA data structure holding solver information

5.39.4.12 int kmsPreconditioner (const Matrix < double > & r, Matrix < double > & Mr, const void \* data)

Preconditioner function for the Krylov Multi-Space.

This function is required to take the form of the user supplied preconditioning functions for other iterative methods. However, it cannot be used in conjunction with any other Krylov method. It is only called by the KMS function when the preconditioner needs to be applied.

#### **Parameters**

r	vector supplied to the preconditioner to operate on
Mr	vector to hold the result of the preconditioning operation
data	void pointer to the KMS_DATA data structure

5.39.4.13 int krylovMultiSpace ( int(\*)(const Matrix< double > &x, Matrix< double > &Ax, const void \*data) matvec, int(\*)(const Matrix< double > &r, Matrix< double > &Mr, const void \*data) terminal\_precon, Matrix< double > &b, KMS\_DATA \* kms\_dat, const void \* matvec\_data, const void \* term\_precon\_data)

Function to iteratively solve a non-symmetric, indefinite linear system with KMS.

This function iteratively solves a non-symmetric, indefinite linear system using the Krylov Multi-Space (KMS) method. This algorithm uses GMRESRP at both outer and inner iterations to implicitly form a variable preconditioner

to the linear system. As such, this is one of only two methods that inherently includes preconditioning, without any user supplied preconditioning operator (the other being GMRESR). The advantage to this method over GMRESR is that this method is GMRES at its core, and will therefore never breakdown or need to be stabilized. Additionally, you can call this method and set it's max\_level parameter (see KMS\_DATA) to 0, which will make this algorithm exactly equal to GMRESRP. If the max\_level is set to 1, then this algorithm is exactly FGMRES (Saad, 1993) with the GMRES algorithm as a preconditioner. However, you can set max\_level higher to precondition the preconditioners with more preconditioners. Thus creating a method with any desired complexity or rate of convergence.

#### **Parameters**

matvec	user supplied linear operator given as an int function
terminal_precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
kms_dat	pointer to the KMS_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
term_precon	user supplied void pointer to a data structure needed for the precondtioning operator
data	

#### Note

int (\*matvec) (const Matrix<double>& v, Matrix<double> &Av, const void \*data)

-----

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

int (\*terminal\_precon) (const Matrix<double>& b, Matrix<double> &Mb, const void \*data)

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

\_\_\_\_\_

5.39.4.14 int Numerical Jacobian ( int(\*)(const Matrix < double > &x, Matrix < double > &F, const void \*user\_data) Func, const Matrix < double > & x, Matrix < double > & J, int Nx, int Nf, NUM\_JAC\_DATA \*  $jac_dat$ , const void \*  $user_data$ )

Function to form a full numerical Jacobian matrix from a given non-linear function.

This function uses finite differences to form a full rank Jacobian matrix for a user supplied non-linear function. The Jacobian matrix will be formed at the current state of the non-linear variables x and stored in a full matrix J. Integers Nx and Nf are used to determine the size of the Jacobian matrix.

# **Parameters**

Func	user supplied function for evaluation of the non-linear system
X	matrix holding the current value of the non-linear variables
J	matrix that will store the numerical Jacobian result
Nx	number of non-linear variables in the system
Nf	number of non-linear functions in the system
jac_dat	pointer to the NUM_JAC_DATA data structure
user_data	user supplied void pointer to a data structure used in the non-linear function

5.39 lark.h File Reference 234

5.39.4.15 int operatorTranspose (int(\*)(const Matrix< double > &v, Matrix< double > &v, const void \*data) matvec,

Matrix< double > & r, Matrix< double > & u, OPTRANS DATA \* transpose\_dat, const void \* matvec\_data)

Function that is used to perform transposition of a linear operator and results in a new vector A^T\*r=u.

This function takes a user supplied linear operator and forms the result of that operator transposed and multiplied by a given vector r ( $A^T*r=u$ ). Transposition is accomplised by reordering the transpose operator and multiplying the non-transposed operator by a complete set of orthonormal vectors. The end result gives the ith component of the vector u for each operation ( $u_i = r^T*A*i$ ). Here, i is a vector made from the ith column of the identity matrix. If the linear system if sufficiently large, then this operation may take some time.

#### **Parameters**

matvec	user supplied linear operator given as an int function
r	vector to be multiplied by the transpose of the operator
и	vector to store the result of the operator transposition (u=A^T*r)
transpose_dat	pointer to the OPTRANS_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator

#### Note

int (\*matvec) (const Matrix < double > & v, Matrix < double > & Av, const void \*data)

data structure that the function may need in order to perform the linear operation.

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user

\_\_\_\_

5.39.4.16 int pcg ( int(\*)(const Matrix< double > &p, Matrix< double > &p, const void \*data) matvec, int(\*)(const Matrix< double > &r, Matrix< double > &z, const void \*data) precon, Matrix< double > & b, PCG\_DATA \* pcg\_dat, const void \* matvec\_data, const void \* precon\_data )

Function to iteratively solve a symmetric, definite linear system with PCG.

This function iteratively solves a symmetric, definite linear system using the Preconditioned Conjugate Gradient (PCG) method. The PCG algorithm is optimal in terms of efficiency and residual reduction, but only if the linear system is symmetric. PCG will fail if the linear operator is non-symmetric!

#### **Parameters**

matvec	user supplied linear operator given as an int function
precon	user supplied preconditioning operator given as an int function
b	matrix of boundary conditions in the linear system Ax=b
pcg_dat	pointer to the PCG_DATA data structure
matvec_data	user supplied void pointer to a data structure needed for the linear operator
precon_data	user supplied void pointer to a data structure needed for the precondtioning operator

# Note

int (\*matvec) (const Matrix<double>& v, Matrix<double> &Av, const void \*data)

This is a user supplied function for a linear operator. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix v that will act on the linear operator a modified the matrix entries of Av to form the result of a matrix-vector product. Void pointer data is used to pass any user data structure that the function may need in order to perform the linear operation.

int (\*precon) (const Matrix<double>& b, Matrix<double> &Mb, const void \*data)

\_\_\_\_\_

-----

This is a user supplied function for a preconditioning operator. It has the same form as the above linear operator function and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the original linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

235

-----

5.39.4.17 int picard ( int(\*)(const Matrix < double > &x, Matrix < double > &x, const void \*data) res, int(\*)(const Matrix < double > &x0, Matrix < double > &x, const void \*data) evalx, Matrix < double > & x, PICARD\_DATA \* picard\_dat, const void \* res\_data, const void \* evalx\_data )

Function to iteratively solve a non-linear system using the Picard or Fixed-Point method.

This function iteratively solves a non-linear system using the Picard method. User supplies a residual function and a weak solution form function. The weak form function is used to approximate the next solution vector for the non-linear system and the residual function is used to determine convergence. User also supplies an initial guess to the non-linear system as a matix x, which will also be used to store the solution. This algorithm is very simple and may not be sufficient to solve complex non-linear systems.

#### **Parameters**

res	user supplied function for the non-linear residuals of the system
evalx	user supplied function for the weak form to estimate the next solution
X	user supplied matrix holding the initial guess to the non-linear system
picard_dat	pointer to the PICARD_DATA data structure
res_data	user supplied void pointer to a data structure used for residual evaluations
evalx_data	user supplied void pointer to a data structure used for evaluation of weak form

#### Note

int (\*res) (const Matrix<double>& x, Matrix<double> &F, const void \*data)

This is a user supplied function for the non-linear residuals. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix x representing the current non-linear variables. Those variables are used to evaluate the users functions and return the residuals in the matrix F. The void pointer data is a data structure provided by the user to hold information the function may need in order to form the residuals.

int (\*evalx) (const Matrix<double>& x0, Matrix<double> &x, const void \*data)

This is a user supplied function to approximate the next solution vector x based on the previous solution vector x0. The x0 matrix is passed to this function and must be used to edit the entries of x based on the weak form of the problem. The user is free to define any weak form approximation. Void pointer data is the users data structure that may be used to pass additional information into this function in order to evaluate the weak form.

Example Residual:  $F(x) = x^2 + x - 1$  Goal is to make this function equal zero

Example Weak Form:  $x = 1 - x0^2$  Rearrage residual to form a weak solution

\_\_\_\_\_

5.39.4.18 int pjfnk ( int(\*)(const Matrix< double > &x, Matrix< double > &F, const void \*data) res, int(\*)(const Matrix< double > &r, Matrix< double > &r, Matrix< double > &x, PJFNK\_DATA \* pjfnk\_dat, const void \* res\_data, const void \* precon\_data )

Function to perform the PJFNK algorithm to solve a non-linear system of equations.

This function solves a non-linear system of equations using the Preconditioned Jacobian- Free Newton-Krylov (P-JFNK) algorithm. Each non-linear step of this method results in a linear sub-problem that is solved iteratively with one of the Krylov methods in the krylov\_method enum. User must supplied a residual function that computes the

non-linear residuals of the system given the current state of the variables x. Additionally, the user must also supplied an initial guess to the non-linear system. Optionally, the user may supply a preconditioning function for the linear sub-problem.

Basic Steps: (i) Calc  $F(x_k)$ , (ii) Solve  $J(x_k)*s_k=-F(x_k)$  for  $s_k$ , (iii) Form  $x_kp1=x_k+s_k$ 

#### **Parameters**

res	user supplied residual function for the non-linear system
precon	user supplied preconditioning function for the linear sub-problems
Х	user supplied initial guess and storage location of the solution
pjfnk_dat	pointer to the PJFNK_DATA data structure
res_data	user supplied void pointer to data structure used in residual function
precon_data	user supplied void pointer to data structure used in preconditioning function

#### Note

int (\*res) (const Matrix<double>& x, Matrix<double> &F, const void \*data)

\_\_\_\_

This is a user supplied function for the non-linear residuals. User's function must return an int of 0 upon success and anything else denotes a failure. The function accepts a matrix x representing the current non-linear variables. Those variables are used to evaluate the users functions and return the residuals in the matrix F. The void pointer data is a data structure provided by the user to hold information the function may need in order to form the residuals.

int (\*precon) (const Matrix<double>& b, Matrix<double> &Mb, const void \*data)

This is a user supplied function for a preconditioning operator. It has the same form as the linear operators from the Krylov methods and should have all the same properties. The only difference is that this function must form an approximate matrix inversion on the jacvec linear operator and modify the entries of Mb to represent the result of that approximate matrix inversion. The matrix b is given as the vector that this operator is acting on and the void pointer data is for any user data structure that the operator may need.

\_\_\_\_\_

5.39.4.19 int update\_arnoldi\_solution ( Matrix < double > & x, Matrix < double > & x0, ARNOLDI\_DATA \* arnoldi\_dat )

Function to update the linear vector x based on the Arnoldi Krylov subspace.

This function will update a solution vector x based on the previous solution x0 given the orthonormal basis and upper Hessenberg matrix formed in the Arnoldi algorithm. Updating is automatically called by the GMRESLP function. It is expected that the Arnoldi algorithm has already been called prior to calling this function.

# **Parameters**

X	matrix that will hold the new updated solution to the linear system
x0	matrix that holds the previous solution to the linear system
arnoldi_dat	pointer to the ARNOLDI_DATA data structure

# 5.40 LinearDrivingForce.h File Reference

#include "Kernel.h"

# Classes

• class LinearDrivingForce

# **Functions**

```
    template<>
        InputParameters validParams< LinearDrivingForce > ()
```

# 5.40.1 Function Documentation

```
5.40.1.1 template <> InputParameters validParams < LinearDrivingForce > ( )
```

#### 5.41 macaw.h File Reference

#### MAtrix CAlculation Workspace.

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <vector>
#include <time.h>
#include <float.h>
#include <string>
#include <exception>
#include "error.h"
```

#### Classes

class Matrix< T >

Templated C++ Matrix Class Object (click Matrix to go to function definitions)

## Macros

#define M\_PI 3.14159265358979323846264338327950288
 Value of PI with double precision.

## 5.41.1 Detailed Description

# MAtrix CAlculation Workspace. macaw.cpp

This is a small C++ library that facilitates the use and construction of real matrices using vector objects. The Matrix class is templated so that users are able to work with matrices of any type including, but not limited to: (i) doubles, (ii) ints, (iii) floats, and (iv) even other matrices! Routines and functions are defined for Dense matrix operations. As a result, we typically only use Column Matrices (or Vectors) when doing any actual simulations. However, the development of this class was integral to the development and testing of the Sparse matrix operators in lark.h.

While the primary goal of this object was to define how to operate on real matrices, we could extend this idea to complex matrices as well. For this, we could develop objects that represent imaginary and complex numbers and then create a Matrix of those objects. For this reason, the matrix operations here are all templated to abstract away the specificity of the type of matrix being operated on.

#### **Author**

Austin Ladshaw

Date

01/07/2014

# Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

Definition in file macaw.h.

```
5.41.2 Macro Definition Documentation
```

5.41.2.1 #define M\_PI 3.14159265358979323846264338327950288

Value of PI with double precision.

Definition at line 43 of file macaw.h.

Referenced by Matrix< T >::IntegralTotal().

# 5.42 magpie.h File Reference

Multicomponent Adsorption Generalized Procedure for Isothermal Equilibria.

```
#include "lmcurve.h"
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <vector>
#include <time.h>
#include <float.h>
#include <string>
#include "error.h"
```

# Classes

struct GSTA\_DATA

GSTA Data Structure.

struct mSPD DATA

MSPD Data Structure.

struct GPAST\_DATA

GPAST Data Structure.

struct SYSTEM\_DATA

System Data Structure.

struct MAGPIE\_DATA

MAGPIE Data Structure.

#### Macros

#define DBL\_EPSILON 2.2204460492503131e-016

Machine precision value used for approximating gradients.

#define Z 10.0

Surface coordination number used in the MSPD activity model.

#define A 3.13E+09

Corresponding van der Waals standard area for our coordination number (cm<sup>\(\chi\)</sup>2/mol)

#define V 18.92

Corresponding van der Waals standard volume for our coordination number (cm<sup>\(\circ\)</sup> 3/mol)

• #define Po 100.0

Standard State Pressure - Units: kPa.

#define R 8.3144621

Gas Constant - Units: J/(K\*mol) = kB \* Na.

• #define Na 6.0221413E+23

Avagadro's Number - Units: molecules/mol.

• #define kB 1.3806488E-23

Boltzmann's Constant - Units: J/K.

#define shapeFactor(v\_i) ( ( (Z - 2) \* v\_i ) / (Z \* V ) ) + (2 / Z )

This macro replaces all instances of shapeFactor(#) with the following single line calculation.

#define InKo(H, S, T) -( H / ( R \* T ) ) + ( S / R )

This macro calculates the natural log of the dimensionless isotherm parameter.

#define He(qm, K1, m) ( qm \* K1 ) / ( m \* Po )

This macro calculates the Henry's Coefficient for the ith component.

#### **Functions**

• double qo (double po, const void \*data, int i)

Function computes the result of the GSTA isotherm for the ith species.

double dq\_dp (double p, const void \*data, int i)

Function computes the derivative of the GSTA model with respect to partial pressure.

double q\_p (double p, const void \*data, int i)

Function computes the ratio between the adsorbed amount and partial pressure for the GSTA isotherm.

double PI (double po, const void \*data, int i)

Function computes the spreading pressure integral of the ith species.

• double Qst (double po, const void \*data, int i)

Function computes the heat of adsorption based on the ith species GSTA parameters.

double eMax (const void \*data, int i)

Function to approximate the maximum lateral energy term for the ith species.

• double Inact mSPD (const double \*par, const void \*data, int i, volatile double PI)

Function to evaluate the MSPD activity coefficient for the ith species.

double grad\_mSPD (const double \*par, const void \*data, int i)

Function to approximate the derivative of the MSPD activity model with spreading pressure.

double qT (const double \*par, const void \*data)

Function to calculate the total adsorbed amount (mol/kg) for the mixed surface phase.

void initialGuess\_mSPD (double \*par, const void \*data)

Function to provide an initial guess to the unknown parameters being solved for in GPAST.

• void eval\_po\_PI (const double \*par, int m\_dat, const void \*data, double \*fvec, int \*info)

Function used with Imfit to evaluate the reference state pressure of a species based on spreading pressure.

void eval\_po\_qo (const double \*par, int m\_dat, const void \*data, double \*fvec, int \*info)

Function used with Imfit to evaluate the reference state pressure of a species based on that species isotherm.

• void eval\_po (const double \*par, int m\_dat, const void \*data, double \*fvec, int \*info)

Function used with Imfit to evaluate the reference state pressure of a species based on a sub-system.

void eval\_eta (const double \*par, int m\_dat, const void \*data, double \*fvec, int \*info)

Function used with Imfit to evaluate the binary interaction parameters for each unique species pair.

• void eval\_GPAST (const double \*par, int m\_dat, const void \*data, double \*fvec, int \*info)

Function used with Imfit to solve the GPAST system of equations.

int MAGPIE (const void \*data)

Function to call all sub-routines to solve a MAGPIE/GPAST problem at a given temperature and pressure.

# 5.42.1 Detailed Description

Multicomponent Adsorption Generalized Procedure for Isothermal Equilibria. magpie.cpp

This file contains all functions and routines associated with predicting isothermal adsorption equilibria from only single component isotherm information. The basis of the model is the Adsorbed Solution Theory developed by Myers and Prausnitz (1965). Added to that base model is a procedure by which we can predict the non-idealities present at the surface phase by solving a closed system of equations involving the activity model.

For more details on this procedure, check out our publication in AIChE where we give a fully feature explaination of our Generalized Predictive Adsorbed Solution Theory (GPAST).

Reference: Ladshaw, A., Yiacoumi, S., and Tsouris, C., "A generalized procedure for the prediction of multicomponent adsorption equilibria", AlChE J., vol. 61, No. 8, p. 2600-2610, 2015.

MAGPIE represents a special case of the more general GPAST procedure, wherin the isotherm for each species is respresent by the GSTA isotherm (see gsta\_opt.h) and the activity model for non-ideality at the adsorbent surface is a Modified Spreading Pressure Dependent (MSPD) model. See the above paper reference for more details.

**Author** 

Austin Ladshaw

Date

12/17/2013

# Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

Definition in file magpie.h.

# 5.42.2 Macro Definition Documentation

5.42.2.1 #define A 3.13E+09

Corresponding van der Waals standard area for our coordination number (cm<sup>2</sup>/mol)

Definition at line 56 of file magpie.h.

Referenced by Matrix < T >::inverse(), Matrix < T >::ladshawSolve(), and Matrix < T >::tridiagonalSolve().

5.42.2.2 #define DBL\_EPSILON 2.2204460492503131e-016

Machine precision value used for approximating gradients.

Definition at line 48 of file magpie.h.

5.42.2.3 #define He( qm, K1, m)(qm \* K1)/(m \* Po)

This macro calculates the Henry's Coefficient for the ith component.

Definition at line 91 of file magpie.h.

5.42.2.4 #define kB 1.3806488E-23

Boltzmann's Constant - Units: J/K.

Definition at line 76 of file magpie.h.

5.42.2.5 #define InKo( H, S, T)-(H/(R\*T))+(S/R)

This macro calculates the natural log of the dimensionless isotherm parameter.

Definition at line 86 of file magpie.h.

5.42.2.6 #define Na 6.0221413E+23

Avagadro's Number - Units: molecules/mol.

Definition at line 72 of file magpie.h.

5.42.2.7 #define Po 100.0

Standard State Pressure - Units: kPa.

Definition at line 64 of file magpie.h.

5.42.2.8 #define R 8.3144621

Gas Constant - Units: J/(K\*mol) = kB \* Na.

Definition at line 68 of file magpie.h.

5.42.2.9 #define shapeFactor( $v_i$ )(((Z-2) \*  $v_i$ )/(Z \* V))+(2/Z)

This macro replaces all instances of shapeFactor(#) with the following single line calculation.

Definition at line 81 of file magpie.h.

5.42.2.10 #define V 18.92

Corresponding van der Waals standard volume for our coordination number (cm<sup>3</sup>/mol)

Definition at line 60 of file magpie.h.

5.42.2.11 #define Z 10.0

Surface coordination number used in the MSPD activity model.

Definition at line 52 of file magpie.h.

5.42.3 Function Documentation

5.42.3.1 double dq\_dp ( double p, const void \* data, int i)

Function computes the derivative of the GSTA model with respect to partial pressure.

This function just computes the result of the derivative of GSTA isotherm model for the ith species at the given the partial pressure p.

## **Parameters**

р	partial pressure in kPa at which to evaluate the GSTA model
data	void pointer to the MAGPIE_DATA data structure
i	index of the gas species for which the GSTA model is being evaluated

5.42.3.2 double eMax ( const void \* data, int i )

Function to approximate the maximum lateral energy term for the ith species.

The function attempts to approximate the maximum lateral energy term for the ith species. This is not a true maximum, but a cheaper estimate. Value being computed is used to shift the geometric mean and formulate the average cross-lateral energy term between species i and j.

5.42.3.3 void eval\_eta ( const double \* par, int  $m_{\perp}dat$ , const void \* data, double \* fvec, int \* info )

Function used with Imfit to evaluate the binary interaction parameters for each unique species pair.

This function is used to estimate the binary interaction parameters for all species pairs in a given sub-system. Those parameters are then stored for later used when evaluating the activity coefficients for the overall mixture. User does not need to call this function. GPAST will call automatically when needed.

#### **Parameters**

par	list of parameters representing variables to be solved for in GPAST
m_dat	number of functions/variables in the GPAST system of equations
data	void pointer for the MAGPIE_DATA data structure
fvec	list of residuals formed by the functions in GPAST
info	integer flag variable used in the Imfit routine

5.42.3.4 void eval\_GPAST ( const double \* par, int  $m_{-}$ dat, const void \* data, double \* fvec, int \* info )

Function used with Imfit to solve the GPAST system of equations.

This function is used after having calculated and stored all necessary information to solve a closed form GPAST system of equations. User does not need to call this function. GPAST will call automatically when needed.

## **Parameters**

par	list of parameters representing variables to be solved for in GPAST
m_dat	number of functions/variables in the GPAST system of equations
data	void pointer for the MAGPIE_DATA data structure
fvec	list of residuals formed by the functions in GPAST
info	integer flag variable used in the Imfit routine

5.42.3.5 void eval\_po ( const double \* par, int m\_dat, const void \* data, double \* fvec, int \* info )

Function used with Imfit to evaluate the reference state pressure of a species based on a sub-system.

This function is used to approximate reference state pressures based on the spreading pressure of a sub-system in GPAST. The sub-system will be one of the unique binary systems that exist in the overall mixed gas system. User does not need to call this function. GPAST will call automatically when needed.

## **Parameters**

par	list of parameters representing variables to be solved for in GPAST
m_dat	number of functions/variables in the GPAST system of equations
data	void pointer for the MAGPIE_DATA data structure
fvec	list of residuals formed by the functions in GPAST
info	integer flag variable used in the Imfit routine

5.42.3.6 void eval\_po\_PI ( const double \* par, int m\_dat, const void \* data, double \* fvec, int \* info )

Function used with Imfit to evaluate the reference state pressure of a species based on spreading pressure.

This function is used inside of the MSPD activity model to calculate the reference state pressure of a particular species at a given spreading pressure for the system. User does not need to call this function. GPAST will call

automatically when needed.

#### **Parameters**

par	list of parameters representing variables to be solved for in GPAST
m_dat	number of functions/variables in the GPAST system of equations
data	void pointer for the MAGPIE_DATA data structure
fvec	list of residuals formed by the functions in GPAST
info	integer flag variable used in the Imfit routine

5.42.3.7 void eval\_po\_qo ( const double \* par, int  $m_{\perp}dat$ , const void \* data, double \* fvec, int \* info )

Function used with Imfit to evaluate the reference state pressure of a species based on that species isotherm.

This function is used to evaluate the partial pressure or reference state pressure for a particular species given single-component adsorbed amount. User does not need to call this function. GPAST will call automatically when needed.

## **Parameters**

par	list of parameters representing variables to be solved for in GPAST
m_dat	number of functions/variables in the GPAST system of equations
data	void pointer for the MAGPIE_DATA data structure
fvec	list of residuals formed by the functions in GPAST
info	integer flag variable used in the Imfit routine

5.42.3.8 double grad\_mSPD ( const double \* par, const void \* data, int i)

Function to approximate the derivative of the MSPD activity model with spreading pressure.

This function returns a 2nd order, finite different approximation of the derivative of the MSPD activity model with the spreading pressure. The par argument will either hold the current iterates estimate of spreading pressure or should be passed as null. User does not need to call this function. GPAST will call automatically when needed.

### **Parameters**

par	list of parameters representing variables to be solved for in GPAST
data	void pointer for the MAGPIE_DATA data structure
i	ith species for which we will approximate the activty model gradient

5.42.3.9 void initialGuess\_mSPD ( double \* par, const void \* data )

Function to provide an initial guess to the unknown parameters being solved for in GPAST.

This function intends to provide an initial guess for the unknown values being solved for in the GPAST system. Depending on what type of solve is requested, this algorithm will provide a guess for the adsorbed or gas phase composition.

## **Parameters**

par	list of parameters representing variables to be solved for in GPAST
data	void pointer for the MAGPIE_DATA data structure

5.42.3.10 double lnact\_mSPD ( const double \* par, const void \* data, int i, volatile double PI )

Function to evaluate the MSPD activity coefficient for the ith species.

This function will return the natural log of the ith species activity coefficient using the Modified Spreading Pressure Dependent (MSPD) activity model. The par argument holds the variable values being solved for by GPAST and their contents will change depending on whether we are doing a forward or reverse evaluation. This function should

not be called by the user and will only be called when needed in the GPAST routine.

#### **Parameters**

par	list of parameters representing variables to be solved for in GPAST
data	void pointer for the MAGPIE_DATA data structure
i	ith species that we want to calculate the activity coefficient for
PI	lumped spreading pressure term used in gradient estimations

## 5.42.3.11 int MAGPIE ( const void \* data )

Function to call all sub-routines to solve a MAGPIE/GPAST problem at a given temperature and pressure.

This is the function that a typical user will want to incorporate into their own codes when evaluating adsorption of a gas mixture. Prior to calling this function, all required structures and information in the MAGPIE\_DATA structure must have been properly initialized. After this function has completed it's operations, it will return an integer used to denote a success or failure of the routine. Integers 0, 1, 2, and 3 all denote success. Anything else is considered a failure.

To setup the MAGPIE\_DATA structure correctly, you must reserve space for all vector objects based on the number of gas species in the mixture. In general, you only need to reserve space for the adsorbing species. However, you can also reserve space for non-adsorbing species, but you MUST give a gas/adsorbed mole fraction of the non-adsorbing species 0.0 so that the routine knows to ignore them (very important)!

After setting up the memory for the vector objects, you can intialize information specific to the simulation you want to request. The number of species (N), total pressure (PT) and gas temperature (T) must always be given. You can neglect the non-idealities of the surface phase by setting the Ideal bool to true. This will result in faster calculations, because MAGPIE will just revert down to the Ideal Adsorbed Solution Theory (IAST).

The Recover bool will denote whether we are doing a forward or reverse GPAST evaluation. Forward evaluation is for solving for the composition of the adsorbed phase given the composition of the gas phase (Recover = false). Reverse evaluation is for solve for the composition of the gas phase given the composition of the adsorbed phase (Recover = true).

For a reverse evaluation (Recover = true) you will also need to stipulate whether or not there is a carrier gas (Carrier = true or false). A carrier gas is considered any non-adsorbing species that may be present in the gas phase and contributing to the total pressure in the system.

The parameters that must be initialized for all species include all GSTA\_DATA parameters and the van der Waals volume parameter (v) in the mSPD\_DATA structure. For non-adsorbing species, you can ignore these parameters, but need to set the sites (m) from GSTA\_DATA to 1. GPAST cannot run any evaluations without these parameters being set properly AND set in the same order for all species (i.e., make sure that gpast\_dat[i].qmax corresponds to mspd\_dat[i].v and so on).

Lastly, you need to give either the gas phase or adsorbed phase mole fractions, depending on whether you are going to run a forward or reverse evaluation, respectively. For a forward evaluation, provide the gas mole fractions (y) in GPAST\_DATA for each species (non-adsorbing species should have this value set to 0.0). For a reverse evaluation, provide the adsorbed mole fractions (x) in GPAST\_DATA for each species, as well as the total adsorbed amount (qT) in SYSTEM\_DATA. Again, non-adsorbing species should have their respective phase mole fractions set to 0.0 to exclude them from the simulation. Additionally, if there are non-adsorbing species present, then the Carrier bool in SYSTEM\_DATA must be set to true.

## **Parameters**

data void pointer for the MAGPIE_DATA data structure holding all necessary information
--

5.42.3.12 double PI ( double po, const void \* data, int i)

Function computes the spreading pressure integral of the ith species.

This function uses an analytical solution to the spreading pressure integral with the GSTA isotherm to evaluate and return the value computed by that integral equation.

#### **Parameters**

ро	partial pressure in kPa at which to evaluate the lumped spreading pressure
data	void pointer to the MAGPIE_DATA data structure
i	index of the gas species for which the GSTA model is being evaluated

## 5.42.3.13 double $q_p$ (double p, const void \* data, int i)

Function computes the ratio between the adsorbed amount and partial pressure for the GSTA isotherm.

This function just computes the ratio between the adsorbed amount q (mol/kg) and the partial pressure p (kPa) at the given partial pressure. If p == 0, then this function returns the Henry's Law constant for the isotherm of the ith species.

#### **Parameters**

р	partial pressure in kPa at which to evaluate the GSTA model
data	void pointer to the MAGPIE_DATA data structure
i	index of the gas species for which the GSTA model is being evaluated

## 5.42.3.14 double qo ( double po, const void \* data, int i )

Function computes the result of the GSTA isotherm for the ith species.

This function just computes the result of the GSTA isotherm model for the ith species given the partial pressure po.

### **Parameters**

ро	po partial pressure in kPa at which to evaluate the GSTA model	
data	data void pointer to the MAGPIE_DATA data structure	
i index of the gas species for which the GSTA model is being evaluated		

Referenced by Matrix< T >::IntegralAvg(), and Matrix< T >::IntegralTotal().

5.42.3.15 double Qst ( double po, const void \* data, int i )

Function computes the heat of adsorption based on the ith species GSTA parameters.

This function computes the isosteric heat of adsorption (J/mol) for the GSTA parameters of the ith species.

## **Parameters**

ро	po partial pressure in kPa at which to evaluate the heat of adsorption	
data	void pointer to the MAGPIE_DATA data structure	
i	i index of the gas species for which the GSTA model is being evaluated	

## 5.42.3.16 double qT ( const double \* par, const void \* data )

Function to calculate the total adsorbed amount (mol/kg) for the mixed surface phase.

This function will uses the obtained system parameters from par and estimate the total amount of gases adsorbed to the surface in mol/kg. The user does not need to call this function, since this result will be stored in the SYSTE-M DATA structure.

### **Parameters**

par	ar list of parameters representing variables to be solved for in GPAST	
data	void pointer for the MAGPIE_DATA data structure	

## 5.43 MAGPIE\_Adsorption.h File Reference

Auxillary kernel to calculate adsorption equilibria of a particular gas species in the system.

