

VVE

1.0

Generated by Doxygen 1.6.3

Fri May 31 15:37:49 2013

Contents

1	VVE documentation	1
2	Creating libraries for VVE	3
3	VVE examples	5
4	VVE model helpers	7
4.1	Table of content	8
4.2	Basics of model helpers	8
4.2.1	What is a model helper?	8
4.2.2	Commands and methods found in all helpers	8
4.3	Existing model helpers	9
4.3.1	Cell-System model helper	9
4.3.2	Growing tissue model helper	9
5	Installing VVE	11
5.1	Linux	12
5.2	Windows	13
5.2.1	From binary distribution	13
5.2.2	From source	14
5.3	Post-installation specific to L-Studio ATE	14
6	VVE introduction	17
6.1	Table of content	18
6.2	Structure of a model	18
6.3	Model compilation	20
6.3.1	Table of content	20

6.3.2	General intructions	20
6.3.3	Adding dependencies	21
6.3.4	Changing the model name	21
6.3.5	Changing the compilation options	21
6.4	VVE constructs	22
6.4.1	The forall loop	22
6.4.2	Defining your own graph	24
6.4.3	Lookup methods	24
6.4.3.1	Vertex lookup methods	25
6.4.3.2	Neighbors lookup methods	25
6.4.3.3	Edge lookup methods	26
6.4.4	Edition methods	26
6.4.4.1	Vertex edition methods	27
6.4.4.2	Neighbors edition methods	27
6.5	Advanced user interaction with a Viewer	28
6.5.1	Creating a context menu	28
6.5.2	Selecting a 3D object	30
7	VVE libraries	35
8	Deprecated List	37
9	Bug List	39
10	Directory Hierarchy	41
10.1	Directories	41
11	Namespace Index	43
11.1	Namespace List	43
12	Class Index	45
12.1	Class Hierarchy	45
13	Class Index	49
13.1	Class List	49
14	File Index	53

14.1 File List	53
15 Directory Documentation	57
15.1 vvelib/algorithms/ Directory Reference	57
15.2 vvelib/storage/config/ Directory Reference	59
15.3 doc/ Directory Reference	60
15.4 doc/examples/ Directory Reference	61
15.5 vvelib/factory/ Directory Reference	62
15.6 vvelib/geometry/ Directory Reference	63
15.7 vvelib/graph/ Directory Reference	64
15.8 vvelib/shape/ Directory Reference	65
15.9 vvelib/storage/ Directory Reference	66
15.10 vvelib/test/ Directory Reference	67
15.11 vvelib/util/ Directory Reference	68
15.12 vvelib/ Directory Reference	71
16 Namespace Documentation	73
16.1 algorithms Namespace Reference	73
16.1.1 Detailed Description	75
16.1.2 Function Documentation	75
16.1.2.1 drawSymetricVVGraph	75
16.1.2.2 drawVVBIGraphConnections1	76
16.1.2.3 drawVVBIGraphConnections2	76
16.1.2.4 drawVVGraphConnections	76
16.1.2.5 insertAfter	77
16.1.2.6 insertBefore	77
16.1.2.7 shortest_paths_FloydWarshall	77
16.1.2.8 split	78
16.2 bspline_tissue_model Namespace Reference	79
16.2.1 Detailed Description	79
16.2.2 Variable Documentation	79
16.2.2.1 epsilon	79
16.3 cell_system Namespace Reference	80
16.3.1 Detailed Description	81
16.3.2 Typedef Documentation	81

16.3.2.1	label_t	81
16.3.3	Function Documentation	81
16.3.3.1	findDivisionPoints	81
16.3.3.2	findDivisionPoints	83
16.3.3.3	removeWhitespace	84
16.3.3.4	volumeLeftCell	84
16.4	complex_factory Namespace Reference	86
16.4.1	Detailed Description	87
16.4.2	Function Documentation	87
16.4.2.1	connectJunction	87
16.4.2.2	hex_grid	88
16.4.2.3	objreader	91
16.4.2.4	objreader	91
16.4.2.5	objreader	92
16.4.2.6	square_grid	98
16.4.2.7	squareFiller	101
16.5	factory Namespace Reference	102
16.5.1	Detailed Description	102
16.5.2	Function Documentation	102
16.5.2.1	hex_grid	102
16.5.2.2	square_grid	106
16.6	geometry Namespace Reference	111
16.6.1	Detailed Description	113
16.6.2	Typedef Documentation	113
16.6.2.1	Matrix2d	113
16.6.2.2	Matrix3d	113
16.6.2.3	Matrix4d	113
16.6.2.4	Point2d	114
16.6.2.5	Point3d	114
16.6.2.6	Point4d	114
16.6.3	Function Documentation	114
16.6.3.1	barycentricToCartesian	114
16.6.3.2	barycentricToCartesian_matrix	114
16.6.3.3	cartesianToBarycentric	114

16.6.3.4	cartesianToBarycentric_matrix	115
16.6.3.5	centroid	115
16.6.3.6	lineLineIntersection	115
16.6.3.7	lineSegmentIntersection	115
16.6.3.8	lineTriangleIntersection	116
16.6.3.9	planeLineIntersection	116
16.6.3.10	pointInPolygon	117
16.6.3.11	pointInTriangle	117
16.6.3.12	pointInTriangle	117
16.6.3.13	polygonArea	117