```
#include "AuxKernel.h"
#include "flock.h"
```

#### Classes

· class MAGPIE Adsorption

Magpie Adsorption class inherits from AuxKernel.

## **Functions**

template<>
 InputParameters validParams< MAGPIE\_Adsorption > ()

## 5.43.1 Detailed Description

Auxillary kernel to calculate adsorption equilibria of a particular gas species in the system. This file is responsible for calculating the adsorption equilibria of a particular species in the system. The MAGPIE object is stored as a material property whose constants are set in the corresponding material property file (see MagpieAdsorbateProperties.h). That information is then used to call the MAGPIE routine to calculate the mixed gas adsorption for a specific species of interest.

Unfortunately, the material property system has recently changed in MOOSE, making this operation much less efficient. Under the new system, all material properties are declared as constants when outside of their respective material property files. This means that in order for me to call the MAGPIE subroutine, which edits values in the MAGPIE object, I have to create a copy of the entire object and have the subroutine act on that copy.

## Author

Austin Ladshaw

### Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file MAGPIE Adsorption.h.

## 5.43.2 Function Documentation

5.43.2.1 template<> InputParameters validParams< MAGPIE\_Adsorption > ( )

## 5.44 MAGPIE\_AdsorptionHeat.h File Reference

Auxillary kernel to calculate heat of adsorption of a particular gas species in the system.

```
#include "AuxKernel.h"
#include "flock.h"
```

### Classes

• class MAGPIE AdsorptionHeat

Magpie Adsorption Heat class inherits from AuxKernel.

### **Functions**

template<>
 InputParameters validParams< MAGPIE\_AdsorptionHeat > ()

## 5.44.1 Detailed Description

Auxillary kernel to calculate heat of adsorption of a particular gas species in the system. This file is responsible for calculating the heat of adsorption of a particular species in the system. The MAGPIE object is stored as a material property whose constants are set in the corresponding material property file (see MagpieAdsorbateProperties.h). That information is then used to call the MAGPIE routine to calculate the mixed gas adsorption for a specific species of interest.

Unfortunately, the material property system has recently changed in MOOSE, making this operation much less efficient. Under the new system, all material properties are declared as constants when outside of their respective material property files. This means that in order for me to call the MAGPIE subroutine, which edits values in the MAGPIE object, I have to create a copy of the entire object and have the subroutine act on that copy.

## **Author**

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file MAGPIE\_AdsorptionHeat.h.

```
5.44.2 Function Documentation
```

5.44.2.1 template <> InputParameters validParams < MAGPIE\_AdsorptionHeat > ( )

## 5.45 MAGPIE\_HeatAccumulation.h File Reference

```
#include "Kernel.h"
```

#### Classes

class MAGPIE HeatAccumulation

### **Functions**

template<>
 InputParameters validParams< MAGPIE\_HeatAccumulation > ()

### 5.45.1 Function Documentation

```
5.45.1.1 template <> InputParameters validParams < MAGPIE HeatAccumulation > ( )
```

### 5.46 MAGPIE\_Perturbation.h File Reference

Auxillary kernel to calculate the perturbed adsorption equilibria of a particular gas species in the system.

```
#include "AuxKernel.h"
#include "flock.h"
```

#### Classes

· class MAGPIE Perturbation

Magpie Perturbation class inherits from AuxKernel.

## **Functions**

template<>
 InputParameters validParams< MAGPIE Perturbation > ()

## 5.46.1 Detailed Description

Auxillary kernel to calculate the perturbed adsorption equilibria of a particular gas species in the system. This file is responsible for calculating the perturbed adsorption equilibria of a particular species in the system. The MAGPIE object is stored as a material property whose constants are set in the corresponding material property file (see MagpieAdsorbateProperties.h). That information is then used to call the MAGPIE routine to calculate the mixed gas perturbed adsorption for a specific species of interest.

The perturbation is used to approximate the strength of adsorption via first order finite difference. That adsorption strength is then loosely coupled to the gaseous species non-linear variable through a retardation coefficient in the mass transport equations. We use loose coupling to improve the efficiency of the solutions for this multi-scale mass transfer problem. Full coupling would result in significant losses in efficiency, or even complete failure to converge. DO NOT TRY FULL COUPLING!

Unfortunately, the material property system has recently changed in MOOSE, making this operation much less efficient. Under the new system, all material properties are declared as constants when outside of their respective material property files. This means that in order for me to call the MAGPIE subroutine, which edits values in the MAGPIE object, I have to create a copy of the entire object and have the subroutine act on that copy.

## Note

We will only use this kernel to approximate the retardation effect of adsorption IF we are neglecting the microscale kinetics of adsorption/mass transfer into the adsorbent pellets. Kinetics coupling will be accomplished in another kernel.

**Author** 

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file MAGPIE\_Perturbation.h.

### 5.46.2 Function Documentation

```
5.46.2.1 template<> InputParameters validParams< MAGPIE_Perturbation > ( )
```

## 5.47 MagpieAdsorbateProperties.h File Reference

Material Properties kernel that will setup and hold all information associated with MAGPIE simulations.

```
#include "Material.h"
#include "flock.h"
```

## Classes

· class MagpieAdsorbateProperties

MagpieAdsorbateProperties class object inherits from Material object.

## **Functions**

template<>
 InputParameters validParams< MagpieAdsorbateProperties > ()

## 5.47.1 Detailed Description

Material Properties kernel that will setup and hold all information associated with MAGPIE simulations. This file creates a material property object for the MAGPIE data structure and associated constants. That information is used in conjunction with the MAGPIE simulation functions (see magpie.h) in order to approximate the adsorption capacities and adsorbed amounts of each gas species in a given system for a given adsorbent.

**Author** 

Austin Ladshaw

Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file MagpieAdsorbateProperties.h.

#### 5.47.2 Function Documentation

5.47.2.1 template<> InputParameters validParams< MagpieAdsorbateProperties > ( )

## 5.48 RetardedTimeDerivative.h File Reference

```
#include "TimeDerivative.h"
```

#### Classes

· class RetardedTimeDerivative

### **Functions**

template<>
 InputParameters validParams< RetardedTimeDerivative > ()

## 5.48.1 Function Documentation

5.48.1.1 template<> InputParameters validParams< RetardedTimeDerivative > ( )

## 5.49 scopsowl.h File Reference

Simultaneously Coupled Objects for Pore and Surface diffusion Operations With Linear systems.

```
#include "egret.h"
#include "skua.h"
```

### Classes

struct SCOPSOWL\_PARAM\_DATA

Data structure for the species' parameters in SCOPSOWL.

struct SCOPSOWL\_DATA

Primary data structure for SCOPSOWL simulations.

## Macros

- #define SCOPSOWL HPP
- #define Dp(Dm, ep) (ep\*ep\*Dm)

Estimate of Pore Diffusivity (cm\^2/s)

#define Dk(rp, T, MW) (9700.0\*rp\*pow((T/MW),0.5))

Estimate of Knudsen Diffusivity (cm $^{\land}$ 2/s)

#define avgDp(Dp, Dk) (pow(((1/Dp)+(1/Dk)),-1.0))

Estimate of Average Pore Diffusion (cm<sup>2</sup>/s)

### **Functions**

void print2file species header (FILE \*Output, SCOPSOWL DATA \*owl dat, int i)

Function to print out the main header for the output file.

void print2file\_SCOPSOWL\_time\_header (FILE \*Output, SCOPSOWL\_DATA \*owl\_dat, int i)

Function to print out the time and space header for the output file.

void print2file\_SCOPSOWL\_header (SCOPSOWL\_DATA \*owl\_dat)

Function to call the species and time header functions.

void print2file SCOPSOWL result old (SCOPSOWL DATA \*owl dat)

Function to print out the old time results to the output file.

void print2file\_SCOPSOWL\_result\_new (SCOPSOWL\_DATA \*owl\_dat)

Function to print out the new time results to the output file.

• double default adsorption (int i, int I, const void \*user data)

Default function for evaluating adsorption and adsorption strength.

double default retardation (int i, int I, const void \*user data)

Default function for evaluating retardation coefficient.

double default\_pore\_diffusion (int i, int I, const void \*user\_data)

Default function for evaluating pore diffusivity.

• double default\_surf\_diffusion (int i, int I, const void \*user\_data)

Default function for evaluating surface diffusion for HOMOGENEOUS pellets.

double default\_effective\_diffusion (int i, int I, const void \*user\_data)

Default function for evaluating effective diffusivity for HOMOGENEOUS pellets.

• double const\_pore\_diffusion (int i, int I, const void \*user\_data)

Constant pore diffusion function for homogeneous or heterogeneous pellets.

 $\bullet \ \ double \ \ default\_film Mass Transfer \ (int \ i, \ const \ void \ *user\_data)$ 

Default function for evaluating the film mass transfer coefficient.

double const\_filmMassTransfer (int i, const void \*user\_data)

Constant film mass transfer coefficient function.

int setup\_SCOPSOWL\_DATA (FILE \*file, double(\*eval\_sorption)(int i, int I, const void \*user\_data), double(\*eval\_retardation)(int i, int I, const void \*user\_data), double(\*eval\_pore\_diff)(int i, int I, const void \*user\_data), double(\*eval\_surface\_diff)(int i, int I, const void \*user\_data), double(\*eval\_surface\_diff)(int i, int I, const void \*user\_data), const void \*user\_data, MIXED\_GAS \*gas\_data, SCOPSOWL\_DATA \*owl\_data)

Setup function to allocate memory and setup function pointers for the SCOPSOWL simulation.

int SCOPSOWL\_Executioner (SCOPSOWL\_DATA \*owl\_dat)

SCOPSOWL executioner function to solve a time step.

int set\_SCOPSOWL\_ICs (SCOPSOWL\_DATA \*owl\_dat)

Function to set the initial conditions for a SCOPSOWL simulation.

int set\_SCOPSOWL\_timestep (SCOPSOWL\_DATA \*owl\_dat)

Function to set the timestep of the SCOPSOWL simulation.

int SCOPSOWL\_preprocesses (SCOPSOWL\_DATA \*owl\_dat)

Function to perform all preprocess SCOPSOWL operations.

int set\_SCOPSOWL\_params (const void \*user\_data)

Function to set the values of all non-linear system parameters during simulation.

int SCOPSOWL\_postprocesses (SCOPSOWL\_DATA \*owl\_dat)

Function to perform all postprocess SCOPSOWL operations.

int SCOPSOWL\_reset (SCOPSOWL\_DATA \*owl\_dat)

Function to reset all stateful information to prepare for next simulation.

int SCOPSOWL (SCOPSOWL\_DATA \*owl\_dat)

Function to progress the SCOPSOWL simulation through time till complete.

## 5.49.1 Detailed Description

Simultaneously Coupled Objects for Pore and Surface diffusion Operations With Linear systems. scopsowl.cpp

This file contains structures and functions associated with modeling adsorption in commercial, bi-porous adsorbents such as zeolites and mordenites. The pore diffusion and mass transfer equations are coupled with adsorption and surface diffusion through smaller crystals embedded in a binder matrix. However, you can also direct this simulation to treat the adsorbent as homogeneous (instead of heterogeneous) in order to model an even greater variety of gaseous adsorption kinetic problems. This object is coupled with either MAGPIE, SKUA, or BOTH depending on the type of simulation requested.

**Author** 

Austin Ladshaw

Date

01/29/2015

## Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

Definition in file scopsowl.h.

5.49.2 Macro Definition Documentation

5.49.2.1 #define avgDp( Dp, Dk ) (pow(((1/Dp)+(1/Dk)),-1.0))

Estimate of Average Pore Diffusion (cm<sup>2</sup>/s)

Definition at line 37 of file scopsowl.h.

5.49.2.2 #define Dk( rp, T, MW ) (9700.0\*rp\*pow((T/MW),0.5))

Estimate of Knudsen Diffusivity (cm<sup>2</sup>/s)

Definition at line 33 of file scopsowl.h.

5.49.2.3 #define Dp( *Dm*, *ep* ) (ep\*ep\*Dm)

Estimate of Pore Diffusivity (cm<sup>2</sup>/s)

Definition at line 29 of file scopsowl.h.

5.49.2.4 #define SCOPSOWL\_HPP\_

Definition at line 26 of file scopsowl.h.

5.49.3 Function Documentation

5.49.3.1 double const\_filmMassTransfer ( int i, const void \* user\_data )

Constant film mass transfer coefficient function.

This function is used when the user wants to specify a constant value for film mass transfer. The value of that coefficient is then set equal to the value of film\_transfer in the SCOPSOWL\_PARAM\_DATA structure.

**Parameters** 

i index for the ith species in the system	
user_data   pointer for the SCOSPOWL_DATA structure	

## 5.49.3.2 double const\_pore\_diffusion ( int i, int l, const void \* user\_data )

Constant pore diffusion function for homogeneous or heterogeneous pellets.

This function should be used if the user wants to specify a constant pore diffusivity. The value of pore diffusion is then set equal to the value of pore\_diffusion in the SCOPSOWL\_PARAM\_DATA structure.

#### **Parameters**

i index for the ith species in the system	
/ index for the Ith node in the macro-scale domain	
user_data pointer for the SCOSPOWL_DATA structure	

## 5.49.3.3 double default\_adsorption ( int i, int I, const void \* user\_data )

Default function for evaluating adsorption and adsorption strength.

This function is called in the preprocesses and postprocesses to estimate the strength of adsorption in the macroscale problem from perturbations. It will use perturbations in either the MAGPIE simulation or SKUA simulation, depending on the type of problem the user is solving.

## **Parameters**

i index for the ith species in the system	
/ index for the lth node in the macro-scale domain	
user_data pointer for the SCOSPOWL_DATA structure	

## 5.49.3.4 double default\_effective\_diffusion ( int i, int I, const void \* user\_data )

Default function for evaluating effective diffusivity for HOMOGENEOUS pellets.

This function is ONLY used if the pellet is determined to be homogeneous. Otherwise, this is replaced by the pore diffusion function. The effective diffusivity is determined by the combination of pore diffusivity and surface diffusivity with adsorption strength in an homogeneous pellet.

## **Parameters**

i index for the ith species in the system	
I index for the Ith node in the macro-scale domain	
user_data pointer for the SCOSPOWL_DATA structure	

## 5.49.3.5 double default\_filmMassTransfer ( int i, const void \* user\_data )

Default function for evaluating the film mass transfer coefficient.

This function is called during the setup of the boundary conditions and is used to estimate the film mass transfer coefficient for the macro-scale problem. The coefficient is calculated according to the kinetic theory of gases (see egret.h).

## **Parameters**

i	i index for the ith species in the system	
user_data	pointer for the SCOSPOWL_DATA structure	

5.49.3.6 double default\_pore\_diffusion ( int i, int l, const void \* user\_data )

Default function for evaluating pore diffusivity.

This function is called during the evaluation of non-linear residuals to more accurately represent non-linearities in the pore diffusion behavior. The pore diffusion is calculated based on kinetic theory of gases (see egret.h) and is adjusted according to the Knudsen Diffusion model and the porosity of the binder material.

#### **Parameters**

i	index for the ith species in the system
/ index for the lth node in the macro-scale domain	
user_data	pointer for the SCOSPOWL_DATA structure

5.49.3.7 double default\_retardation ( int i, int I, const void \* user\_data )

Default function for evaluating retardation coefficient.

This function is called in the preprocesses and postprocesses to estimate the retardation coefficient for the simulation. It is recalculated at every time step to keep track of all changing conditions in the simulation.

#### **Parameters**

i index for the ith species in the system	
/ index for the lth node in the macro-scale domain	
user_data pointer for the SCOSPOWL_DATA structure	

5.49.3.8 double default\_surf\_diffusion ( int i, int l, const void \* user\_data )

Default function for evaluating surface diffusion for HOMOGENEOUS pellets.

This function is ONLY used if the pellet is determined to be homogeneous. Otherwise, this is replaced by the surface diffusion function for the SKUA simulation. The diffusivity is calculated based on the Arrhenius rate expression and then adjusted by the outside partial pressure of the adsorbing species.

## **Parameters**

i	i index for the ith species in the system	
1	/ index for the lth node in the macro-scale domain	
user_data	pointer for the SCOSPOWL_DATA structure	

5.49.3.9 void print2file\_SCOPSOWL\_header ( SCOPSOWL\_DATA \* owl\_dat )

Function to call the species and time header functions.

5.49.3.10 void print2file\_SCOPSOWL\_result\_new ( SCOPSOWL\_DATA \* owl\_dat )

Function to print out the new time results to the output file.

 $5.49.3.11 \quad \text{void print2file\_SCOPSOWL\_result\_old ( SCOPSOWL\_DATA} * \textit{owl\_dat )}$ 

Function to print out the old time results to the output file.

5.49.3.12 void print2file\_SCOPSOWL\_time\_header ( FILE \* Output, SCOPSOWL\_DATA \* owl\_dat, int i )

Function to print out the time and space header for the output file.

5.49.3.13 void print2file\_species\_header ( FILE \* Output, SCOPSOWL\_DATA \* owl\_dat, int i )

Function to print out the main header for the output file.

5.49.3.14 int SCOPSOWL ( SCOPSOWL\_DATA \* owl\_dat )

Function to progress the SCOPSOWL simulation through time till complete.

This function will call the initial conditions, then progressively call the executioner, time step, and reset functions to propagate the simulation in time. As such, this function is primarily used when running a SCOPSOWL simulation by itself and not when coupling it to an other problem.

#### **Parameters**

owl\_dat | pointer to the SCOPSOWL\_DATA structure (must be initialized)

5.49.3.15 int SCOPSOWL\_Executioner ( SCOPSOWL DATA \* owl\_dat )

SCOPSOWL executioner function to solve a time step.

This function will call the preprocess, solver, and postprocess functions to evaluate a single time step in a simulation. All simulation conditions must be set prior to calling this function. This function will typically be the one called from other simulations that will involve a SCOPSOWL evaluation to resolve kinetic coupling.

#### **Parameters**

owl\_dat | pointer to the SCOPSOWL\_DATA structure (must be initialized)

5.49.3.16 int SCOPSOWL\_postprocesses ( SCOPSOWL\_DATA \* owl\_dat )

Function to perform all postprocess SCOPSOWL operations.

This function will update the retardation coefficients based on newly obtained simulation results for the current time step and calculate the average and total amount of adsorption of each species in the domain. Additionally, this function will call the print functions to store simulation results in the output file.

### **Parameters**

owl\_dat | pointer to the SCOPSOWL\_DATA structure (must be initialized)

5.49.3.17 int SCOPSOWL\_preprocesses ( SCOPSOWL\_DATA \* owl\_dat )

Function to perform all preprocess SCOPSOWL operations.

This function will update the boundary conditions and simulation conditions based on the current temperature, pressure, and gas phase composition, which may all vary in time. Since this function is called by the SCOPSOWL\_Executioner, it does not need to be called explicitly by the user.

### **Parameters**

owl\_dat | pointer to the SCOPSOWL\_DATA structure (must be initialized)

5.49.3.18 int SCOPSOWL\_reset ( SCOPSOWL\_DATA \* owl\_dat )

Function to reset all stateful information to prepare for next simulation.

This function will update the stateful information used in SCOPSOWL to prepare the system for the next time step in the simulation. However, because updating the states erases the old state, the user must be absolutely sure that the simulation is ready to be updated. For just running standard simulations, this is not an issue, but in coupling with other simulations it is very important.

## **Parameters**

owl\_dat | pointer to the SCOPSOWL\_DATA structure (must be initialized)

5.49.3.19 int set\_SCOPSOWL\_ICs ( SCOPSOWL\_DATA \* owl\_dat )

Function to set the initial conditions for a SCOPSOWL simulation.

This function will setup the initial conditions of the simulation based on the initial temperature, pressure, and adsorbed molefractions. It assumes that the initial conditions are constant throughout the domain of the problem. This function should only be called once during a simulation.

### **Parameters**

owl_dat	pointer to the SCOPSOWL	DATA structure (must be initialized)

5.49.3.20 int set\_SCOPSOWL\_params ( const void \* user\_data )

Function to set the values of all non-linear system parameters during simulation.

This is the function override for the FINCH setparams function (see finch.h). It will update the values of non-linear parameters in the residuals so that all variables in a species' system are fully coupled.

#### **Parameters**

5.49.3.21 int set\_SCOPSOWL\_timestep ( SCOPSOWL\_DATA \* owl\_dat )

Function to set the timestep of the SCOPSOWL simulation.

This function is used to set the next time step to be used in the SCOPSOWL simulation. A constant time step based on the size of the pellet discretization will be used. Users may want to use a custom time step to ensure that coupled-multi-scale systems are all in sync.

## **Parameters**

owl_dat   pointer to the SCOPSOWL_DATA structure (must be initialized)
--

5.49.3.22 int setup\_SCOPSOWL\_DATA ( FILE \* file, double(\*)(int i, int I, const void \*user\_data) eval\_sorption, double(\*)(int i, int I, const void \*user\_data) eval\_pore\_diff, double(\*)(int i, int I, const void \*user\_data) eval\_filmMT, double(\*)(int i, int I, const void \*user\_data) eval\_surface\_diff, const void \*user\_data, MIXED\_GAS \* gas\_data, SCOPSOWL\_DATA \* owl\_data)

Setup function to allocate memory and setup function pointers for the SCOPSOWL simulation.

This function sets up the memory and function pointers used in SCOPSOWL simulations. User can provide NULL in place of functions for the function pointers and the setup will automatically use just the default settings. However, the user is required to pass the necessary data structure pointers for MIXED\_GAS and SCOPSOWL\_DATA.

## **Parameters**

file	pointer to the output file to print out results	
eval_sorption	pointer to the adsorption evaluation function	
eval_retardation	eval_retardation pointer to the retardation evaluation function	
eval_pore_diff	pointer to the pore diffusion function	
eval_filmMT	= '	
eval_surface_diff		
user_data		
gas_data pointer to the MIXED_GAS structure used to evaluate kinetic gas theory		
owl_data	pointer to the SCOPSOWL_DATA structure	

### 5.50 skua.h File Reference

Surface Kinetics for Uptake by Adsorption.

```
#include "finch.h"
#include "magpie.h"
#include "egret.h"
```

#### Classes

struct SKUA\_PARAM

Data structure for species' parameters in SKUA.

struct SKUA DATA

Data structure for all simulation information in SKUA.

#### Macros

```
    #define SKUA_HPP_
```

 $\bullet \ \ \, \text{\#define D\_inf}(\text{Dref, Tref, B, p, T}) \ ( \ \, \text{Dref} * pow(p+sqrt(\text{DBL\_EPSILON}),(\text{Tref/T})-B) \ ) \\$ 

Empirical correction of diffusivity (um\^2/hr)

#define D\_o(Diff, E, T) ( Diff \* exp(-E/(Rstd\*T)) )

Arrhenius Rate Expression for Diffusivity (um\^2/hr)

• #define D\_c(Diff, phi) ( Diff \* (1.0/((1.0+1.1E-6)-phi) ) )

Approximate Darken Diffusivity Equation (um\^2/hr)

### **Functions**

void print2file\_species\_header (FILE \*Output, SKUA\_DATA \*skua\_dat, int i)

Function to print out the species' headers to output file.

void print2file\_SKUA\_time\_header (FILE \*Output, SKUA\_DATA \*skua\_dat, int i)

Function to print out time and space headers to output file.

void print2file\_SKUA\_header (SKUA\_DATA \*skua\_dat)

Function calls the other header functions to establish output file structure.

void print2file\_SKUA\_results\_old (SKUA\_DATA \*skua\_dat)

Function to print out the old time step simulation results to the output file.

void print2file\_SKUA\_results\_new (SKUA\_DATA \*skua\_dat)

Function to print out the new time step simulation results to the output file.

double default\_Dc (int i, int I, const void \*data)

Default function for surface diffusivity.

double default\_kf (int i, const void \*data)

Default function for film mass transfer coefficent.

• double const Dc (int i, int I, const void \*data)

Constant surface diffusivity function.

double simple\_darken\_Dc (int i, int I, const void \*data)

Simple Darken model for surface diffusivity.

double theoretical\_darken\_Dc (int i, int I, const void \*data)

Theoretical Darken model for surface diffusivity.

double empirical\_kf (int i, const void \*data)

Empirical function for film mass transfer coefficent.

double const\_kf (int i, const void \*data)

Constant function for film mass transfer coefficent.

int molefractionCheck (SKUA\_DATA \*skua\_dat)

Function to check mole fractions in gas and solid phases for errors.

• int setup\_SKUA\_DATA (FILE \*file, double(\*eval\_Dc)(int i, int I, const void \*user\_data), double(\*eval\_Kf)(int i, const void \*user\_data), const void \*user\_data, MIXED\_GAS \*gas\_data, SKUA\_DATA \*skua\_dat)

Function to setup the function pointers and vector objects in memory to setup the SKUA simulation.

int SKUA\_Executioner (SKUA\_DATA \*skua\_dat)

Function to execute preprocesses, solvers, and postprocesses for a SKUA simulation.

int set SKUA ICs (SKUA DATA \*skua dat)

Function to establish the initial conditions of adsorption in the adsorbent.

int set\_SKUA\_timestep (SKUA\_DATA \*skua\_dat)

Function to establish the time step for the current simulation.

int SKUA\_preprocesses (SKUA\_DATA \*skua\_dat)

Function to perform the necessary preprocess operations before a solve.

int set\_SKUA\_params (const void \*user\_data)

Function to call the diffusivity function during the solve.

int SKUA\_postprocesses (SKUA\_DATA \*skua\_dat)

Function to perform the necessary postprocess operations after a solve.

int SKUA reset (SKUA DATA \*skua dat)

Function to reset the stateful information in SKUA after a simulation.

int SKUA (SKUA\_DATA \*skua\_dat)

Function to iteratively call all execution steps to evolve a simulation through time.

### 5.50.1 Detailed Description

Surface Kinetics for Uptake by Adsorption. skua.cpp

This file contains structures and functions associated with solving the surface diffusion partial differential equations for adsorption kinetics in spherical and/or cylindrical adsorbents. For this system, it is assumed that the pore size is so small that all molecules are confined to movement exclusively on the surface area of the adsorbent. The total amount of adsorption for each species is drive by the MAGPIE model for non-ideal mixed gas adsorption. Spatial and temporal varience in adsorption is caused by a combination of different kinetics between adsorbing species and different adsorption affinities for the surface.

The function for surface diffusion involves four parameters, although not all of these parameters are required to be used. Surface diffusion theoretically varies with temperature according to the Arrhenius rate expression, but we also add in an empirical correction term to account for variations in diffusivity with the partial pressure of the species in the gas phase.

```
D surf = D ref * \exp(-E / (R*T)) * pow(p, (T ref/T) - B)
```

D\_ref is the Reference Diffusivity (um $^2$ 2/hr), E is the activation energy for adsorption (J/mol), R is the gas law constant (J/K/mol), T is the system temperature (K), p is the partial pressure of the adsorbing species (kPa), T\_ref is the Reference Temperature (K), and B is the Affinity constant.

Author

Austin Ladshaw

Date

01/26/2015

## Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

Definition in file skua.h.

5.50.2 Macro Definition Documentation

```
5.50.2.1 #define D_c( Diff, phi ) ( Diff * (1.0/((1.0+1.1E-6)-phi) ) )
```

Approximate Darken Diffusivity Equation (um<sup>2</sup>/hr)

Definition at line 48 of file skua.h.

```
5.50.2.2 #define D_inf( Dref, Tref, B, p, T) ( Dref * pow(p+sqrt(DBL_EPSILON),(Tref/T)-B) )
```

Empirical correction of diffusivity (um<sup>2</sup>/hr)

Definition at line 40 of file skua.h.

```
5.50.2.3 #define D_o( Diff, E, T) (Diff * \exp(-E/(Rstd*T)))
```

Arrhenius Rate Expression for Diffusivity (um<sup>2</sup>/hr)

Definition at line 44 of file skua.h.

5.50.2.4 #define SKUA\_HPP\_

Definition at line 37 of file skua.h.

5.50.3 Function Documentation

5.50.3.1 double const\_Dc ( int i, int I, const void \* data )

Constant surface diffusivity function.

This function allows the user to specify just a single constant value for surface diffusivity. The value of diffusivity applied at all nodes will be the ref\_diffusion parameter in SKUA\_PARAM.

## **Parameters**

<ul> <li>i index of the gas/adsorbed phase species that this function acts on</li> <li>l index of the node in the spatial discretization that this function acts on</li> </ul>	

5.50.3.2 double const\_kf ( int i, const void \* data )

Constant function for film mass transfer coefficent.

This function allows the user to specify a constant value for the film mass transfer coefficient. The value of the film mass transfer coefficient will be the value of film\_transfer given in the SKUA\_PARAM data structure.

## **Parameters**

i	index of the gas/adsorbed phase species that this function acts on
data	pointer to the SKUA_DATA structure

5.50.3.3 double default\_Dc ( int i, int l, const void \* data )

Default function for surface diffusivity.

This is the default function provided by SKUA for the calculation of the surface diffusivity parameter. The diffusivity is calculated based on the Arrhenius rate expression, then corrected for using the empirical correction term with the outside partial pressure of the gas species.

#### **Parameters**

i	i index of the gas/adsorbed phase species that this function acts on	
<ul><li>I index of the node in the spatial discretization that this function acts on</li><li>data pointer to the SKUA_DATA structure</li></ul>		

### 5.50.3.4 double default\_kf ( int i, const void \* data )

Default function for film mass transfer coefficent.

This is the default function provided by SKUA for the calculation of the film mass transfer parameter. By default, we are usually going to couple the SKUA model with a pore diffusion model (see scopsowl.h). Therefore, the film mass transfer coefficient would be zero, because we would only consider a Dirichlet boundary condition for this sub-problem.

### **Parameters**

i	index of the gas/adsorbed phase species that this function acts on
data	pointer to the SKUA_DATA structure

## 5.50.3.5 double empirical\_kf ( int i, const void \* data )

Empirical function for film mass transfer coefficent.

This function provides an empirical estimate of the mass transfer coefficient using the gas velocity, molecular diffusivities, and dimensionless numbers (see egret.h). It is used as the default film mass transfer function IF the boundary condition is specified to be a Neumann type boundary by the user.

#### **Parameters**

i	index of the gas/adsorbed phase species that this function acts on
data	pointer to the SKUA_DATA structure

## 5.50.3.6 int molefractionCheck ( SKUA\_DATA \* skua\_dat )

Function to check mole fractions in gas and solid phases for errors.

This function is called after reading input and before calling the primary solution routines. It will force and error and quit the program if their are inconsistencies in the mole fractions it was given. All mole fractions must sum to 1, otherwise there is missing information.

```
5.50.3.7 void print2file_SKUA_header ( SKUA_DATA * skua_dat )
```

Function calls the other header functions to establish output file structure.

```
5.50.3.8 void print2file_SKUA_results_new ( SKUA_DATA * skua_dat )
```

Function to print out the new time step simulation results to the output file.

```
5.50.3.9 void print2file_SKUA_results_old ( SKUA_DATA * skua_dat )
```

Function to print out the old time step simulation results to the output file.

```
5.50.3.10 void print2file_SKUA_time_header ( FILE * Output, SKUA_DATA * skua_dat, int i )
```

Function to print out time and space headers to output file.

```
5.50.3.11 void print2file_species_header ( FILE * Output, SKUA_DATA * skua_dat, int i )
```

Function to print out the species' headers to output file.

5.50.3.12 int set\_SKUA\_ICs ( SKUA\_DATA \* skua\_dat )

Function to establish the initial conditions of adsorption in the adsorbent.

This function needs to be called before doing any simulation or execution of a time step, but only once per simulation. It sets the value of adsorption for each adsorbable species to the specified initial values given via qT and xIC in SKUA\_DATA.

5.50.3.13 int set\_SKUA\_params ( const void \* user\_data )

Function to call the diffusivity function during the solve.

This is the function passed into FINCH to be called during the FINCH solver (see finch.h). It will call the diffusion functions set by the user in the setup function above. This is not overridable.

5.50.3.14 int set\_SKUA\_timestep ( SKUA\_DATA \* skua\_dat )

Function to establish the time step for the current simulation.

This function is called to set a time step value for a particular simulation step. By default, the time step is set to (1/4)x space step size. If you need to change the step size, you must do so manually.

5.50.3.15 int setup\_SKUA\_DATA ( FILE \* file, double(\*)(int i, int I, const void \*user\_data) eval\_Dc, double(\*)(int i, const void \*user\_data) eval\_Kf, const void \* user\_data, MIXED\_GAS \* gas\_data, SKUA\_DATA \* skua\_dat )

Function to setup the function pointers and vector objects in memory to setup the SKUA simulation.

This function is called to setup the SKUA problem in memory and set function pointers to either defaults or user specified functions. It must be called prior to calling any other SKUA function and will report an error if the object was not setup properly.

### **Parameters**

file	pointer to the output file for SKUA simulations	
eval_Dc	pointer to the function to evaluate the surface diffusivity	
eval_Kf	Kf pointer to the function to evaluate the film mass transfer coefficient	
user_data         pointer to a user defined data structure used in the calculation the the parameters           gas_data         pointer to the MIXED_GAS data structure for egret.h calculations		
		skua_dat

5.50.3.16 double simple\_darken\_Dc ( int i, int I, const void \* data )

Simple Darken model for surface diffusivity.

This function uses an approximation to Darken's model for surface diffusion. The approximation is exact if the isotherm for adsorption takes the form of the Langmuir model, but is only approximate if the isotherm is heterogeneous. Forming the approximation in this manner is significantly cheaper than forming the true Darken model expression for the GSTA isotherm.

### **Parameters**

i	i index of the gas/adsorbed phase species that this function acts on	
I index of the node in the spatial discretization that this function acts on data pointer to the SKUA_DATA structure		

5.50.3.17 int SKUA ( SKUA\_DATA \* skua\_dat )

Function to iteratively call all execution steps to evolve a simulation through time.

This function is used in conjunction with the scenario call from the UI to numerically solve the adsorption kinetics problem in time. It will call the initial conditions function once, then iteratively call the reset, time step, and executioner functions for SKUA to push the simulation forward in time. This function will be called from the SKUA\_SCENARIOS

function.

```
5.50.3.18 int SKUA_Executioner ( SKUA_DATA * skua_dat )
```

Function to execute preprocesses, solvers, and postprocesses for a SKUA simulation.

This function calls the preprocess, solver, and postprocess functions to complete a single time step in a SKUA simulation. User's will want to call this function whenever a time step simulation result is needed. This is used primarily when coupling with other models (see scopsowl.h).

```
5.50.3.19 int SKUA_postprocesses ( SKUA_DATA * skua_dat )
```

Function to perform the necessary postprocess operations after a solve.

This function performs postprocess operations after a solve was completed successfully. Those operations include estimating average total adsorption, average adsorbed mole fractions, and heat of adsorption for each species. Results are then printed to the output file.

```
5.50.3.20 int SKUA_preprocesses ( SKUA_DATA * skua_dat )
```

Function to perform the necessary preprocess operations before a solve.

This function performs preprocess operations prior to calling the solver routine. Those preprocesses include establishing boundary conditions and performing a MAGPIE simulation for the adsorption on the surface (see magpie.h).

```
5.50.3.21 int SKUA_reset ( SKUA_DATA * skua_dat )
```

Function to reset the stateful information in SKUA after a simulation.

This function sets all the old state data to the newly formed state data. It needs to be called after a successful execution of the simulation step and before calling for the next time step to be solved. Do not call out of turn, otherwise information will be lost.

```
5.50.3.22 double theoretical_darken_Dc ( int i, int l, const void * data )
```

Theoretical Darken model for surface diffusivity.

This function uses the full theoretical expression of the Darken's diffusion model to calculate the surface diffusivity. This calculation involves formulating the reference state pressures for the adsorbed amount at every node, then calculating derivatives of the adsorption isotherm for each species. It is more accurate than the simple Darken model function, but costs significantly more computational time.

## Parameters

i index of the gas/adsorbed pha		index of the gas/adsorbed phase species that this function acts on
	1	index of the node in the spatial discretization that this function acts on
	data	pointer to the SKUA DATA structure

### 5.51 TotalColumnPressure.h File Reference

Auxillary kernel to calculate total column pressure based on temperature and concentrations.

```
#include "AuxKernel.h"
```

## Classes

· class TotalColumnPressure

Total Column Pressure class inherits from AuxKernel.

### **Functions**

template<>
 InputParameters validParams< TotalColumnPressure > ()

## 5.51.1 Detailed Description

Auxillary kernel to calculate total column pressure based on temperature and concentrations. This file is responsible for calculating the total column pressure in a fixed-bed adsorber given the temperature and the concentrations of all species in the gas phase. The gas phase is assumed to behave ideally and ideal gas law is employed to estimate the pressure.

### Author

Austin Ladshaw

#### Date

11/20/2015

## Copyright

This kernel was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science and was developed for use by Idaho National Laboratory and Oak Ridge National Laboratory engineers and scientists. Portions Copyright (c) 2015, all rights reserved.

Austin Ladshaw does not claim any owership or copyright to the MOOSE framework in which these kernels are constructed, only the kernels themselves. The MOOSE framework copyright is held by the Battelle Energy Alliance, LLC (c) 2010, all rights reserved.

Definition in file TotalColumnPressure.h.

```
5.51.2 Function Documentation
```

5.51.2.1 template<> InputParameters validParams< TotalColumnPressure > ( )

## 5.52 TotalPressurelC.h File Reference

#include "InitialCondition.h"

## Classes

class TotalPressureIC

## **Functions**

template<>
 InputParameters validParams< TotalPressureIC > ()

## 5.52.1 Function Documentation

5.52.1.1 template <> InputParameters validParams < TotalPressureIC > ( )

## 5.53 WallAmbientHeatTransfer.h File Reference

```
#include "Kernel.h"
```

## Classes

• class WallAmbientHeatTransfer

## **Functions**

template<>
 InputParameters validParams< WallAmbientHeatTransfer > ()

## 5.53.1 Function Documentation

```
5.53.1.1 \quad template <> Input Parameters \ valid Params < Wall Ambient Heat Transfer > ( \quad )
```

## 5.54 WallHeatAccumulation.h File Reference

```
#include "TimeDerivative.h"
```

## Classes

· class WallHeatAccumulation

## **Functions**

template<>
 InputParameters validParams< WallHeatAccumulation > ()

## 5.54.1 Function Documentation

 $5.54.1.1 \quad template <> Input Parameters \ valid Params < \ Wall Heat Accumulation > ( \ \ )$ 

# Index

$\sim$ DgospreyApp	DGFluxBC, 54
DgospreyApp, 73	DGFluxLimitedBC, 57
~Matrix	DGHeatFluxBC, 60
Matrix, 138	DGHeatFluxLimitedBC, 63
	DGMassFluxBC, 67
_Diffusion	DGMassFluxLimitedBC, 71
DGAnisotropicDiffusion, 33	
DGColumnHeatDispersion, 39	GAnisotropicDiffusion, 97
DGColumnMassDispersion, 43	GColumnHeatDispersion, 100
DGColumnWallHeatFluxBC, 47	GColumnMassDispersion, 103
DGColumnWallHeatFluxLimitedBC, 50	_Dyx
DGFluxBC, 54	DGAnisotropicDiffusion, 34
DGFluxLimitedBC, 56	DGColumnHeatDispersion, 39
DGHeatFluxBC, 59	DGColumnMassDispersion, 44
DGHeatFluxLimitedBC, 63	DGColumnWallHeatFluxBC, 47
DGMassFluxBC, 67	DGColumnWallHeatFluxLimitedBC, 50
DGMassFluxLimitedBC, 71	DGFluxBC, 54
GAnisotropicDiffusion, 96	DGFluxLimitedBC, 57
GColumnHeatDispersion, 100	DGHeatFluxBC, 60
GColumnMassDispersion, 103	DGHeatFluxLimitedBC, 63
_Dxx	DGMassFluxBC, 67
DGAnisotropicDiffusion, 33	DGMassFluxLimitedBC, 71
DGColumnHeatDispersion, 39	GAnisotropicDiffusion, 97
DGColumnMassDispersion, 44	GColumnHeatDispersion, 100
DGColumnWallHeatFluxBC, 47	GColumnMassDispersion, 104
DGColumnWallHeatFluxLimitedBC, 50	_Dyy
DGFluxBC, 54	DGAnisotropicDiffusion, 34
DGFluxLimitedBC, 56	DGColumnHeatDispersion, 39
DGHeatFluxBC, 60	DGColumnMassDispersion, 44
DGHeatFluxLimitedBC, 63	DGColumnWallHeatFluxBC, 47
DGMassFluxBC, 67	DGColumnWallHeatFluxLimitedBC, 50
DGMassFluxLimitedBC, 71	DGFluxBC, 54
GAnisotropicDiffusion, 96	DGFluxLimitedBC, 57
GColumnHeatDispersion, 100	DGHeatFluxBC, 60
GColumnMassDispersion, 103	DGHeatFluxLimitedBC, 63
_Dxy	DGMassFluxBC, 67
DGAnisotropicDiffusion, 34	DGMassFluxLimitedBC, 71
DGColumnHeatDispersion, 39	GAnisotropicDiffusion, 97
DGColumnMassDispersion, 44	GColumnHeatDispersion, 100
DGColumnWallHeatFluxBC, 47	GColumnMassDispersion, 104
DGColumnWallHeatFluxLimitedBC, 50	_Dyz
DGFluxBC, 54	DGAnisotropicDiffusion, 34
DGFluxLimitedBC, 56	DGColumnHeatDispersion, 39
DGHeatFluxBC, 60	DGColumnMassDispersion, 44
DGHeatFluxLimitedBC, 63	DGColumnWallHeatFluxBC, 47
DGMassFluxBC, 67	DGColumnWallHeatFluxLimitedBC, 50
DGMassFluxLimitedBC, 71	DGFluxBC, 54
GAnisotropicDiffusion, 97	DGFluxLimitedBC, 57
GColumnHeatDispersion, 100	DGHeatFluxBC, 60
GColumnMassDispersion, 103	DGHeatFluxLimitedBC, 63
_Dxz	DGMassFluxBC, 67
DGAnisotropicDiffusion, 34	DGMassFluxLimitedBC, 71
DGColumnHeatDispersion, 39	GAnisotropicDiffusion, 97
•	•
DGColumnWallHeatFluxPC 47	GColumnHeatDispersion, 100
DGColumnWallHeatFluxI imitedPC 50	GColumnMassDispersion, 104
DGColumnWallHeatFluxLimitedBC, 50	_Dzx

DGAnisotropicDiffusion, 34	WallAmbientHeatTransfer, 183
DGColumnHeatDispersion, 39	_bed_wall_transfer_coeff
DGColumnMassDispersion, 44	BedProperties, 18
DGColumnWallHeatFluxBC, 47	BedWallHeatTransfer, 20
DGColumnWallHeatFluxLimitedBC, 50	DGColumnWallHeatFluxBC, 46
DGFluxBC, 54	DGColumnWallHeatFluxLimitedBC, 50
DGFluxLimitedBC, 57	FluidHeatTransfer, 94
DGHeatFluxBC, 60	_binder_fraction
DGHeatFluxLimitedBC, 63	AdsorbentProperties, 10
DGMassFluxBC, 67	_binder_porosity
DGMassFluxLimitedBC, 71	AdsorbentProperties, 10
GAnisotropicDiffusion, 97	_coef
GColumnHeatDispersion, 100	LinearDrivingForce, 124
GColumnMassDispersion, 104	_column_length
Dzy	FlowProperties, 91
DGAnisotropicDiffusion, 34	_column_temp
DGColumnHeatDispersion, 39	BedWallHeatTransfer, 20
DGColumnMassDispersion, 44	_comp_Sutherland_const
DGColumnWallHeatFluxBC, 47	FlowProperties, 91
DGColumnWallHeatFluxLimitedBC, 50	_comp_heat_capacity
DGFluxBC, 54	FlowProperties, 91
DGFluxLimitedBC, 57	_comp_ref_temp
DGHeatFluxBC, 60	FlowProperties, 91
DGHeatFluxLimitedBC, 63	_comp_ref_viscosity
DGMassFluxBC, 67	FlowProperties, 91
DGMassFluxLimitedBC, 71	_conductivity
GAnisotropicDiffusion, 97	BedProperties, 18
GColumnHeatDispersion, 100	DGColumnHeatDispersion, 39
GColumnMassDispersion, 104	DGColumnWallHeatFluxBC, 46
_Dzz	DGColumnWallHeatFluxLimitedBC, 50
DGAnisotropicDiffusion, 34	DGHeatFluxBC, 59
DGColumnHeatDispersion, 39	DGHeatFluxLimitedBC, 63
DGColumnMassDispersion, 44	GColumnHeatDispersion, 100
DGColumnWallHeatFluxBC, 47	HeatConductivity, 120
DGColumnWallHeatFluxLimitedBC, 50	crystal radius
DGFluxBC, 54	AdsorbentProperties, 10
DGFluxLimitedBC, 57	_din
DGHeatFluxBC, 60	BedProperties, 18
DGHeatFluxLimitedBC, 63	_dispersion
DGMassFluxBC, 67	DGColumnMassDispersion, 44
DGMassFluxLimitedBC, 71	DGMassFluxBC, 67
GAnisotropicDiffusion, 97	DGMassFluxLimitedBC, 71
GColumnHeatDispersion, 100	Dispersion, 75
GColumnMassDispersion, 104	FlowProperties, 91
_Kz	GColumnMassDispersion, 103
BedProperties, 18	_dout
_PT_IC	BedProperties, 18
DGConcentrationIC, 52	_driving_value
TotalPressureIC, 182	LinearDrivingForce, 124
_TC_IC	_eb
ColumnTemperatureIC, 29	BedProperties, 18
_T_IC	_enthalpy_1
DGConcentrationIC, 52	MagpieAdsorbateProperties, 133
_Ua	_enthalpy_2
BedProperties, 19	MagpieAdsorbateProperties, 133
_Uw	
ow BedProperties, 19	_enthalpy_3 MagpieAdsorbateProperties, 133
_ambient_temp	_enthalpy_4
_amorem_temp	_======================================

MagpieAdsorbateProperties, 133	_hw
_enthalpy_5	BedProperties, 18
MagpieAdsorbateProperties, 133	_index
_enthalpy_6	AdsorbentProperties, 11
MagpieAdsorbateProperties, 133	BedProperties, 18
entropy 1	DGColumnMassDispersion, 44
MagpieAdsorbateProperties, 133	DGMassFluxBC, 67
entropy 2	DGMassFluxLimitedBC, 71
,_	Dispersion, 75
MagpieAdsorbateProperties, 133	•
_entropy_3	FlowProperties, 92
MagpieAdsorbateProperties, 133	GColumnMassDispersion, 104
_entropy_4	MAGPIE_Adsorption, 126
MagpieAdsorbateProperties, 133	MAGPIE_AdsorptionHeat, 127
_entropy_5	MAGPIE_Perturbation, 130
MagpieAdsorbateProperties, 133	MagpieAdsorbateProperties, 134
_entropy_6	RetardedTimeDerivative, 163
MagpieAdsorbateProperties, 133	TotalColumnPressure, 181
_epsilon	_inner_dia
DGAnisotropicDiffusion, 34	BedProperties, 18
DGColumnHeatDispersion, 39	BedWallHeatTransfer, 20
DGColumnMassDispersion, 44	FlowProperties, 92
DGColumnWallHeatFluxLimitedBC, 50	FluidHeatTransfer, 94
DGFluxLimitedBC, 57	WallAmbientHeatTransfer, 183
DGHeatFluxLimitedBC, 64	_input_molefraction
DGMassFluxLimitedBC, 71	DGMassFluxBC, 67
_flow_rate	DGMassFluxLimitedBC, 71
FlowProperties, 91	_input_pressure
_gaining	DGMassFluxBC, 67
LinearDrivingForce, 124	DGMassFluxLimitedBC, 72
_gas_conc	_input_temperature
AdsorbentProperties, 11	DGHeatFluxBC, 60
BedProperties, 18	DGHeatFluxLimitedBC, 64
FlowProperties, 91	DGMassFluxBC, 68
MagpieAdsorbateProperties, 134	DGMassFluxLimitedBC, 72
TotalColumnPressure, 181	_length
gas const	BedProperties, 19
FlowProperties.h, 219	_macropore_radius
_gas_density	AdsorbentProperties, 11
DGColumnHeatAdvection, 36	_magpie_dat
DGHeatFluxBC, 60	MAGPIE_Adsorption, 126
DGHeatFluxLimitedBC, 64	MAGPIE_AdsorptionHeat, 127
FlowProperties, 91	MAGPIE_Perturbation, 130
GColumnHeatAdvection, 98	MagpieAdsorbateProperties, 134
_gas_heat_capacity	_max_capacity
DGColumnHeatAdvection, 36	MagpieAdsorbateProperties, 134
DGHeatFluxBC, 60	_molar_volume
DGHeatFluxLimitedBC, 64	MagpieAdsorbateProperties, 134
FlowProperties, 91	_molecular_diffusion
GColumnHeatAdvection, 98	DGColumnMassDispersion, 44
_gas_molecular_wieght	DGMassFluxBC, 68
FlowProperties, 91	DGMassFluxLimitedBC, 72
_gas_viscosity	FlowProperties, 92
FlowProperties, 91	GColumnMassDispersion, 104
_heat_retardation	_molecular_wieght
BedHeatAccumulation, 16	FlowProperties, 92
FlowProperties, 92	_num_sites
hs	MagpieAdsorbateProperties, 134
AdsorbentProperties, 11	
Ausonachii Toperiles, TT	_outer_dia

BedProperties, 19	DGHeatFluxBC, 60
BedWallHeatTransfer, 21	DGHeatFluxLimitedBC, 64
WallAmbientHeatTransfer, 183	DGMassFluxBC, 68
_pellet_density	DGMassFluxLimitedBC, 72
AdsorbentProperties, 11	_var
FlowProperties, 92	LinearDrivingForce, 124
MAGPIE_HeatAccumulation, 129	_vel
_pellet_diameter	DGColumnHeatAdvection, 36
AdsorbentProperties, 11	DGColumnMassAdvection, 41
_pellet_heat_capacity	DGHeatFluxBC, 61
AdsorbentProperties, 11	DGHeatFluxLimitedBC, 64
FlowProperties, 92	DGMassFluxBC, 68
_porosity	DGMassFluxLimitedBC, 72
BedProperties, 19	GColumnHeatAdvection, 98
FlowProperties, 92	GColumnMassAdvection, 102
FluidHeatTransfer, 94	_velocity
HeatConductivity, 120	DGAdvection, 31
MAGPIE_HeatAccumulation, 129	DGColumnHeatAdvection, 36
retardation	DGColumnMassAdvection, 41
FlowProperties, 92	DGColumnWallHeatFluxBC, 47
RetardedTimeDerivative, 163	DGColumnWallHeatFluxLimitedBC, 51
_rhos	DGFluxI inited BC F7
AdsorbentProperties, 11	DGFluxLimitedBC, 57
_rhow	DGHeatFluxBC, 61
BedProperties, 19	DGHeatFluxLimitedBC, 64
_sigma	DGMassFluxBC, 68
DGAnisotropicDiffusion, 34	DGMassFluxLimitedBC, 72
DGColumnHeatDispersion, 39	FlowProperties, 93
DGColumnMassDispersion, 44	GAdvection, 95
DGColumnWallHeatFluxLimitedBC, 51	GColumnHeatAdvection, 98
DGFluxLimitedBC, 57	GColumnMassAdvection, 102
DGHeatFluxLimitedBC, 64	_vx
DGMassFluxLimitedBC, 72	DGAdvection, 31
_solid_conc	DGColumnHeatAdvection, 36
FlowProperties, 92	DGColumnMassAdvection, 41
MAGPIE_AdsorptionHeat, 127	DGColumnWallHeatFluxBC, 47
_solid_heat	DGColumnWallHeatFluxLimitedBC, 51
MAGPIE_HeatAccumulation, 129	DGFluxBC, 54
_solid_heat_old	DGFluxLimitedBC, 57
MAGPIE HeatAccumulation, 129	DGHeatFluxBC, 61
_solid_perturb	DGHeatFluxLimitedBC, 64
FlowProperties, 92	DGMassFluxBC, 68
temperature	DGMassFluxLimitedBC, 72
AdsorbentProperties, 11	GAdvection, 95
BedProperties, 19	GColumnHeatAdvection, 98
ConcentrationIC, 30	GColumnMassAdvection, 102
FlowProperties, 93	•
MagpieAdsorbateProperties, 134	_vy DGAdvection, 31
TotalColumnPressure, 181	DGColumnHeatAdvection, 37
_total_pressure	DGColumnMassAdvection, 41
ConcentrationIC, 30	DGColumnWallHeatFluxBC, 47
FlowProperties, 93	DGColumnWallHeatFluxLimitedBC, 51
MagpieAdsorbateProperties, 134	DGFluxBC, 55
_u_input	DGFluxLimitedBC, 57
DGColumnWallHeatFluxBC, 47	DGHeatFluxBC, 61
DGColumnWallHeatFluxLimitedBC, 51	DGHeatFluxLimitedBC, 64
DGFluxBC, 54	DGMassFluxBC, 68
DGFluxLimitedBC, 57	DGMassFluxLimitedBC, 72

GAdvection, 95	_binder_fraction, 10
GColumnHeatAdvection, 98	binder porosity, 10
GColumnMassAdvection, 102	_crystal_radius, 10
_VZ	_gas_conc, 11
DGAdvection, 32	_gas_cone, 11 _hs, 11
DGColumnHeatAdvection, 37	_index, 11
DGColumnMassAdvection, 41	_macropore_radius, 11
DGColumnWallHeatFluxBC, 48	_nacropore_radius, 11
DGColumnWallHeatFluxLimitedBC, 51	_pellet_diameter, 11
DGGoldminwaiineati idxLiinitedBC, 31	
	_pellet_heat_capacity, 11 rhos, 11
DGFluxLimitedBC, 57	<i>_ ,</i>
DGHeatFluxBC, 61	_temperature, 11
DGHeatFluxLimitedBC, 64	AdsorbentProperties, 10
DGMassFluxBC, 68	AdsorbentProperties, 10
DGMassFluxLimitedBC, 72	computeQpProperties, 10
GAdvection, 95	AdsorbentProperties.h, 185
GColumnHeatAdvection, 99	validParams< AdsorbentProperties >, 186
GColumnMassAdvection, 102	affinity
_wall_density	SCOPSOWL_PARAM_DATA, 170
BedProperties, 19	SKUA_PARAM, 177
WallHeatAccumulation, 184	Ai
_wall_exterior_transfer_coeff	OPTRANS_DATA, 150
BedProperties, 19	alpha
WallAmbientHeatTransfer, 183	BACKTRACK_DATA, 14
_wall_heat_capacity	BiCGSTAB_DATA, 22
BedProperties, 19	CGS_DATA, 26
WallHeatAccumulation, 184	GCR_DATA, 105
_wall_temp	PCG_DATA, 152
DGColumnWallHeatFluxBC, 48	anchor_alias_dne
DGColumnWallHeatFluxLimitedBC, 51	error.h, 210
FluidHeatTransfer, 94	Ap
_y_IC	PCG_DATA, 152
ConcentrationIC, 30	arg
DGConcentrationIC, 52	GMRESR_DATA, 111
	arg_matrix_same
A	error.h, 210
magpie.h, 240	arnoldi
ARNOLDI_DATA, 12	lark.h, 226
beta, 12	arnoldi dat
e1, 12	GMRESLP DATA, 109
Hkp1, 13	As
hp1, 13	SYSTEM DATA, 179
iter, 13	associateSyntax
k, 13	DgospreyApp, 73
Output, 13	avg_norm
sum, 13	SYSTEM DATA, 179
v, 13	
Vk, 13	avgDp
w, 13	scopsowl.h, 252
yk, 13	BACKTRACK DATA, 14
activation_energy	alpha, 14
SCOPSOWL_PARAM_DATA, 170	a.pa,
SKUA PARAM, 177	constRho, 14
adjoint	constRho, 14 Fk 14
adjoint	Fk, 14
Matrix 138	Fk, 14 fun_call, 14
Matrix, 138	Fk, 14 fun_call, 14 lambdaMin, 15
Adsorbable	Fk, 14 fun_call, 14 lambdaMin, 15 normFkp1, 15
Adsorbable SCOPSOWL_PARAM_DATA, 170	Fk, 14 fun_call, 14 lambdaMin, 15 normFkp1, 15 rho, 15
Adsorbable	Fk, 14 fun_call, 14 lambdaMin, 15 normFkp1, 15

PJFNK DATA, 157	bestx
backtrackLineSearch	BiCGSTAB DATA, 22
lark.h, 226	CGS_DATA, 26
base/DgospreyRevision.h	GCR_DATA, 106
DGOSPREY REVISION, 203	GMRESLP DATA, 109
BedHeatAccumulation, 15	GMRESRP_DATA, 114
_heat_retardation, 16	PCG_DATA, 152
BedHeatAccumulation, 16	PICARD_DATA, 154
BedHeatAccumulation, 16	PJFNK DATA, 157
computeQpJacobian, 16	beta
computeQpResidual, 16	ARNOLDI DATA, 12
BedHeatAccumulation.h, 186	BiCGSTAB_DATA, 22
validParams< BedHeatAccumulation >, 187	CGS_DATA, 26
BedProperties, 16	FINCH_DATA, 79
_Kz, 18	GCR_DATA, 106
Ua, 19	PCG_DATA, 152
_Uw, 19	BICGSTAB
_bed_wall_transfer_coeff, 18	lark.h, 226
_conductivity, 18	BiCGSTAB_DATA, 21
_din, 18	alpha, <mark>22</mark>
_dout, 18	bestres, 22
_eb, 18	bestx, 22
_gas_conc, 18	beta, 22
_hw, 18	breakdown, 22
_index, 18	iter, 23
_inner_dia, 18	maxit, 23
_length, 19	omega, 23
_outer_dia, 19	omega_old, 23
_porosity, 19	Output, 23
_rhow, 19	p, 23
_temperature, 19	r, 23
_wall_density, 19	r0, <del>23</del>
_wall_exterior_transfer_coeff, 19	relres, 23
_wall_heat_capacity, 19	relres_base, 23
BedProperties, 17	res, 23
BedProperties, 17	rho, <mark>24</mark>
computeQpProperties, 18	rho_old, 24
BedProperties.h, 187	s, 24
validParams< BedProperties >, 187	t, 24
BedWallHeatTransfer, 20	tol_abs, 24
_bed_wall_transfer_coeff, 20	tol_rel, 24
_column_temp, 20	v, 24
_inner_dia, 20	x, 24
_outer_dia, 21	y, 24
BedWallHeatTransfer, 20	z, 24
BedWallHeatTransfer, 20	bicgstab
computeQpJacobian, 20	lark.h, 227
computeQpResidual, 20	bicgstab_dat
BedWallHeatTransfer.h, 187	PJFNK_DATA, 157
validParams< BedWallHeatTransfer >, 187	binary_diffusion
bestres	MIXED_GAS, 147
BICGSTAB_DATA, 22	binder_fraction
CGS_DATA, 26	SCOPSOWL_DATA, 165
GCR_DATA, 105	binder_poresize
GMRESLP_DATA, 109	SCOPSOWL_DATA, 165
GMRESRP_DATA, 114	binder_porosity
PCG_DATA, 152	SCOPSOWL_DATA, 165
PICARD DATA, 154	Bounce

PJFNK_DATA, 158	SYSTEM_DATA, 179
breakdown	Cartesian
BiCGSTAB_DATA, 22	finch.h, 213
CGS_DATA, 26	cgs
GCR_DATA, 106	lark.h, 228
0	cgs_dat
c CGS_DATA, 26	PJFNK_DATA, 158
GCR_DATA, 20	char_length
CGS	MIXED_GAS, 147
lark.h, 226	char_macro
c temp	SCOPSOWL_DATA, 165
GCR_DATA, 106	char_measure
CC_E	SKUA_DATA, 174 char_micro
FINCH DATA, 79	
CC_I	SCOPSOWL_DATA, 166 check_Mass
FINCH DATA, 79	finch.h, 214
CE3	CheckMass
egret.h, 206	FINCH DATA, 80
CGS_DATA, 25	CheckMolefractions
alpha, 26	MIXED_GAS, 147
bestres, 26	cofactor
bestx, 26	Matrix, 138
beta, 26	columnExtend
breakdown, 26	Matrix, 138
c, 26	columnExtract
iter, 26	Matrix, 138
maxit, 26	columnProjection
Output, 27	Matrix, 138
p, 27	columnReplace
r, 27	Matrix, 138
r0, 27	columnShrink
relres, 27	Matrix, 139
relres_base, 27	ColumnTemperatureIC, 28
res, 27	_TC_IC, 29
rho, 27	ColumnTemperatureIC, 29
sigma, 27	ColumnTemperatureIC, 29
tol_abs, 27	value, 29
tol_rel, 27	ColumnTemperatureIC.h, 187
u, 28	validParams< ColumnTemperatureIC >, 188
v, 28 w, 28	columnVectorFill
w, 26 x, 28	Matrix, 139
z, 28	columns
CL_E	Matrix, 139
FINCH_DATA, 80	computeQpJacobian
CL_I	BedHeatAccumulation, 16
FINCH_DATA, 80	BedWallHeatTransfer, 20
CN	DGAdvection, 31
FINCH_DATA, 80	DGAnisotropicDiffusion, 33 DGColumnHeatAdvection, 36
CR E	DGColumnHeatAdvection, 38
FINCH_DATA, 80	DGColumnMassAdvection, 41
CR_I	DGColumnMassDispersion, 43
FINCH_DATA, 80	DGColumnWallHeatFluxBC, 46
calculate_properties	DGColumnWallHeatFluxLimitedBC, 49
egret.h, 207	DGFluxBC, 53
callroutine	DGFluxLimitedBC, 56
FINCH_DATA, 79	DGHeatFluxBC, 59
Carrier	

DGHeatFluxLimitedBC, 63	ConcentrationIC, 29
DGMassFluxBC, 66	_temperature, 30
DGMassFluxLimitedBC, 70	_total_pressure, 30
Dispersion, 74	_y_IC, 30
FluidHeatTransfer, 94	ConcentrationIC, 29
GAdvection, 95	ConcentrationIC, 29
GAnisotropicDiffusion, 96	value, 29
GColumnHeatAdvection, 98	ConcentrationIC.h, 188
GColumnHeatDispersion, 100	validParams < ConcentrationIC >, 188
GColumnMassAdvection, 101	const Dc
GColumnMassDispersion, 103	skua.h, 259
HeatConductivity, 120	const_filmMassTransfer
LinearDrivingForce, 124	scopsowl.h, 252
MAGPIE_HeatAccumulation, 129	const_kf
RetardedTimeDerivative, 163	skua.h, 259
WallAmbientHeatTransfer, 183	
	const_pore_diffusion
WallHeatAccumulation, 184	scopsowl.h, 253
computeQpProperties	constRho
AdsorbentProperties, 10	BACKTRACK_DATA, 14
BedProperties, 18	ConstantICFill
FlowProperties, 90	Matrix, 139
MagpieAdsorbateProperties, 132	coord
computeQpResidual	SKUA_DATA, 174
BedHeatAccumulation, 16	coord_macro
BedWallHeatTransfer, 20	SCOPSOWL_DATA, 166
DGAdvection, 31	coord_micro
DGAnisotropicDiffusion, 33	SCOPSOWL_DATA, 166
DGColumnHeatAdvection, 36	crystal_radius
DGColumnHeatDispersion, 38	SCOPSOWL_DATA, 166
DGColumnMassAdvection, 41	Cstd
DGColumnMassDispersion, 43	egret.h, 206
DGColumnWallHeatFluxBC, 46	Cylindrical
DGColumnWallHeatFluxLimitedBC, 49	finch.h, 213
DGFluxBC, 53	
DGFluxLimitedBC, 56	d
DGHeatFluxBC, 59	FINCH_DATA, 80
DGHeatFluxLimitedBC, 63	D_c
DGMassFluxBC, 66	skua.h, <mark>259</mark>
DGMassFluxLimitedBC, 70	D_ii
Dispersion, 74	egret.h, 206
FluidHeatTransfer, 94	D_ij
GAdvection, 95	egret.h, 206
GAnisotropicDiffusion, 96	D inf
GColumnHeatAdvection, 98	 skua.h, 259
•	D o
GColumnHeatDispersion, 100	skua.h, 259
GColumnMassAdvection, 102	DBL EPSILON
GColumnMassDispersion, 103	magpie.h, 240
HeatConductivity, 120	DGAdvection, 30
LinearDrivingForce, 124	_velocity, 31
MAGPIE_HeatAccumulation, 129	_ •
RetardedTimeDerivative, 163	_vx, 31
WallAmbientHeatTransfer, 183	_vy, 31
WallHeatAccumulation, 184	_VZ, 32
computeValue	computeQpJacobian, 31
MAGPIE_Adsorption, 125	computeQpResidual, 31
MAGPIE_AdsorptionHeat, 127	DGAdvection, 31
MAGPIE_Perturbation, 130	DGAdvection, 31
TotalColumnPressure, 181	DGAdvection.h, 188
	validParams< DGAdvection >, 189

DGAnisotropicDiffusion, 32	_vz, 41
_Diffusion, 33	computeQpJacobian, 41
_Dxx, 33	computeQpResidual, 41
_Dxy, <mark>34</mark>	DGColumnMassAdvection, 41
_Dxz, 34	DGColumnMassAdvection, 41
	DGColumnMassAdvection.h, 193
	validParams< DGColumnMassAdvection >, 193
_Dyz, 34	DGColumnMassDispersion, 42
_Dzx, 34	_Diffusion, 43
Dzy, 34	
_Dzz, 34	Dxy, 44
_epsilon, 34	_Dxz, 44
_sigma, 34	
computeQpJacobian, 33	
computeQpResidual, 33	_Dyz, 44
DGAnisotropicDiffusion, 33	
DGAnisotropicDiffusion, 33	Dzy, 44
DGAnisotropicDiffusion.h, 189	_Dzz, 44
validParams< DGAnisotropicDiffusion >, 190	dispersion, 44
DGColumnHeatAdvection, 34	_epsilon, 44
_gas_density, 36	_index, 44
_gas_heat_capacity, 36	_molecular_diffusion, 44
_vel, 36	_sigma, 44
_velocity, 36	computeQpJacobian, 43
_vx, 36	computeQpResidual, 43
_vy, 37	DGColumnMassDispersion, 43
_vz, 37	DGColumnMassDispersion, 43
computeQpJacobian, 36	DGColumnMassDispersion.h, 194
computeQpResidual, 36	validParams < DGColumnMassDispersion >, 195
DGColumnHeatAdvection, 36	DGColumnWallHeatFluxBC, 45
DGColumnHeatAdvection, 36	_Diffusion, 47
DGColumnHeatAdvection.h, 190	
validParams< DGColumnHeatAdvection >, 191	
DGColumnHeatDispersion, 37	_Dxz, 47
_Diffusion, 39	
	Dyy, 47
	_Dyz, 47
_Dxz, 39	_Dzx, 47
_Dyy, 39	Dzz, 47
_Dyz, 39	_bed_wall_transfer_coeff, 46
_Dzx, 39	conductivity, 46
	_u_input, 47
_Dzz, 39	_velocity, 47
_conductivity, 39	_vx, 47
_epsilon, 39	_vy, 47
_sigma, 39	_vz, 48
computeQpJacobian, 38	_wall_temp, 48
computeQpResidual, 38	computeQpJacobian, 46
DGColumnHeatDispersion, 38	computeQpResidual, 46
DGColumnHeatDispersion, 38	DGColumnWallHeatFluxBC, 46
DGColumnHeatDispersion.h, 191	DGColumnWallHeatFluxBC, 46
validParams< DGColumnHeatDispersion >, 192	DGColumnWallHeatFluxBC.h, 195
DGColumnMassAdvection, 40	validParams< DGColumnWallHeatFluxBC >, 195
_vel, 41	DGColumnWallHeatFluxLimitedBC, 48
_velocity, 41	_Diffusion, 50
_vx, 41	_Dxx, 50
_vy, 41	_Dxy, 50
- · ·	_ //

_Dxz, 50	_Dxy, <del>56</del>
_Dyx, 50	_Dxz, <del>5</del> 7
_Dyy, <del>50</del>	_Dyx, <del>57</del>
_Dyz, 50	_Dyy, <del>57</del>
_Dzx, 50	_Dyz, <del>57</del>
_Dzy, 50	_Dzx, 57
_Dzz, 50	_Dzy, <del>57</del>
_bed_wall_transfer_coeff, 50	_Dzz, <del>57</del>
_conductivity, 50	_epsilon, 57
_epsilon, 50	_sigma, 57
sigma, 51	_u_input, 57
_u_input, 51	_velocity, 57
_velocity, 51	_vx, 57
_vx, 51	_vy, 57
_vy, 51	_vz, 57
_vz, 51	computeQpJacobian, 56
_vall_temp, 51	computeQpResidual, 56
computeQpJacobian, 49	DGFluxLimitedBC, 56
computeQpdacobian, 49	
	DGFluxLimitedBC, 56
DGColumnWallHeatFluxLimitedBC, 49	DGFluxLimitedBC.h, 198
DGColumnWallHeatFluxLimitedBC, 49	validParams< DGFluxLimitedBC >, 199
DGColumnWallHeatFluxLimitedBC.h, 196	DGHeatFluxBC, 58
validParams< DGColumnWallHeatFluxLimitedBC	_Diffusion, 59
>, 196	_Dxx, 60
DGConcentrationIC, 51	_Dxy, 60
_PT_IC, 52	_Dxz, 60
_T_IC, 52	_Dyx, 60
_y_IC, 52	_Dyy, <mark>60</mark>
DGConcentrationIC, 52	_Dyz, 60
DGConcentrationIC, 52	_Dzx, 60
value, 52	_Dzy, 60
DGConcentrationIC.h, 196	_Dzz, 60
validParams< DGConcentrationIC >, 197	_conductivity, 59
DGFluxBC, 52	_gas_density, 60
_Diffusion, 54	_gas_heat_capacity, 60
_Dxx, 54	_input_temperature, 60
	_u_input, 60
_Dxz, 54	_vel, 61
	_velocity, 61
_Dyy, 54	_vx, 61
_Dyz, 54	_vx, 61 _vy, 61
_Dzx, 54	_vy, 61 _vz, 61
_Dzy, 54	computeQpJacobian, 59
_Dzz, 54	
	computeQpResidual, 59
_u_input, 54	DGHeatFluxBC, 59
_velocity, 54	DGHeatFluxBC, 59
_vx, 54	DGHeatFluxBC.h, 199
_vy, 55	validParams< DGHeatFluxBC >, 200
_vz, 55	DGHeatFluxLimitedBC, 61
computeQpJacobian, 53	_Diffusion, 63
computeQpResidual, 53	_Dxx, 63
DGFluxBC, 53	_Dxy, <mark>63</mark>
DGFluxBC, 53	_Dxz, 63
DGFluxBC.h, 197	_Dyx, 63
validParams< DGFluxBC >, 198	_Dyy, 63
DGFluxLimitedBC, 55	_Dyz, 63
_Diffusion, 56	_Dzx, 63
_Dxx, 56	_Dzy, 63

B	D =1
_Dzz, 63	_Dzz, 71
_conductivity, 63	_dispersion, 71
_epsilon, 64	_epsilon, 71
_gas_density, 64	_index, 71
_gas_heat_capacity, 64	_input_processes_72
_input_temperature, 64	_input_temperature_72
_sigma, 64	_input_temperature, 72
_u_input, 64 _vel, 64	_molecular_diffusion, 72 _sigma, 72
_velocity, 64	_u_input, 72
_velocity, 64 vx, 64	_u_iiput, 72 _vel, 72
_vy, 64	velocity, 72
_vz, 64	_vx, 72
computeQpJacobian, 63	_vx, 72 _vy, 72
computeQpResidual, 63	_vz, 72
DGHeatFluxLimitedBC, 62	computeQpJacobian, 70
DGHeatFluxLimitedBC, 62	computeQpResidual, 70
DGHeatFluxLimitedBC.h, 200	DGMassFluxLimitedBC, 70
validParams< DGHeatFluxLimitedBC >, 200	DGMassFluxLimitedBC, 70
DGMassFluxBC, 65	DGMassFluxLimitedBC.h, 201
Diffusion, 67	validParams< DGMassFluxLimitedBC >, 202
	DGOSPREY_REVISION
_Dxy, 67	base/DgospreyRevision.h, 203
_Dxz, 67	DgospreyRevision.h, 203
_Dyx, 67	dHo
_Dyy, 67	GSTA_DATA, 119
_Dyz, 67	DIC
_Dzx, 67	FINCH_DATA, 80
_Dzy, 67	dSo
_Dzz, 67	GSTA_DATA, 119
_dispersion, 67	Data
_index, 67	Matrix, 145
_input_molefraction, 67	default_Dc
_input_pressure, 67	skua.h, 259
_input_temperature, 68	default_adsorption
_molecular_diffusion, 68	scopsowl.h, 253
_u_input, 68	default_bcs
_vel, 68	finch.h, 214
_velocity, 68	default_effective_diffusion
_vx, 68	scopsowl.h, 253
_vy, 68	default_execution finch.h, 214
_vz, 68 computeQpJacobian, 66	default_filmMassTransfer
computeQpSacobian, 66	scopsowl.h, 253
DGMassFluxBC, 66	default ic
DGMassFluxBC, 66	finch.h, 214
DGMassFluxBC.h, 200	default_kf
validParams< DGMassFluxBC >, 201	skua.h, 260
DGMassFluxLimitedBC, 68	default_params
_Diffusion, 71	finch.h, 214
_Dxx, 71	default_pore_diffusion
_Dxy, 71	scopsowl.h, 253
_Dxz, 71	default_postprocess
	finch.h, 214
_ Dyy, 71	default_precon
_Dyz, 71	finch.h, 214
_Dzx, 71	default_preprocess
_Dzy, 71	finch.h, 214

default_res	Dp
finch.h, 215	scopsowl.h, 252
default_reset	Dp_ij
finch.h, 215	egret.h, 206
default_retardation	dq_dc
scopsowl.h, 254	SCOPSOWL_PARAM_DATA, 170
default_solve	dq_dco
finch.h, 215	SCOPSOWL_PARAM_DATA, 170
default_surf_diffusion	dq_dp
scopsowl.h, 254	magpie.h, 241
default_timestep	dt
finch.h, 215	FINCH_DATA, 81
density	dt old
PURE_GAS, 161	FINCH_DATA, 81
determinate	duplicate_variable
Matrix, 139	error.h, 210
DgospreyApp, 73	dxj
~DgospreyApp, 73	NUM_JAC_DATA, 150
associateSyntax, 73	
	dynamic_viscosity PURE GAS, 161
DgospreyApp, 73	_ <i>_ '</i>
DgospreyApp, 73	dz
registerApps, 73	FINCH_DATA, 81
registerObjects, 74	e0
DgospreyApp.h, 202	GMRESRP_DATA, 115
validParams< DgospreyApp >, 203	e0_bar
DgospreyRevision.h, 203	
DGOSPREY_REVISION, 203	GMRESRP_DATA, 115
diagonalSolve	e1
Matrix, 139	ARNOLDI_DATA, 12
dim_mis_match	EGRET_HPP_
error.h, 209	egret.h, 206
Dirichlet	eMax
FINCH_DATA, 80	magpie.h, 241
DirichletBC	mSPD_DATA, 149
SCOPSOWL_DATA, 166	edit
SKUA_DATA, 174	Matrix, 140
dirichletBCFill	egret.h, 204
Matrix, 139	CE3, 206
discretize	calculate_properties, 207
FINCH_DATA, 80	Cstd, 206
Dispersion, 74	D_ii, 206
_dispersion, 75	D_ij, 206
index, 75	Dp_ij, 206
computeQpJacobian, 74	EGRET_HPP_, 206
computeQpResidual, 74	FilmMTCoeff, 206
Dispersion, 74	initialize_data, 207
Dispersion.h, 204	Mu, 206
validParams< Dispersion >, 204	Nu, 206
Display	PE3, 206
Matrix, 139	PSI, 206
Dk	Po, 206
scopsowl.h, 252	Pstd, 207
	RE3, 207
Dn EINCH DATA 90	ReNum, 207
FINCH_DATA, 80	Rstd, 207
Dnp1	ScNum, 207
FINCH_DATA, 81	set_variables, 207
Do Single Bata of	empirical_kf
FINCH_DATA, 81	• —
	skua.h, <mark>260</mark>

error.h, 209 eps  NUM_JAC_DATA, 150 PJFNK_DATA, 158 error error.h, 210 error.h, 210 error.h, 210 error.h, 210 error.h, 210 error.h, 210 arg_matrix_same, 210 arg_matrix_same, 210 dim_mis_match, 209 duplicate_variable, 210 empty_matrix, 209 file_dne, 209 generic_error, 209 indexing_error, 209 indexing_error, 209 invalid_components, 209 invalid_console_input, 210 invalid_gatom, 210 invalid_gatom, 210 invalid_molerraction, 209 invalid_molerraction, 209 invalid_molerraction, 200 invalid_proton, 210 invalid_species, 210 invalid_proton, 210 magpie, pverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 non_real_edge, 210 non_square_matrix, 209 onto_check_fail, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 simulation_antiv, 209 simulation_fail, 209 si		
eps     NUM_JAC_DATA, 150     PJFNK_DATA, 158     error     error, 210     error, 210     error, 210     error, 210     error, 210     error, 210     error, 209     error, 209     error, 209     error, 209     error, 209     duplicate variable, 210     emptmatrix, 209     file_dne, 209     generic_error, 209     indexing_error, 209     indexing_error, 209     invalid_boolean, 209     invalid_boolean, 209     invalid_console_input, 210     invalid_gas_sum, 209     invalid_norm, 210     invalid_size, 210     invalid_species, 210     invalid_species, 210     invalid_species, 210     invalid_species, 210     invalid_species, 210     invalid_species, 210     invalid_valence, 210     key_not_found, 210     magpie_reverse_error, 209     matrix_to_small_210     negative_mass, 210     negative_mass, 210     negative_mass, 210     negative_mass, 210     non_square_matrix, 209     not_a_token, 210     on_real_edge, 210     on_neal_edge, 210     on_t_a_token, 210     onl_found, 200     read_error, 210     scenario_fail, 209     simulation_fail, 209     simulation_fai	empty_matrix	vector_out_of_bounds, 210
NUM_JAC_DATA, 150 PJFNK_DATA, 158 error error, 210 error, 210 error, 210 error, 210 error, 210 arg_matrix_same, 210 arg_matrix_same, 210 dim_mis_match, 209 duplicate_variable, 210 empty_matrix, 209 file_dne, 209 generic_error, 209 indexing_error, 209 indexing_error, 209 invalid_components, 209 invalid_components, 209 invalid_components, 209 invalid_fraction, 210 invalid_gas_sum, 209 invalid_neutron, 210 invalid_neutron, 210 invalid_neutron, 210 invalid_norm, 210 invalid_norm, 210 invalid_species, 210 invalid	•	<del>-</del> '
PJFNK_DATA, 158 error errorh, 210 error.h, 210 error.h anchor_alias_dne, 210 ang_matrix_same, 210 dim_mis_match, 209 duplicate_variable, 210 empty_matrix, 209 file_dne, 209 generic_error, 209 indexing_error, 209 indexing_error, 209 invalid_boolean, 209 invalid_console_input, 210 invalid_sas_sum, 209 invalid_molertraction, 210 invalid_gas_sum, 209 invalid_molertraction, 210 invalid_patton, 210 invalid_patton, 210 invalid_patton, 210 invalid_proton, 210 invalid_proton, 210 invalid_species, 210 invalid_species, 210 invalid_species, 210 invalid_type, 210 invalid_species, 210 invalid_type, 210 invalid_type, 210 invalid_type, 210 no_diffusion, 210 mappie_reverse_error, 209 matrix_too_small, 210 mappie_reverse_error, 209 matrix_too_small, 210 no_mon_real_edge, 210 no_odiffusion, 210 no_file_tiven, 210 no_diffusion, 210 no_file_tiven, 210 no_diffusion, 210 no_file_tiven, 210 no_file_tiven	·	
error error.h, 210 error.h anchor_alias_dne, 210 arg_matrix_same, 210 dim_mis_match, 209 duplicate_variable, 210 empty_matrix, 209 generic_error, 209 indexing_error, 209 invalid_components, 209 invalid_components, 209 invalid_components, 209 invalid_components, 209 invalid_components, 209 invalid_components, 209 invalid_molerraction, 210 invalid_gas_sum, 209 invalid_norm, 210 invalid_species, 210 invalid_		
error.h anchor alias dne, 210 arg_matrix_same, 210 dim_mis_match, 209 duplicate variable, 210 empty_matrix, 209 file_dne, 209 generic_error, 209 indexing_error, 209 indexing_error, 210 invalid_boolean, 209 invalid_console_input, 210 invalid_fraction, 210 invalid_fraction, 210 invalid_fraction, 210 invalid_norm, 210 invalid_poton, 210 invalid_species, 210 invalid_species, 210 invalid_species, 210 invalid_species, 210 invalid_valence, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 magpie_reverse_error, 209 matrix_too_small, 210 magpie_reverse_error, 209 matrix_too_small, 210 mon_real_edge, 210 non_feal_edge, 210 non_real_edge, 210 not_a_token, 210 val_poor to_beck_fail, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 simulaton_fail, 209 simul	<del>-</del>	
error.h		
anchor_alias_dne, 210 arg_matrix_same, 210 dim_mis_match, 209 duplicate_variable, 210 empty_matrix, 209 file_dne, 209 generic_error, 209 indexing_error, 209 indexing_error, 209 indexing_error, 210 invalid_atom, 210 invalid_components, 209 invalid_components, 209 invalid_components, 200 invalid_faction, 210 invalid_gas_sum, 209 invalid_neutron, 210 invalid_proton, 210 invalid_proton, 210 invalid_solid_sum, 209 invalid_solid_sum, 209 invalid_solid_sum, 209 invalid_species, 210 invalid_valence, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_loo_small, 210 matvec_mis_match, 210 megative_mass, 210 negative_mass, 210 negative_mass, 210 negative_mass, 210 non_real_edge, 210 non_real_edge, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 not_founds, 209 read_error, 210 cot_o_beck_fail, 210 out_of_bounds, 209 simulation_ali, 209 simulation_fail, 209 simulation_fail		
arg_matrix_same, 210 dim_mis_match, 209 duplicate_variable, 210 empty_matrix, 209 file_dne, 209 generic_error, 209 indexing_error, 209 indexing_error, 210 invalid_boolean, 209 invalid_components, 209 invalid_console_input, 210 invalid_fraction, 210 invalid_fraction, 210 invalid_gas_sum, 209 invalid_neutron, 210 invalid_neutron, 210 invalid_proton, 210 invalid_proton, 210 invalid_size, 210 invalid_size, 210 invalid_size, 210 invalid_type, 210 invalid_type, 210 invalid_type, 210 invalid_valence, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 magpie_reverse_error, 209 matrix_too_small, 210 negative_mass, 210 negative_mass, 210 non_real_edge, 210 non_support, 209 ortho_check_fail, 210 out_of_bounds, 209 simulation_fail, 209 simula		
dim_mis_match, 209         eval_GPAST           duplicate_variable, 210         magpie.h, 242           empty_matrix, 209         scOPSOWL_DATA, 166           generic_error, 209         scOPSOWL_DATA, 166           indexing_error, 209         scOPSOWL_DATA, 166           invalid_boolean, 209         scOPSOWL_DATA, 166           invalid_components, 209         magpie.h, 242           invalid_components, 209         magpie.h, 242           invalid_components, 209         scOPSOWL_DATA, 166           invalid_components, 209         scOPSOWL_DATA, 166           invalid_components, 209         scOPSOWL_DATA, 166           invalid_components, 209         scOPSOWL_DATA, 166           invalid_pas_sum, 209         magpie.h, 242           invalid_pas_sum, 209         magpie.h, 242           invalid_neutron, 210         magpie.h, 242           invalid_neutron, 210         magpie.h, 242           invalid_size, 210         magpie.h, 243           invalid_size, 210         scOPSOWL_DATA, 166           invalid_solid_sum, 209         scOPSOWL_DATA, 166           invalid_type, 210         scOPSOWL_DATA, 166           invalid_type, 210         scOPSOWL_DATA, 166           invalid_type, 210         scopsimulation, 210           magpie.h, 242		
duplicate_variable, 210 empty_matrix, 209 file_dne, 209 generic_error, 209 indexing_error, 209 indial_error, 210 invalid_boolean, 209 invalid_components, 209 invalid_components, 209 invalid_electron, 210 invalid_gas_sum, 209 invalid_noterfraction, 209 invalid_norm, 210 invalid_proton, 210 invalid_size, 210 invalid_species, 210 invalid_species, 210 invalid_type, 210 inva	<del>-</del>	<del>-</del> · · · ·
empty_matrix, 209 file_dne, 209 generic error, 209 indexing_error, 209 initial_error, 210 invalid_boolean, 209 invalid_components, 209 invalid_components, 209 invalid_lelectron, 210 invalid_gas_sum, 209 invalid_neutron, 210 invalid_notefraction, 209 invalid_notefraction, 209 invalid_notefraction, 210 invalid_proton, 210 invalid_size, 210 invalid_species, 210 invalid_species, 210 invalid_valence, 210 inval	:	<del>_</del>
file_dne, 209     generic_error, 209     generic_error, 209     indexing_error, 209     indexing_error, 209     indexing_error, 209     indexing_error, 209     indexing_error, 209     invalid_boolean, 200     invalid_boolean, 209     invalid_console_input, 210     invalid_cectron, 210     invalid_electron, 210     invalid_gas_sum, 209     invalid_noelfraction, 209     invalid_noelfraction, 210     invalid_noelfraction, 210     invalid_solid_sum, 209     invalid_solid_sum, 209     invalid_solid_sum, 209     invalid_species, 210     invalid_species, 210     invalid_valence, 210     key_not_found, 210     matypie_reverse_error, 209     matrix_too_small_210     matvec_mis_match, 210     negative_mass, 210     negative_mass, 210     negative_time, 210     no_square_matrix, 209     ortho_check_fail, 210     out_of_bounds, 209     read_error, 210     scopsowl_DATA, 166     eval_po_qo     magpie.h, 242     eval_po_qo     magpie.h, 243     eval_po_qo     magpie.h, 243     eval_po_qo     magpie.h, 243     eval_po_qo     magpie.h, 243     eval_po_qo     magpie.h, 245     eval_po_qo     magpie.h, 245     eval_po_qo     magpie.h, 242     eval_po_qo     magpie.h, 242     eval_po_qo     magpie.h, 242     eval_po_qo     magpie.h, 243     eval_po_qo     magpie.h, 245     eval_po_qo     magpie.h, 245     eval_po_qo     magpie.h, 246     eval_po_qo     magpie.h, 247     eval_po_qo     magpie.h, 248     eval_po_qo     magpie.h, 248     eval_po_qo     magpie.h, 242     eval_po_qo     magpie.h, 242     eval_po_qo     magpie.h, 242     eval_po_qo	• —	
generic_error, 209 indexing_error, 209 indexing_error, 200 initial_error, 210 invalid_atom, 210 invalid_boolean, 209 invalid_components, 209 invalid_components, 209 invalid_components, 209 invalid_lelectron, 210 invalid_gas_sum, 209 invalid_molefraction, 210 invalid_neutron, 210 invalid_neutron, 210 invalid_solid_sum, 209 invalid_solid_sum, 209 invalid_species, 210 invalid_syecies, 210 invalid_tylence, 21	• •	<del>-</del>
indexing_error, 209 initial_error, 210 invalid_atom, 210 invalid_boolean, 209 invalid_components, 209 invalid_components, 209 invalid_lelectron, 210 invalid_gas_sum, 209 invalid_neletron, 210 invalid_sproton, 21		_ ′
initial_error, 210 invalid_atom, 210 invalid_boolean, 209 invalid_components, 209 invalid_console_input, 210 invalid_lelectron, 210 invalid_gas_sum, 209 invalid_gas_sum, 209 invalid_molefraction, 209 invalid_molefraction, 209 invalid_molefraction, 210 invalid_molefraction, 209 invalid_norm, 210 invalid_proton, 210 invalid_size, 210 invalid_size, 210 invalid_species, 210 invalid_species, 210 invalid_valence, 210 key_not_found, 210 mappie_reverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 megative_time, 210 no_diffusion, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_error, 210 nullptr_error, 210 nullptr_error, 210 nullptr_error, 210 rxn_rate_error, 210 scenario_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210 creative_mase, 210 creative_mase, 210 rxn_rate_error, 210 scenario_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210 creative_impet magpie.h, 242 eval_pc magpie.h, 242 eval_po magpie.h, 242 eva	· -	
invalid_atom, 210 invalid_boolean, 209 invalid_components, 209 invalid_console_input, 210 invalid_lectron, 210 invalid_gas_sum, 209 invalid_molefraction, 210 invalid_neutron, 210 invalid_neutron, 210 invalid_neutron, 210 invalid_nolefraction, 209 invalid_norm, 210 invalid_solid_sum, 209 invalid_solid_sum, 209 invalid_species, 210 invalid_species, 210 invalid_species, 210 invalid_species, 210 invalid_valence, 210 key_not_found, 210 matypie_reverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 negative_mass, 210 negative_mass, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullpt_error, 210 nullpt_func, 210 opt_no_support, 209 ortho_check_fail, 210 opt_no_support, 209 simulation_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210 cn, 80 cn_table cval_eta magpie.h, 242 eval_po_Pl magpie	<u>~_</u>	<del>-</del>
invalid_boolean, 209 invalid_components, 209 invalid_console_input, 210 invalid_electron, 210 invalid_fraction, 210 invalid_gas_sum, 209 invalid_neutron, 210 invalid_neutron, 210 invalid_proton, 210 invalid_proton, 210 invalid_size, 210 invalid_species, 210 invalid_species, 210 invalid_type, 210 invalid_proton, 210 invalid_proto		<del>-</del>
invalid_components, 209 invalid_console_input, 210 invalid_electron, 210 invalid_fraction, 210 invalid_gas_sum, 209 invalid_molefraction, 209 invalid_neutron, 210 invalid_proton, 210 invalid_size, 210 invalid_size, 210 invalid_slize, 210 invalid_slize, 210 invalid_species, 210 invalid_valence, 210 invalid_valence, 210 magpie_reverse_error, 209 matrix_loo_small, 210 matvec_mis_match, 210 no_diffusion, 210 no_diffusion, 210 non_real_edge, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_func, 210 nullptr_func, 210 opt_no_support, 209 read_error, 210 scenario_fail, 209 simulation_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  livalid_console_input, 210 magpie_lectron, 210 magpie_h, 242 eval_po_Pl magpie.h, 242 eval_po_Po magpie.h, 242 eval_po_Po magpie.h, 242 eval_po_Pl magpie.hevaleplane folapon magpie.h, 243 eval_real po_Pl magpie.hevaleplane po_Pl magpie.hevaleplane po_Pl magpie	<del>-</del>	
invalid_console_input, 210 invalid_electron, 210 invalid_graction, 210 invalid_gas_sum, 209 invalid_molefraction, 209 invalid_nometraction, 210 invalid_soletron, 210 invalid_proton, 210 invalid_solid_sum, 209 invalid_solid_sum, 210 magpie_reverse_error, 209 matrix_loo_small, 210 matvec_mis_match, 210 missing_information, 210 negative_time, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_error, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 simulation_fail, 209 singular_matrix, 210 unregistered_name, 210  SCOPSOWL_DATA, 166 eval_po_qu magpie.h, 242 eval_po_qu magpie.h, 243 eval_retard SCOPSOWL_DATA, 166 eval_retard  FINCH_DATA, 81 F		
invalid_electron, 210 invalid_fraction, 210 invalid_gas_sum, 209 invalid_molefraction, 209 invalid_neutron, 210 invalid_norm, 210 invalid_norm, 210 invalid_size, 210 invalid_size, 210 invalid_size, 210 invalid_solid_sum, 209 invalid_species, 210 invalid_species, 210 invalid_species, 210 invalid_type, 210 invalid_valence, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 megative_mass, 210 negative_time, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_error, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  SKUA_DATA, 174 eval_po magpie.h, 242 eval_po_Pl magpie.h, 243 eval_real scorp.Pl magpie.h, 243 eval_real scorp.Pl magpie.h, 243 eval_real eval_real scorp.Pl magpie.h, 243 eval_real scorp.Pl magpie.h, 243 eval_real scorp.Pl magpie.h 242 eval_po_Pl magpie.h 242 eval_po_Pl madein_poup.pl scorp.Pl magpie.h 243 eval_real scorp.Pl magpie.h 243 eval		
invalid_fraction, 210 invalid_gas_sum, 209 invalid_molefraction, 209 invalid_molefraction, 210 invalid_neutron, 210 invalid_norm, 210 invalid_size, 210 invalid_solid_sum, 209 invalid_valence, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 megative_time, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_error, 210 nullptr_error, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 simulation_fail, 209 simulation_fail, 209 singular_matrix, 210 unregistered_name, 210  reval_po_Pl magpie.h, 242 eval_po_Pl magpie.h pagle hevel_surdine scopPsoul_data	_ ·	
invalid_molefraction, 209 invalid_neutron, 210 invalid_proton, 210 invalid_proton, 210 invalid_size, 210 invalid_solid_sum, 209 invalid_species, 210 invalid_species, 210 invalid_type, 210 invalid_valence, 210 key_not_found, 210 matypic_reverse_error, 209 matrix_too_small, 210 megative_mass, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 simulation_fail, 209 simulation_fail, 209 singular_matrix, 210 tensor_out_of_bounds, 210 unregistered_name, 210  invalid_norm, 210 magpie.h, 242 eval_po_Qo magpie.h, 243 eval_po_Qo magpie.h, 242 eval_po_Qo magpie.h, 243 eval_po_Qo magpie.h, 242 eval_po_Qo magpie.h, 243 eval_po_Qo magpie.h, 243 eval_po_Qo magpie.h, 243 eval_retard SCOPSOWL_DATA, 166 eval_sur/Diff SCOPSOWL_DATA, 166 e		eval_po
invalid_neutron, 210 invalid_norm, 210 invalid_norm, 210 invalid_size, 210 invalid_size, 210 invalid_solid_sum, 209 invalid_species, 210 invalid_species, 210 invalid_type, 210 invalid_valence, 210 key_not_found, 210 matypie_reverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 megative_mass, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_func, 210 nullptr_func, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  invalid_size, 210 magpie.h, 242 eval_po_qo magpie.h, 243 eval_retard eval_retard SCOPSOWL_DATA, 166 eval_surfDiff SCOPSOWL_DA	invalid_gas_sum, 209	magpie.h, 242
invalid_norm, 210 invalid_proton, 210 invalid_size, 210 invalid_solid_sum, 209 invalid_species, 210 invalid_species, 210 invalid_type, 210 invalid_valence, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 megative_mass, 210 negative_time, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_func, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  invalid_spice, 210 eval_retard scOPSOWL_DATA, 166 eval_surfDiff SCOPSOWL_DA	invalid_molefraction, 209	eval_po_PI
invalid_proton, 210 invalid_size, 210 invalid_size, 210 invalid_species, 210 invalid_species, 210 invalid_species, 210 invalid_type, 210 invalid_valence, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 missing_information, 210 negative_time, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_error, 210 opt_no_support, 209 ortho_check_fail, 210 orx_rate_error, 210 scenario_fail, 209 singular_matrix, 210 tensor_out_of_bounds, 210 unregistered_name, 210  invalid_size, 210 cval_serid scopSOWL_DATA, 166 eval_serId scopSOWL_DATA, 166 evaleserId scopSOWL_DATA, 166 evaleserId scopSOWL_DATA, 166 evaleserId scopSOW	invalid_neutron, 210	magpie.h, 242
invalid_size, 210 invalid_solid_sum, 209 invalid_species, 210 invalid_species, 210 invalid_type, 210 invalid_type, 210 invalid_valence, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 missing_information, 210 negative_mass, 210 negative_time, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 ovt_of_bounds, 209 read_error, 210 scenario_fail, 209 simulation_fail, 209 singular_matrix, 210 tensor_out_of_bounds, 210 unregistered_name, 210  invalid_species, 210 eval_retard sCOPSOWL_DATA, 166 eval_surfDiff SCOPSOWLDATA, 166 eval_surfDiff S	invalid_norm, 210	eval_po_qo
invalid_solid_sum, 209 invalid_species, 210 invalid_type, 210 invalid_type, 210 invalid_type, 210 invalid_type, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 mative_mis_match, 210 megative_mass, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210 unregistered_name, 210 unregistered_name, 210 unregistered_name, 210  SCOPSOWL_DATA, 166 eval_surfDiff SCOPSOME_Data, 166 eval_surfD	invalid_proton, 210	magpie.h, 243
invalid_species, 210 invalid_type, 210 invalid_type, 210 invalid_valence, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 missing_information, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 simulation_fail, 209 singular_matrix, 210 tensor_out_of_bounds, 210 unregistered_name, 210 eval_surfDiff SCOPSOWL_DATA, 166 evalprecon FINCH_DATA, 81 evalres FINCH_DATA, 81 ExplicitFlux FINCH_DATA, 81 FINCH_DATA, 158 FINCH_Picard finch.h, 213 FOM lark.h, 226 fC_E FINCH_DATA, 81 fC_I FINCH_DATA, 81 fC_I FINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_I, 79 CL_E, 80 CR_E, 80 CR_	invalid_size, 210	eval_retard
invalid_type, 210 invalid_valence, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 matveo_mis_match, 210 negative_mass, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 simulation_fail, 209 singular_matrix, 210 tensor_out_of_bounds, 210 invalid_valence, 210 invalid_valence, 210 scenario_tail, 209 singular_matrix, 210 tensor_out_of_bounds, 210 unregistered_name, 210  SCOPSOWL_DATA, 166 evalprecon FINCH_DATA, 81 evalres FINCH_DATA, 81  FXPJFNK_DATA, 81  FOM lark.h, 226 fC_E FINCH_DATA, 81  FC_I FINCH_DATA, 81  FINCH_DATA, 81  FINCH_DATA, 75 beta, 79 CC_E, 79 CC_I, 79 CL_E, 80 CR_E, 80 CR_E, 80 CR_E, 80 CR_E, 80	invalid_solid_sum, 209	SCOPSOWL_DATA, 166
invalid_valence, 210 key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 megative_mass, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nopt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 singular_matrix, 210 tensor_out_of_bounds, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  evalprecon FINCH_DATA, 81 evalprecon FINCH_DATA, 81 evalprecon FINCH_DATA, 81 FXINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 158 FINCH_Picard finch.h, 213 FOM lark.h, 226 fC_E FINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_I, 79 CC_I, 79 CL_E, 80 CN, 80 CR_E, 80 CR_E, 80 CR_E, 80	invalid_species, 210	eval_surfDiff
key_not_found, 210 magpie_reverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 missing_information, 210 negative_mass, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nopt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  FINCH_DATA, 81  ExplicitFlux FINCH_DATA, 81  FINCH_DATA, 81  FINCH_Picard finch.h, 213  FOM lark.h, 226  fC_E FINCH_DATA, 81  FO_I FINCH_DATA, 81  FINCH_DATA, 81  FINCH_DATA, 81  CC_E, 79  CC_E, 79  CC_I, 79  CL_E, 80  CR_E, 80  CR_E, 80  CR_E, 80	— ·	<del>_</del>
magpie_reverse_error, 209 matrix_too_small, 210 matvec_mis_match, 210 missing_information, 210 negative_mass, 210 no_diffusion, 210 no_diffusion, 210 no_square_matrix, 209 not_a_token, 210 nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  matrix_too_small, 210 ExplicitFlux FINCH_DATA, 81  FUNCH_DATA, 81  FUNCH_DATA, 158 FINCH_Picard finch.h, 213  FOM lark.h, 226  fC_E FINCH_DATA, 81  FC_I FINCH_DATA, 81  FINCH_DATA, 81  FINCH_DATA, 81  FINCH_DATA, 75  beta, 79 CC_E, 79 CC_I, 79 CL_E, 80 CR_E, 80 CR_E, 80 CR_E, 80 CR_E, 80 CR_E, 80	<del>_</del>	•
matrix_too_small, 210 matvec_mis_match, 210 missing_information, 210 negative_mass, 210 negative_time, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_func, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  FINCH_DATA, 81 ExplicitFlux FINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 158 FINCH_Picard finch.h, 213 FOM lark.h, 226  FINCH_DATA, 81 FOM lark.h, 226  FOM lark.h, 226  CC_E FINCH_DATA, 81 FINCH_DATA,	· ·	<del>-</del>
matvec_mis_match, 210 missing_information, 210 negative_mass, 210 negative_time, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_func, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  ExplicitFlux FINCH_DATA, 81 FINCH_DATA, 158 FINCH_Picard finch.h, 213 FOM lark.h, 226 fC_E FINCH_DATA, 81 fC_I FINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_I, 79 CL_E, 80 CN, 80 CR_E, 80 CR_E	· · · · · · · · · · · · · · · · · · ·	
missing_information, 210 negative_mass, 210 negative_time, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 notho_check_fail, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 siring_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  FINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_I, 79 CL_E, 80 CN, 80 CR_E,		<i>_ '</i>
negative_mass, 210 negative_time, 210 no_diffusion, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 singular_matrix, 210 tensor_out_of_bounds, 210 unregistered_name, 210  FINCH_DATA, 158 FINCH_Picard finch.h, 213 FOM lark.h, 226 fC_E FINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_I, 79 CL_E, 80 CR_E, 80 CR_E, 80 CR_E, 80		
negative_time, 210 no_diffusion, 210 no_real_edge, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  FINCH_DATA, 158 FINCH_Picard finch.h, 213 FOM lark.h, 226 fC_E FINCH_DATA, 81 fC_I FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_E, 79 CL_E, 80 CN, 80 CR_E, 80	<del>-</del>	FINCH_DATA, 81
no_diffusion, 210 no_real_edge, 210 non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  PJFNK_DATA, 158 FINCH_Picard finch.h, 213 FOM lark.h, 226 fC_E FINCH_DATA, 81 fC_I FINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_I, 79 CL_E, 80 CL_I, 80 CN, 80 CR_E, 80 CR_E, 80	<del>-</del> —	F
non_real_edge, 210 non_square_matrix, 209 not_a_token, 210 nullptr_error, 210 nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  FINCH_Picard finch.h, 213 FOM lark.h, 226 fC_E FINCH_DATA, 81 fC_I FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_I, 79 CL_E, 80 CN, 80 CR_E,	<del>-</del>	
non_square_matrix, 209  not_a_token, 210  nullptr_error, 210  nullptr_func, 210  opt_no_support, 209  ortho_check_fail, 210  out_of_bounds, 209  read_error, 210  rxn_rate_error, 210  scenario_fail, 209  simulation_fail, 209  string_parse_error, 210  tensor_out_of_bounds, 210  unregistered_name, 210  finch.h, 213  FOM  lark.h, 226  fC_E  FINCH_DATA, 81  FINCH_DATA, 81  FINCH_DATA, 75  beta, 79  CC_E, 79  CC_I, 79  CL_E, 80  CN, 80  CR_E, 80		<del>-</del> · · ·
not_a_token, 210 nullptr_error, 210 nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 simulation_fail, 209 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  FOM lark.h, 226 fC_E FINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_I, 79 CL_E, 80 CL_I, 80 CN, 80 CR_E, 80 CR_E	<b>~</b> .	<del>-</del>
nullptr_error, 210 nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 scenario_fail, 209 simulation_fail, 209 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  lark.h, 226 fC_E FINCH_DATA, 81 FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_I, 79 CL_E, 80 CL_I, 80 CR_E, 80 C	_ · _	
nullptr_func, 210 opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  fC_E FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_E, 79 CC_I, 79 CL_E, 80 CL_I, 80 CN, 80 CR_E, 80		
opt_no_support, 209 ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_E, 79 CC_I, 79 CL_E, 80 CL_I, 80 CN, 80 CR_E, 8	• —	
ortho_check_fail, 210 out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_E, 79 CC_I, 79 CL_E, 80 CL_I, 80 CN, 80 CR_E, 80	• —	FINCH_DATA, 81
out_of_bounds, 209 read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  FINCH_DATA, 81 FINCH_DATA, 75 beta, 79 CC_E, 79 CC_E, 79 CC_I, 79 CL_E, 80 CL_I, 80 CN, 80 CR_E, 80		fC_I
read_error, 210 rxn_rate_error, 210 scenario_fail, 209 simulation_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  FINCH_DATA, 75 beta, 79 CC_E, 79 CC_E, 79 CC_I, 79 CL_E, 80 CL_I, 80 CN, 80 CN, 80 CR_E, 80		FINCH_DATA, 81
rxn_rate_error, 210 beta, 79 scenario_fail, 209 CC_E, 79 simulation_fail, 209 CC_I, 79 singular_matrix, 210 CL_E, 80 string_parse_error, 210 CL_I, 80 tensor_out_of_bounds, 210 unregistered_name, 210 CR_E, 80		FINCH_DATA, 75
scenario_fail, 209 simulation_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  CC_E, 79 CC_I, 79 CL_E, 80 CL_E, 80 CN, 80 CN, 80 CR_E, 80		beta, 79
simulation_fail, 209 singular_matrix, 210 string_parse_error, 210 tensor_out_of_bounds, 210 unregistered_name, 210  CC_I, 79 CL_E, 80 CL_I, 80 CN, 80 CR_E, 80		CC_E, 79
singular_matrix, 210 CL_E, 80 string_parse_error, 210 CL_I, 80 tensor_out_of_bounds, 210 CR_E, 80 unregistered_name, 210 CR_E, 80		
string_parse_error, 210 CL_I, 80 tensor_out_of_bounds, 210 CR_E, 80 unregistered_name, 210 CR_E, 80		
tensor_out_of_bounds, 210 CN, 80 Unregistered_name, 210 CR_E, 80		
unregistered_name, 210 CR_E, 80		
unstable_matrix, 210	unstable_matrix, 210	
callroutine, 79		callroutine, 79

CheckMass, 80	setpostprocess, 85
d, 80	setpreprocess, 85
DIC, 80	settime, 85
Dirichlet, 80	Sn, 86
discretize, 80	Snp1, 86
Dn, 80	solve, 86
Dnp1, 81	SteadyState, 86
Do, 81	T, 86
dt, 81	t, 86
dt_old, 81	t_old, 86
dz, 81	tol_abs, 86
evalprecon, 81	tol_rel, 86
evalres, 81	total_iter, 86
ExplicitFlux, 81	u_star, 86
fC_E, 81	uAvg, 87
fC_I, 81	uAvg old, 87
fL E, 81	uIC, 87
fL I, 82	uT, <mark>8</mark> 7
fR_E, 82	uT_old, 87
fR_I, 82	ubest, 87
Fn, 82	un, 87
	un, 67 unm1, 87
Fnp1, 82	
gE, 82	unp1, 87
gl, 82	uo, 87
Iterative, 82	Update, 87
kIC, 82	uz_I_E, <mark>88</mark>
kfn, 82	uz_l_l, <mark>88</mark>
kfnp1, 82	uz_lm1_E, <mark>88</mark>
kn, 83	uz_lm1_l, 88
knp1, 83	uz_lp1_E, <mark>88</mark>
ko, 83	uz_lp1_l, 88
L, 83	vIC, 88
LN, 83	vn, <mark>88</mark>
lambda E, 83	vnp1, 88
lambda_I, 83	vo, 88
ME, 83	fL_E
MI, 83	FINCH_DATA, 81
	fL I
max_iter, 83	<del>_</del>
NE, 83	FINCH_DATA, 82
NI, 84	fR_E
nl_method, 84	FINCH_DATA, 82
NormTrack, 84	fR_I
OE, 84	FINCH_DATA, 82
OI, 84	file_dne
param_data, 84	error.h, 209
picard_dat, 84	film_transfer
pjfnk_dat, 84	SCOPSOWL_PARAM_DATA, 171
pres, 84	SKUA_PARAM, 177
RIC, 85	FilmMTCoeff
res, 84	egret.h, 206
resettime, 84	finch.h
Rn, 85	Cartesian, 213
Rnp1, 85	Cylindrical, 213
Ro, 85	FINCH_Picard, 213
s, 85	LARK_PJFNK, 214
setbcs, 85	LARK_Picard, 213
setic, 85	Spherical, 213
setparams, 85	finch.h, 211

check_Mass, 214	_molecular_diffusion, 92
default bcs, 214	molecular wieght, 92
default execution, 214	_pellet_density, 92
default_ic, 214	_pellet_heat_capacity, 92
default params, 214	_porosity, 92
default_postprocess, 214	_retardation, 92
default_precon, 214	_solid_conc, 92
default_preprocess, 214	_solid_perturb, 92
default res, 215	_temperature, 93
default_reset, 215	_total_pressure, 93
default solve, 215	_velocity, 93
default_timestep, 215	computeQpProperties, 90
finch_coord_type, 213	FlowProperties, 90
finch_solve_type, 213	FlowProperties, 90
I_direct, 215	FlowProperties.h, 218
lark_picard_step, 215	_gas_const, 219
max, 215	validParams< FlowProperties >, 219
min, 215	FluidHeatTransfer, 93
minmod, 215	_bed_wall_transfer_coeff, 94
minmod_discretization, 215	_inner_dia, 94
nl picard, 215	_porosity, 94
ospre_discretization, 216	_wall_temp, 94
print2file_dim_header, 216	computeQpJacobian, 94
print2file_newline, 216	computeQpResidual, 94
print2file_result_new, 216	FluidHeatTransfer, 94
print2file_result_old, 216	FluidHeatTransfer, 94
print2file_tab, 216	FluidHeatTransfer.h, 219
print2file_time_header, 216	validParams< FluidHeatTransfer >, 219
setup_FINCH_DATA, 216	Fn
uAverage, 217	FINCH_DATA, 82
uTotal, 217	Fnp1
vanAlbada_discretization, 217	FINCH_DATA, 82
finch_coord_type	fom
finch.h, 213	lark.h, 228
finch_dat	fun_call
SCOPSOWL_DATA, 166	BACKTRACK_DATA, 14
SKUA_DATA, 174	PJFNK_DATA, 158
finch_solve_type	funeval
finch.h, 213	PJFNK_DATA, 158
Fk	Fv
BACKTRACK_DATA, 14	PJFNK_DATA, 158
flock.h, 217	Fx
FlowProperties, 88	NUM_JAC_DATA, 150
_column_length, 91	Fxp
_comp_Sutherland_const, 91	NUM_JAC_DATA, 150
_comp_heat_capacity, 91	COR
_comp_ref_temp, 91	GCR
_comp_ref_viscosity, 91	lark.h, 226
_dispersion, 91	GMRESLP
_flow_rate, 91	lark.h, 226
_gas_conc, 91	GMRESR
_gas_density, 91	lark.h, 226
_gas_heat_capacity, 91	GMRESRP lark.h, 226
_gas_molecular_wieght, 91	GAdvection, 94
_gas_viscosity, 91	_velocity, 95
_heat_retardation, 92	_velocity, 95 _vx, 95
_index, 92	_vx, 95 _vy, 95
_inner_dia, 92	_vz, 95
	, ••

computeQpJacobian, 95	computeQpResidual, 98
computeQpResidual, 95	GColumnHeatAdvection, 98
GAdvection, 95	GColumnHeatAdvection, 98
GAdvection, 95	GColumnHeatAdvection.h, 220
GAdvection.h, 219	validParams< GColumnHeatAdvection >, 220
validParams< GAdvection >, 219	GColumnHeatDispersion, 99
GAnisotropicDiffusion, 95	_Diffusion, 100
_Diffusion, 96	_Dxx, 100
_Dxx, 96	_Dxy, 100
_Dxy, 97	_Dxz, 100
_Dxz, 97	_Dyx, 100
_Dyx, 97	_Dyy, 100
_Dyy, <mark>97</mark>	_Dyz, 100
_Dyz, <del>9</del> 7	_Dzx, 100
_Dzx, 97	_Dzy, 100
Dzy, 97	_Dzz, 100
_Dzz, 97	_conductivity, 100
computeQpJacobian, 96	computeQpJacobian, 100
computeQpResidual, 96	computeQpResidual, 100
GAnisotropicDiffusion, 96	GColumnHeatDispersion, 100
GAnisotropicDiffusion, 96	GColumnHeatDispersion, 100
GAnisotropicDiffusion.h, 219	GColumnHeatDispersion.h, 220
validParams< GAnisotropicDiffusion >, 220	validParams< GColumnHeatDispersion >, 220
GCR_DATA, 104	GColumnMassAdvection, 101
alpha, 105	_vel, 102
bestres, 105	_velocity, 102
bestx, 106	_vx, 102
beta, 106	_vy, 102 _vy, 102
breakdown, 106	_vz, 102 _vz, 102
c, 106	computeQpJacobian, 101
	computeQpacobian, 101 computeQpResidual, 102
c_temp, 106	
iter_inner, 106	GColumnMassAdvection, 101
iter_outer, 106	GColumnMassAdvection, 101
maxit, 106	GColumnMassAdvection.h, 220
Output, 106	validParams< GColumnMassAdvection >, 221
r, 106	GColumnMassDispersion, 102
relres, 106	_Diffusion, 103
relres_base, 107	_Dxx, 103
res, 107	_Dxy, 103
restart, 107	_Dxz, 103
tol_abs, 107	_Dyx, 104
tol_rel, 107	_Dyy, 104
total_iter, 107	_Dyz, 104
transpose_dat, 107	_Dzx, 104
u, 107	_Dzy, 104
u_temp, 107	_Dzz, 104
x, 107	_dispersion, 103
GCR_Output	_index, 104
GMRESR_DATA, 111	_molecular_diffusion, 104
GColumnHeatAdvection, 97	computeQpJacobian, 103
_gas_density, 98	computeQpResidual, 103
_gas_heat_capacity, 98	GColumnMassDispersion, 103
_vel, 98	GColumnMassDispersion, 103
_velocity, 98	GColumnMassDispersion.h, 221
_vx, 98	validParams< GColumnMassDispersion >, 221
vy, 98	gE
_vz, 99	FINCH_DATA, 82
computeQpJacobian, 98	gl

FINCH_DATA, 82	tol_abs, 116
GMRES_Output	tol rel, 116
GMRESR DATA, 112	v, 116
GMRESLP_DATA, 108	Vk, 116
arnoldi dat, 109	w, 116
bestres, 109	x, 116
bestx, 109	y, 116
	-
iter, 109	Zk, 117
maxit, 109	GPAST_DATA, 117
Output, 109	gama_inf, 118
r, 109	He, 118
relres, 109	Plo, 118
relres_base, 109	po, 118
res, 109	poi, 118
restart, 109	present, 118
steps, 109	q, 118
tol_abs, 110	qo, 118
tol_rel, 110	x, 118
x, 110	y, 118
GMRESR_DATA, 110	GSTA_DATA, 119
arg, 111	dHo, 119
GCR_Output, 111	dSo, 119
GMRES Output, 112	m, 119
gcr_abs_tol, 111	qmax, 119
gcr_dat, 111	gama
gcr_maxit, 111	mSPD_DATA, 149
gcr_rel_tol, 112	gama inf
gcr_restart, 112	GPAST DATA, 118
gmres_dat, 112	_ <i> </i>
-	gas_dat
gmres_maxit, 112	SCOPSOWL_DATA, 167
gmres_restart, 112	SKUA_DATA, 174
gmres_tol, 112	gas_temperature
iter_inner, 112	MIXED_GAS, 147
iter_outer, 112	SCOPSOWL_DATA, 167
matvec, 112	gas_velocity
matvec_data, 112	SCOPSOWL_DATA, 167
N, 113	SKUA_DATA, 174
term_precon, 113	gcr
terminal_precon, 113	lark.h, 229
total_iter, 113	gcr_abs_tol
GMRESRP_DATA, 113	GMRESR_DATA, 111
bestres, 114	gcr_dat
bestx, 114	GMRESR_DATA, 111
e0, 115	PJFNK_DATA, 158
e0_bar, 115	gcr_maxit
H, 115	GMRESR_DATA, 111
H_bar, 115	gcr_rel_tol
iter_inner, 115	GMRESR DATA, 112
iter_outer, 115	gcr restart
iter_total, 115	GMRESR_DATA, 112
maxit, 115	generic_error
Output, 115	error.h, 209
r, 115	gmres_dat
relres, 115	GMRESR_DATA, 112
relres_base, 116	
	gmres_in
res, 116	KMS_DATA, 122
restart, 116	gmres_maxit
sum, 116	GMRESR_DATA, 112

gmres_out	error.h, 210
KMS_DATA, 122	initialGuess_mSPD
gmres_restart	magpie.h, 243
GMRESR_DATA, 112	initialize_data
gmres_tol	egret.h, 207
GMRESR_DATA, 112	inner_iter
gmresLeftPreconditioned	KMS_DATA, 122
lark.h, 230	inner_product
gmresRightPreconditioned	Matrix, 140
lark.h, 231	inner_reltol
gmreslp_dat	KMS_DATA, 122
PJFNK_DATA, 158	IntegralAvg
gmresr	Matrix, 140
lark.h, 230	IntegralTotal
gmresr_dat	Matrix, 140
PJFNK_DATA, 158	invalid_atom
gmresrPreconditioner	error.h, 210
lark.h, 232	invalid boolean
gmresrp_dat	error.h, 209
PJFNK DATA, 158	invalid_components
gpast_dat	error.h, 209
MAGPIE DATA, 128	invalid_console_input
grad mSPD	error.h, 210
magpie.h, 243	invalid electron
gsta_dat	error.h, 210
MAGPIE DATA, 128	invalid fraction
······································	error.h, 210
Н	invalid_gas_sum
GMRESRP DATA, 115	error.h, 209
H bar	invalid molefraction
GMRESRP_DATA, 115	error.h, 209
He	invalid_neutron
GPAST DATA, 118	error.h, 210
magpie.h, 240	invalid norm
HeatConductivity, 119	error.h, 210
_conductivity, 120	invalid proton
_porosity, 120	error.h, 210
computeQpJacobian, 120	invalid size
computeQpResidual, 120	error.h, 210
HeatConductivity, 120	invalid_solid_sum
HeatConductivity, 120	error.h, 209
HeatConductivity.h, 221	invalid_species
validParams< HeatConductivity >, 221	error.h, 210
Heterogeneous	invalid_type
SCOPSOWL_DATA, 167	error.h, 210
Hkp1	invalid_valence
ARNOLDI_DATA, 13	error.h, 210
hp1	
ARNOLDI_DATA, 13	inverse
ALINOLDI_DATA, TO	Matrix, 140
1	iter
SYSTEM_DATA, 179	ARNOLDI_DATA, 13
Ideal	BICGSTAB_DATA, 23
SYSTEM DATA, 179	CGS_DATA, 26
li	GMRESLP_DATA, 109
OPTRANS DATA, 151	PCG_DATA, 152
indexing_error	PICARD_DATA, 155
error.h, 209	iter_inner
initial error	GCR_DATA, 106
IIII(IαI_∇IIVI	

GMRESR DATA, 112	krylov_method
GMRESRP DATA, 115	lark.h, 225
iter_outer	krylovMultiSpace
GCR DATA, 106	lark.h, 232
GMRESR DATA, 112	,
GMRESRP DATA, 115	L
<del>-</del> · · · · · · · · · · · · · · · · · · ·	FINCH DATA, 83
iter_total	LARK PJFNK
GMRESRP_DATA, 115	finch.h, 214
Iterative	LARK Picard
FINCH_DATA, 82	finch.h, 213
J	L_Output
SYSTEM_DATA, 179	PJFNK_DATA, 159
jacvec	I_direct
lark.h, 232	finch.h, 215
	I_iter
K	PJFNK_DATA, 159
SYSTEM_DATA, 179	LN
k	FINCH_DATA, 83
ARNOLDI_DATA, 13	ladshawSolve
kB	Matrix, 140
magpie.h, 240	lambda E
kIC	FINCH DATA, 83
FINCH DATA, 82	lambda I
KMS_DATA, 120	FINCH DATA, 83
gmres_in, 122	lambdaMin
gmres_out, 122	BACKTRACK DATA, 15
inner_iter, 122	lark.h
inner_reltol, 122	BiCGSTAB, 226
level, 122	CGS, 226
matvec, 122	FOM, 226
matvec_data, 122	GCR, 226
max_level, 122	GMRESLP, 226
maxit, 122	GMRESR, 226
outer_abstol, 122	GMRESRP, 226
outer_iter, 122	PCG, 226
outer_reltol, 122	lark.h, 221
Output_in, 123	arnoldi, 226
Output_out, 123	backtrackLineSearch, 226
restart, 123	bicgstab, 227
term precon, 123	cgs, 228
terminal precon, 123	fom, 228
total iter, 123	gcr, 229
key_not_found	gmresLeftPreconditioned, 230
error.h, 210	gmresRightPreconditioned, 231
kfn	gmresr, 230
FINCH_DATA, 82	· ·
	gmresrPreconditioner, 232
kfnp1	jacvec, 232
FINCH_DATA, 82	kmsPreconditioner, 232
kinematic_viscosity	krylov_method, 225
MIXED_GAS, 147	krylovMultiSpace, 232
kmsPreconditioner	MIN_TOL, 225
lark.h, 232	NumericalJacobian, 233
kn	operatorTranspose, 233
FINCH_DATA, 83	pcg, 234
knp1	picard, 235
FINCH_DATA, 83	pjfnk, <mark>235</mark>
ko	update_arnoldi_solution, 236
FINCH_DATA, 83	lark_picard_step
_ ,	

finch.h, 215	mspd_dat, 128
level	sys_dat, 128
KMS_DATA, 122	MAGPIE_HeatAccumulation, 128
SCOPSOWL_DATA, 167	_pellet_density, 129
lin_tol_abs	_porosity, 129
PJFNK_DATA, 159	_solid_heat, 129
lin_tol_rel	_solid_heat_old, 129
PJFNK_DATA, 159	computeQpJacobian, 129
LineSearch	computeQpResidual, 129
PJFNK_DATA, 159	MAGPIE HeatAccumulation, 129
linear solver	MAGPIE_HeatAccumulation, 129
PJFNK_DATA, 159	MAGPIE HeatAccumulation.h, 247
LinearDrivingForce, 123	MAGPIE_Perturbation, 129
_coef, 124	_index, 130
driving_value, 124	_magpie_dat, 130
gaining, 124	computeValue, 130
_var, 124	MAGPIE_Perturbation, 130
computeQpJacobian, 124	MAGPIE Perturbation, 130
computeQpResidual, 124	MAGPIE_Perturbation.h, 248
LinearDrivingForce, 124	validParams< MAGPIE_Perturbation >, 249
LinearDrivingForce, 124	ME
LinearDrivingForce.h, 236	FINCH_DATA, 83
validParams< LinearDrivingForce >, 237	mError
Inko	-
	error.h, 209 MI
magpie.h, 241	
Inact_mSPD	FINCH_DATA, 83
magpie.h, 243	MIN_TOL
lowerHessenberg2Triangular	lark.h, 225
Matrix, 141	MIXED_GAS, 146
lowerHessenbergSolve	binary_diffusion, 147
Matrix, 141	char_length, 147
lowerTriangularSolve	CheckMolefractions, 147
Matrix, 141	gas_temperature, 147
m	kinematic_viscosity, 147
GSTA DATA, 119	molefraction, 147
	N, 147
M_PI	Reynolds, 147
macaw.h, 238	species_dat, 147
MAGPIE	total_density, 147
magpie.h, 244	total_dyn_vis, 147
MAGPIE_Adsorption, 124	total_molecular_weight, 147
_index, 126	total_pressure, 148
_magpie_dat, 126	total_specific_heat, 148
computeValue, 125	velocity, 148
MAGPIE_Adsorption, 125	mSPD_DATA, 148
MAGPIE_Adsorption, 125	eMax, 149
MAGPIE_Adsorption.h, 246	eta, 149
validParams< MAGPIE_Adsorption >, 246	gama, 149
MAGPIE_AdsorptionHeat, 126	s, 149
_index, 127	v, 149
_magpie_dat, 127	macaw.h, 237
_solid_conc, 127	M_PI, 238
computeValue, 127	magpie.h, 238
MAGPIE_AdsorptionHeat, 127	A, 240
MAGPIE_AdsorptionHeat, 127	DBL_EPSILON, 240
MAGPIE_AdsorptionHeat.h, 247	dq_dp, 241
MAGPIE_DATA, 127	eMax, 241
gpast_dat, 128	eval_GPAST, 242
gsta_dat, 128	

eval_eta, 242	columnExtract, 138
eval_po, 242	columnProjection, 138
eval_po_PI, 242	columnReplace, 138
eval_po_qo, 243	columnShrink, 139
grad_mSPD, 243	columnVectorFill, 139
He, 240	columns, 139
initialGuess mSPD, 243	ConstantICFill, 139
kB, 240	Data, 145
InKo, 241	determinate, 139
Inact mSPD, 243	diagonalSolve, 139
MAGPIE, 244	dirichletBCFill, 139
·	
Na, 241	Display, 139
PI, 244	edit, 140
Po, 241	inner_product, 140
q_p, 245	IntegralAvg, 140
qT, 245	IntegralTotal, 140
qo, 245	inverse, 140
Qst, 245	ladshawSolve, 140
R, 241	lowerHessenberg2Triangular, 141
shapeFactor, 241	lowerHessenbergSolve, 141
V, 241	lowerTriangularSolve, 141
Z, 241	Matrix, 137
magpie reverse error	naturalLaplacian3D, 141
error.h, 209	norm, 141
magpie_dat	num_cols, 145
SCOPSOWL_DATA, 167	num_rows, 145
SKUA DATA, 174	operator*, 142
<del>-</del> · · · · ·	•
MagpieAdsorbateProperties, 131	operator(), 141
_enthalpy_1, 133	operator+, 142
_enthalpy_2, 133	operator-, 142
_enthalpy_3, 133	operator/, 142
_enthalpy_4, 133	operator=, 142
_enthalpy_5, 133	rowExtend, 142
_enthalpy_6, 133	rowExtract, 142
_entropy_1, 133	rowReplace, 143
_entropy_2, 133	rowShrink, 143
_entropy_3, 133	rows, 143
_entropy_4, 133	set_size, 143
_entropy_5, 133	SolnTransform, 143
_entropy_6, 133	sphericalAvg, 143
_gas_conc, 134	sphericalBCFill, 143
_index, 134	sum, 143
_magpie_dat, 134	transpose, 144
_max_capacity, 134	transpose_multiply, 144
_molar_volume, 134	tridiagonalFill, 144
_num_sites, 134	tridiagonalSolve, 144
_temperature, 134	tridiagonalVectorFill, 144
_total_pressure, 134	upperHessenberg2Triangular, 144
computeQpProperties, 132	upperHessenbergSolve, 144
MagpieAdsorbateProperties, 132	upperTriangularSolve, 145
MagpieAdsorbateProperties, 132	zeros, 145
MagpieAdsorbateProperties.h, 249	Matrix < T >, 134
validParams< MagpieAdsorbateProperties >, 250	matrix_too_small
Matrix	error.h, 210
∼Matrix, 138	matvec
adjoint, 138	GMRESR_DATA, 112
cofactor, 138	KMS_DATA, 122
columnExtend, 138	matvec_mis_match
22.3	

error.h, 210	naturalLaplacian3D
matvec_data	Matrix, 141
GMRESR_DATA, 112	negative_mass
KMS_DATA, 122	error.h, 210
max	negative_time
finch.h, 215	error.h, 210
max_iter	nl_bestres
FINCH_DATA, 83	PJFNK_DATA, 159
max_level	nl_iter
KMS_DATA, 122	PJFNK_DATA, 159
max_norm	nl_maxit
SYSTEM_DATA, 179	PJFNK_DATA, 159
maxit	nl_method
BiCGSTAB_DATA, 23	FINCH DATA, 84
CGS DATA, 26	nl_picard
GCR DATA, 106	finch.h, 215
GMRESLP DATA, 109	nl relres
GMRESRP DATA, 115	PJFNK DATA, 159
KMS DATA, 122	nl_res
PCG DATA, 152	PJFNK DATA, 160
PICARD DATA, 155	nl_res_base
min	PJFNK_DATA, 160
finch.h, 215	nl_tol_abs
minmod	PJFNK DATA, 160
finch.h, 215	nl_tol_rel
minmod discretization	PJFNK DATA, 160
finch.h, 215	no diffusion
missing information	error.h, 210
error.h, 210	non_real_edge
molecular diffusion	error.h, 210
PURE GAS, 161	non_square_matrix
molecular_weight	error.h, 209
PURE GAS, 162	NonLinear
molefraction	SCOPSOWL DATA, 167
MIXED GAS, 147	SKUA DATA, 174
molefractionCheck	<del>-</del> ·
skua.h, 260	norm Matrix 141
	Matrix, 141 normFkp1
mspd_dat MAGPIE_DATA, 128	•
	BACKTRACK_DATA, 15
Mu	NormTrack
egret.h, 206	FINCH_DATA, 84
N	not_a_token
GMRESR DATA, 113	error.h, 210 Nu
MIXED_GAS, 147	
SYSTEM DATA, 179	egret.h, 206
NE	nullptr_error
FINCH DATA, 83	error.h, 210
NI	nullptr_func
FINCH DATA, 84	error.h, 210
NL_Output	num_cols
PJFNK DATA, 159	Matrix, 145
NUM_JAC_DATA, 149	num_rows
dxj, 150	Matrix, 145
eps, 150	NumericalJacobian
Fx, 150	lark.h, 233
	OE
Fxp, 150 Na	
	FINCH_DATA, 84
magpie.h, 241	OI

FINCH_DATA, 84	lark.h, 226
OPTRANS_DATA, 150	PCG_DATA, 151
Ai, 150	alpha, 152
li, 151	Ap, 152
omega	bestres, 152
BiCGSTAB_DATA, 23	bestx, 152
omega_old	beta, 152
BiCGSTAB_DATA, 23	iter, 152
operator*	maxit, 152
Matrix, 142	Output, 152
operator()	p, 153
Matrix, 141	r, 153
operator+	r_old, 153
Matrix, 142	relres, 153
operator-	relres_base, 153
Matrix, 142	res, 153
operator/	tol_abs, 153
Matrix, 142	tol_rel, 153
operator=	x, 153
Matrix, 142	z, 153
operatorTranspose	z_old, 153
lark.h, 233	PE3
opt_no_support	egret.h, 206 PI
error.h, 209	
ortho_check_fail	magpie.h, 244
error.h, 210	SYSTEM_DATA, 179
ospre_discretization	PICARD_DATA, 154
finch.h, 216	bestres, 154
out_of_bounds	bestx, 154
error.h, 209	iter, 155
outer_abstol	maxit, 155
KMS_DATA, 122	Output, 155
outer_iter	r, 155
KMS_DATA, 122	relres, 155
outer_reltol	relres_base, 155
KMS_DATA, 122	res, 155
Output	tol_abs, 155
ARNOLDI_DATA, 13	tol_rel, 155
BiCGSTAB_DATA, 23	x0, 155
CGS_DATA, 27	Plo
GCR_DATA, 106	GPAST_DATA, 118
GMRESLP_DATA, 109	PJFNK_DATA, 156
GMRESRP_DATA, 115	backtrack_dat, 157
PCG_DATA, 152	bestx, 157
PICARD DATA, 155	bicgstab_dat, 157
SYSTEM DATA, 179	Bounce, 158
Output_in	cgs_dat, 158
KMS DATA, 123	eps, 158
Output_out	F, 158
KMS DATA, 123	fun_call, 158
OutputFile	funeval, 158
SCOPSOWL DATA, 167	Fv, 158
SKUA DATA, 174	gcr_dat, 158
51.07_57171, 17 <del>-</del>	gmreslp_dat, 158
p	gmresr_dat, 158
BiCGSTAB_DATA, 23	gmresr_dat, 158
CGS_DATA, 27	
PCG DATA, 153	L_Output, 159
PCG	l_iter, 159
. 55	

lin_tol_abs, 159	FINCH_DATA, 84
lin_tol_rel, 159	Po
LineSearch, 159	egret.h, 206
linear_solver, 159	magpie.h, 241
NL_Output, 159	ро
nl_bestres, 159	GPAST_DATA, 118
nl_iter, 159	poi
nl_maxit, 159	GPAST_DATA, 118
nl_relres, 159	pore_diffusion
nl_res, 160	SCOPSOWL_PARAM_DATA, 171
nl_res_base, 160	precon
nl_tol_abs, 160	PJFNK_DATA, 160
nl_tol_rel, 160	precon_data
pcg_dat, 160	PJFNK_DATA, 160
precon, 160	pres
precon_data, 160	FINCH_DATA, 84
res_data, 160	present
v, 160	GPAST_DATA, 118
x, 160	Print2Console
PSI	SCOPSOWL_DATA, 168
egret.h, 206	SKUA_DATA, 175
PT	Print2File
SYSTEM_DATA, 180	SCOPSOWL_DATA, 168
PURE_GAS, 161	SKUA_DATA, 175
density, 161	print2file_SCOPSOWL_header
dynamic_viscosity, 161	scopsowl.h, 254
molecular_diffusion, 161	print2file_SCOPSOWL_result_new
molecular_weight, 162	scopsowl.h, 254
Schmidt, 162	print2file SCOPSOWL result old
specific_heat, 162	scopsowl.h, 254
Sutherland_Const, 162	print2file_SCOPSOWL_time_header
Sutherland_Temp, 162	scopsowl.h, 254
Sutherland Viscosity, 162	print2file_SKUA_header
Par	skua.h, 260
SYSTEM_DATA, 179	print2file SKUA results new
param_dat	skua.h, 260
SCOPSOWL DATA, 167	print2file_SKUA_results_old
SKUA DATA, 174	skua.h, 260
param data	print2file SKUA time header
FINCH_DATA, 84	skua.h, 260
pcg	print2file_dim_header
lark.h, 234	finch.h, 216
pcg_dat	print2file newline
PJFNK DATA, 160	finch.h, 216
pellet density	print2file_result_new
SCOPSOWL_DATA, 167	finch.h, 216
pellet radius	print2file_result_old
SCOPSOWL_DATA, 167	finch.h, 216
SKUA_DATA, 175	print2file_species_header
pi	scopsowl.h, 254
SYSTEM_DATA, 180	skua.h, 260
picard	print2file_tab
lark.h, 235	finch.h, 216
picard_dat	print2file_time_header
	finch.h, 216
FINCH_DATA, 84	Pstd
pjfnk	
lark.h, 235	egret.h, 207
pjfnk_dat	q
	7

GPAST_DATA, 118	ReNum
q_p magpie.h, 245	egret.h, 207 read_error
qAvg	error.h, 210
SCOPSOWL_PARAM_DATA, 171	Recover
qAvg_old	SYSTEM_DATA, 180
SCOPSOWL_PARAM_DATA, 171	ref_diffusion
qIntegralAvg	SCOPSOWL_PARAM_DATA, 172
SCOPSOWL_PARAM_DATA, 171	SKUA_PARAM, 177
qIntegralAvg old	ref_pressure
SCOPSOWL_PARAM_DATA, 171	SCOPSOWL_PARAM_DATA, 172
qT	SKUA_PARAM, 177
magpie.h, 245	ref_temperature
SYSTEM_DATA, 180	SCOPSOWL_PARAM_DATA, 172
qTn	SKUA_PARAM, 177
SKUA_DATA, 175	registerApps
qTnp1	DgospreyApp, 73
SKUA_DATA, 175	registerObjects
qmax	DgospreyApp, 74
GSTA_DATA, 119	relres
qo	BiCGSTAB_DATA, 23
GPAST_DATA, 118	CGS_DATA, 27
magpie.h, 245	GCR DATA, 106
SCOPSOWL PARAM DATA, 171	GMRESLP_DATA, 109
Qst	GMRESRP_DATA, 115
magpie.h, 245	PCG DATA, 153
SCOPSOWL PARAM DATA, 171	PICARD DATA, 155
Qst old	relres_base
_	
SCOPSOWL_PARAM_DATA, 171	BICGSTAB_DATA, 23
QstAvg	CGS_DATA, 27
SCOPSOWL_PARAM_DATA, 171	GCR_DATA, 107
QstAvg_old	GMRESLP_DATA, 109
SCOPSOWL_PARAM_DATA, 171	GMRESRP_DATA, 116
Qstn	PCG_DATA, 153
SKUA_PARAM, 177	PICARD_DATA, 155
Qstnp1	res
SKUA_PARAM, 177	BiCGSTAB_DATA, 23
Qsto	CGS_DATA, 27
SCOPSOWL_PARAM_DATA, 172	FINCH_DATA, 84
В	GCR_DATA, 107
magpie.h, 241	GMRESLP_DATA, 109
01 ,	GMRESRP_DATA, 116
PICCETAD DATA 22	PCG_DATA, 153
BiCGSTAB_DATA, 23 CGS_DATA, 27	PICARD_DATA, 155
	res_data
GCR_DATA, 106 GMRESLP DATA, 109	PJFNK_DATA, 160
_ <i>,</i>	resettime
GMRESRP_DATA, 115	FINCH_DATA, 84
PCG_DATA, 153	restart
PICARD_DATA, 155	GCR_DATA, 107
r0	GMRESLP_DATA, 109
BICGSTAB_DATA, 23	GMRESRP_DATA, 116
CGS_DATA, 27	KMS_DATA, 123
r_old	RetardedTimeDerivative, 162
PCG_DATA, 153	_index, 163
RE3	_retardation, 163
egret.h, 207	computeQpJacobian, 163
RIC	computeQpResidual, 163
FINCH DATA, 85	• • •

RetardedTimeDerivative, 163	level, 167
RetardedTimeDerivative, 163	magpie_dat, 167
RetardedTimeDerivative.h, 250	NonLinear, 167
validParams< RetardedTimeDerivative >, 250	OutputFile, 167
Reynolds	param_dat, 167
MIXED_GAS, 147	pellet_density, 167
rho	pellet_radius, 167
BACKTRACK DATA, 15	Print2Console, 168
BICGSTAB DATA, 24	Print2File, 168
CGS DATA, 27	sim time, 168
rho_old	skua dat, 168
BICGSTAB DATA, 24	SurfDiff, 168
Rn	t, 168
FINCH DATA, 85	t counter, 168
Rnp1	t old, 168
FINCH DATA, 85	<del>-</del> ·
Ro	t_print, 168
	tempy, 168
FINCH_DATA, 85	total_pressure, 168
rowExtend	total_steps, 169
Matrix, 142	user_data, 169
rowExtract	y, 169
Matrix, 142	SCOPSOWL_Executioner
rowReplace	scopsowl.h, 255
Matrix, 143	SCOPSOWL_HPP_
rowShrink	scopsowl.h, 252
Matrix, 143	SCOPSOWL_PARAM_DATA, 169
rows	Adsorbable, 170
Matrix, 143	affinity, 170
Rstd	qAvg, 171
egret.h, 207	qo, 171
rxn_rate_error	Qst, 171
error.h, 210	QstAvg, 171
····, _ · ·	Qsto, 172
S	speciesName, 172
BiCGSTAB DATA, 24	SCOPSOWL postprocesses
FINCH DATA, 85	scopsowl.h, 255
mSPD DATA, 149	SCOPSOWL preprocesses
SCOPSOWL	<del>_</del> .
scopsowl.h, 254	scopsowl.h, 255
SCOPSOWL DATA, 163	SCOPSOWL_reset
binder_fraction, 165	scopsowl.h, 255
binder poresize, 165	SKUA
binder porosity, 165	skua.h, 261
char macro, 165	SKUA_DATA, 172
char_micro, 166	char_measure, 174
	coord, 174
coord_macro, 166	DirichletBC, 174
coord_micro, 166	eval_diff, 174
crystal_radius, 166	eval_kf, 174
DirichletBC, 166	finch_dat, 174
eval_ads, 166	gas_dat, 174
eval_diff, 166	gas_velocity, 174
eval_kf, 166	magpie_dat, 174
eval_retard, 166	NonLinear, 174
eval_surfDiff, 166	OutputFile, 174
finch_dat, 166	param_dat, 174
gas_dat, 167	pellet radius, 175
gas_temperature, 167	Print2Console, 175
gas_velocity, 167	Print2File, 175
Heterogeneous, 167	1 1111(Z1 115, 175
<b>-</b>	

-T- 475			
qTn, 175	error.h, 209		
qTnp1, 175	Schmidt		
sim_time, 175	PURE_GAS, 162		
t, 175	scopsowl.h, 250		
t_counter, 175	avgDp, 252		
t_old, 175	const_filmMassTransfer, 252		
t_print, 175	const_pore_diffusion, 253		
total_steps, 175	default_adsorption, 253		
user_data, 176	default_effective_diffusion, 253 default_filmMassTransfer, 253		
y, 176 SKUA Executioner			
skua.h, 262	default_pore_diffusion, 253 default_retardation, 254		
SKUA_HPP_	default_surf_diffusion, 254		
skua.h, 259	Dk, 252		
SKUA PARAM, 176	Dp, 252		
activation_energy, 177	print2file_SCOPSOWL_header, 254		
Adsorbable, 177	print2file_SCOPSOWL_result_new, 254		
affinity, 177	print2file_SCOPSOWL_result_old, 254		
film transfer, 177	print2file SCOPSOWL time header, 254		
Qstn, 177	print2file_species_header, 254		
Qstnp1, 177	SCOPSOWL, 254		
ref_diffusion, 177	SCOPSOWL_Executioner, 255		
ref pressure, 177	SCOPSOWL_HPP_, 252		
ref_temperature, 177	SCOPSOWL_postprocesses, 255		
speciesName, 177	SCOPSOWL_preprocesses, 255		
xIC, 177	SCOPSOWL_reset, 255		
xn, 177	set_SCOPSOWL_ICs, 255		
xnp1, 177	set_SCOPSOWL_params, 256		
y_eff, 177	set_SCOPSOWL_timestep, 256		
SKUA_postprocesses	setup_SCOPSOWL_DATA, 256		
skua.h, 262	set_SCOPSOWL_ICs		
SKUA_preprocesses	scopsowl.h, 255		
skua.h, 262	set_SCOPSOWL_params		
SKUA reset	scopsowl.h, 256		
skua.h, 262	set_SCOPSOWL_timestep		
SYSTEM_DATA, 178	scopsowl.h, 256		
As, 179	set SKUA ICs		
avg_norm, 179	skua.h, 260		
Carrier, 179	set SKUA params		
I, 179	skua.h, 261		
Ideal, 179	set_SKUA_timestep		
J, 179	skua.h, 261		
K, 179	set_size		
max_norm, 179	Matrix, 143		
N, 179	set_variables		
Output, 179	egret.h, 207		
PI, 179	setbcs		
PT, 180	FINCH_DATA, 85		
Par, 179	setic		
pi, 180	FINCH_DATA, 85		
qT, 180	setparams		
Recover, 180	FINCH_DATA, 85		
Sys, 180	setpostprocess		
T, 180	FINCH_DATA, 85		
total_eval, 180	setpreprocess		
ScNum	FINCH_DATA, 85		
egret.h, 207	settime		
scenario_fail	FINCH_DATA, 85		

setup_FINCH_DATA	speciesName
finch.h, 216	SCOPSOWL_PARAM_DATA, 172
setup_SCOPSOWL_DATA	SKUA_PARAM, 177
scopsowl.h, 256	specific_heat
setup_SKUA_DATA	PURE GAS, 162
skua.h, <mark>261</mark>	Spherical
shapeFactor	finch.h, 213
magpie.h, 241	sphericalAvg
	Matrix, 143
sigma CGS_DATA, 27	
	sphericalBCFill
sim_time	Matrix, 143
SCOPSOWL_DATA, 168	SteadyState
SKUA_DATA, 175	FINCH_DATA, 86
simple_darken_Dc	steps
skua.h, <mark>261</mark>	GMRESLP_DATA, 109
simulation_fail	string_parse_error
error.h, 209	error.h, 210
singular matrix	sum
error.h, 210	ARNOLDI DATA, 13
skua.h, 257	GMRESRP_DATA, 116
const Dc, 259	Matrix, 143
	•
const_kf, 259	SurfDiff
D_c, 259	SCOPSOWL_DATA, 168
D_inf, 259	Sutherland_Const
D_o, 259	PURE_GAS, 162
default_Dc, 259	Sutherland_Temp
default_kf, 260	PURE_GAS, 162
empirical_kf, 260	Sutherland_Viscosity
molefractionCheck, 260	PURE GAS, 162
print2file_SKUA_header, 260	Sys
print2file_SKUA_results_new, 260	SYSTEM DATA, 180
print2file_SKUA_results_old, 260	sys dat
print2file SKUA time header, 260	MAGPIE DATA, 128
print2file species header, 260	Witten IE_Bittin, 120
· ·	Т
SKUA, 261	FINCH_DATA, 86
SKUA_Executioner, 262	SYSTEM_DATA, 180
SKUA_HPP_, 259	t
SKUA_postprocesses, 262	BICGSTAB DATA, 24
SKUA_preprocesses, 262	
SKUA_reset, 262	FINCH_DATA, 86
set_SKUA_ICs, 260	SCOPSOWL_DATA, 168
set_SKUA_params, 261	SKUA_DATA, 175
set_SKUA_timestep, 261	t_counter
setup SKUA DATA, 261	SCOPSOWL_DATA, 168
simple_darken_Dc, 261	SKUA_DATA, 175
theoretical darken Dc, 262	t_old
skua_dat	FINCH DATA, 86
	SCOPSOWL DATA, 168
SCOPSOWL_DATA, 168	SKUA DATA, 175
Sn	t_print
FINCH_DATA, 86	
Snp1	SCOPSOWL_DATA, 168
FINCH_DATA, 86	SKUA_DATA, 175
SolnTransform	tempy
Matrix, 143	SCOPSOWL_DATA, 168
solve	tensor_out_of_bounds
FINCH DATA, 86	error.h, 210
species_dat	term_precon
MIXED GAS, 147	GMRESR_DATA, 113
WINED_GAO, 14/	KMS DATA. 123

terminal_precon	TotalPressureIC.h, 263
GMRESR_DATA, 113	validParams< TotalPressureIC >, 263
KMS_DATA, 123	transpose
theoretical_darken_Dc	Matrix, 144
skua.h, 262	transpose_dat
tol abs	GCR_DATA, 107
BiCGSTAB_DATA, 24	transpose_multiply
CGS_DATA, 27	Matrix, 144
FINCH_DATA, 86	tridiagonalFill
GCR_DATA, 107	Matrix, 144
GMRESLP_DATA, 110	tridiagonalSolve
GMRESRP_DATA, 116	Matrix, 144
PCG_DATA, 153	tridiagonalVectorFill
PICARD_DATA, 155	Matrix, 144
tol rel	man, TT
BICGSTAB_DATA, 24	u
CGS_DATA, 27	CGS_DATA, 28
FINCH DATA, 86	GCR_DATA, 107
GCR_DATA, 107	u_star
	FINCH DATA, 86
GMRESLP_DATA, 110	u_temp
GMRESRP_DATA, 116	GCR_DATA, 107
PCG_DATA, 153	uAverage
PICARD_DATA, 155	finch.h, 217
total_density	uAvg
MIXED_GAS, 147	FINCH_DATA, 87
total_dyn_vis	uAvg_old
MIXED_GAS, 147	FINCH_DATA, 87
total_eval	ulC
SYSTEM_DATA, 180	
total_iter	FINCH_DATA, 87
FINCH_DATA, 86	uT
GCR_DATA, 107	FINCH_DATA, 87
GMRESR_DATA, 113	uT_old
KMS_DATA, 123	FINCH_DATA, 87
total_molecular_weight	uTotal
MIXED_GAS, 147	finch.h, 217
total_pressure	ubest
MIXED_GAS, 148	FINCH_DATA, 87
SCOPSOWL_DATA, 168	un
total_specific_heat	FINCH_DATA, 87
MIXED_GAS, 148	unm1
total_steps	FINCH_DATA, 87
SCOPSOWL DATA, 169	unp1
SKUA_DATA, 175	FINCH_DATA, 87
TotalColumnPressure, 180	unregistered_name
_gas_conc, 181	error.h, 210
index, 181	unstable_matrix
_temperature, 181	error.h, 210
computeValue, 181	uo
TotalColumnPressure, 181	FINCH_DATA, 87
TotalColumnPressure, 181	Update
TotalColumnPressure.h, 262	FINCH DATA, 87
validParams< TotalColumnPressure >, 263	update_arnoldi_solution
TotalPressureIC, 182	lark.h, 236
_PT_IC, 182	upperHessenberg2Triangular
TotalPressureIC, 182	Matrix, 144
TotalPressureIC, 182	upperHessenbergSolve
	Matrix, 144
value, 182	upperTriangularSolve

Matrix, 145	DGFluxBC.h, 198	
user_data	validParams< DGFluxLimitedBC >	
SCOPSOWL_DATA, 169	DGFluxLimitedBC.h, 199	
SKUA_DATA, 176	validParams< DGHeatFluxBC >	
uz_I_E	DGHeatFluxBC.h, 200	
FINCH_DATA, 88	validParams< DGHeatFluxLimitedBC >	
uz_l_l	DGHeatFluxLimitedBC.h, 200	
FINCH_DATA, 88	validParams< DGMassFluxBC >	
uz_lm1_E	DGMassFluxBC.h, 201	
FINCH_DATA, 88	validParams< DGMassFluxLimitedBC >	
uz_lm1_l	DGMassFluxLimitedBC.h, 202	
FINCH_DATA, 88	validParams< DgospreyApp >	
uz_lp1_E	DgospreyApp.h, 203	
FINCH_DATA, 88	validParams< Dispersion >	
uz_lp1_l	Dispersion.h, 204	
FINCH_DATA, 88	validParams< FlowProperties >	
TINOTI_DATA, 00	FlowProperties.h, 219	
V	•	
magpie.h, 241	validParams< FluidHeatTransfer >	
v	FluidHeatTransfer.h, 219	
ARNOLDI_DATA, 13	validParams< GAdvection >	
BiCGSTAB DATA, 24	GAdvection.h, 219	
CGS DATA, 28	validParams< GAnisotropicDiffusion >	
GMRESRP DATA, 116	GAnisotropicDiffusion.h, 220	
mSPD DATA, 149	${\it validParams}{<}~{\it GColumnHeatAdvection}>$	
_ '	GColumnHeatAdvection.h, 220	
PJFNK_DATA, 160	${\sf validParams}{<}~{\sf GColumnHeatDispersion}>$	
VIC	GColumnHeatDispersion.h, 220	
FINCH_DATA, 88	${\sf validParams}{<}{\sf GColumnMassAdvection}>$	
validParams< AdsorbentProperties >	GColumnMassAdvection.h, 221	
AdsorbentProperties.h, 186	${\sf validParams}{<}{\sf GColumnMassDispersion}>$	
validParams< BedHeatAccumulation >	GColumnMassDispersion.h, 221	
BedHeatAccumulation.h, 187	validParams< HeatConductivity >	
validParams< BedProperties >	HeatConductivity.h, 221	
BedProperties.h, 187	validParams< LinearDrivingForce >	
validParams< BedWallHeatTransfer >	LinearDrivingForce.h, 237	
BedWallHeatTransfer.h, 187	validParams< MAGPIE_Adsorption >	
validParams< ColumnTemperatureIC >	MAGPIE_Adsorption.h, 246	
ColumnTemperatureIC.h, 188	umnTemperatureIC.h, 188 validParams < MAGPIE AdsorptionHeat >	
validParams< ConcentrationIC >	MAGPIE_AdsorptionHeat.h, 247	
ConcentrationIC.h, 188	validParams< MAGPIE HeatAccumulation >	
validParams< DGAdvection >	MAGPIE_HeatAccumulation.h, 248	
DGAdvection.h, 189	validParams< MAGPIE Perturbation >	
validParams < DGAnisotropicDiffusion >	MAGPIE Perturbation.h, 249	
DGAnisotropicDiffusion.h, 190	validParams < MagpieAdsorbateProperties >	
validParams < DGColumnHeatAdvection >	MagpieAdsorbateProperties.h, 250	
DGColumnHeatAdvection.h, 191	validParams< RetardedTimeDerivative >	
validParams < DGColumnHeatDispersion >	RetardedTimeDerivative.h, 250	
DGColumnHeatDispersion.h, 192	validParams< TotalColumnPressure >	
validParams< DGColumnMassAdvection >	TotalColumnPressure.h, 263	
DGColumnMassAdvection.h, 193	validParams< TotalPressureIC >	
validParams< DGColumnMassDispersion >		
DGColumnMassDispersion.h, 195	TotalPressureIC.h, 263	
validParams< DGColumnWallHeatFluxBC >	validParams< WallAmbientHeatTransfer >	
DGColumnWallHeatFluxBC.h, 195	WallAmbientHeatTransfer.h, 264	
validParams< DGColumnWallHeatFluxLimitedBC >	validParams< WallHeatAccumulation >	
DGColumnWallHeatFluxLimitedBC.h, 196	WallHeatAccumulation.h, 264	
validParams < DGConcentrationIC >	value	
DGConcentrationIC.h, 197	ColumnTemperatureIC, 29	
validParams< DGFluxBC >	ConcentrationIC, 29	
VALIGI GIGHIO > DOLLIUNDO /		

	DGConcentrationIC, 52	xn	
	TotalPressureIC, 182		SKUA_PARAM, 177
van <i>F</i>	Albada_discretization	xnp1	
	finch.h, 217		SKUA_PARAM, 177
vecto	or_out_of_bounds		
	error.h, 210	у	
velo	city		BiCGSTAB_DATA, 24
	MIXED_GAS, 148		GMRESRP_DATA, 116
Vk			GPAST_DATA, 118
	ARNOLDI DATA, 13		SCOPSOWL_DATA, 169
	GMRESRP_DATA, 116		SKUA_DATA, 176
vn	_ ,	y_ef	f
	FINCH DATA, 88		SKUA_PARAM, 177
vnp1	<del>-</del>	yk	
٠٦٠	FINCH DATA, 88	•	ARNOLDI DATA, 13
vo	1 11011_57171,00		_ ,
••	FINCH DATA, 88	Z	
	TINOTI_DATA, 00		magpie.h, 241
w		Z	
••	ARNOLDI DATA, 13		BiCGSTAB DATA, 24
	CGS DATA, 28		CGS DATA, 28
	GMRESRP DATA, 116		PCG DATA, 153
Mall	AmbientHeatTransfer, 183	z ole	_ ′
vvaii	,		PCG DATA, 153
	_ambient_temp, 183	70r0	_vector
	_inner_dia, 183	2610	error.h, 210
	_outer_dia, 183	70r0	
	_wall_exterior_transfer_coeff, 183	zero	
	computeQpJacobian, 183	71.	Matrix, 145
	computeQpResidual, 183	Zk	CMPEODD DATA 447
	WallAmbientHeatTransfer, 183		GMRESRP_DATA, 117
	WallAmbientHeatTransfer, 183		
Wall	AmbientHeatTransfer.h, 264		
	validParams< WallAmbientHeatTransfer >, 264		
Wall	HeatAccumulation, 184		
	_wall_density, 184		
	_wall_heat_capacity, 184		
	computeQpJacobian, 184		
	computeQpResidual, 184		
	WallHeatAccumulation, 184		
	WallHeatAccumulation, 184		
Wall	HeatAccumulation.h, 264		
	validParams< WallHeatAccumulation >, 264		
Χ			
	BiCGSTAB_DATA, 24		
	CGS_DATA, 28		
	GCR DATA, 107		
	GMRESLP DATA, 110		
	GMRESRP DATA, 116		
	GPAST DATA, 118		
	PCG DATA, 153		
	PJFNK DATA, 160		
x0			
-	PICARD DATA, 155		
xIC			
۸۱٥	SCOPSOWL PARAM DATA, 172		
	SKUA PARAM, 177		
xk	ONON_I AITAWI, ITT		
ΛI	BACKTRACK_DATA, 15		
	Briting City Driving 10		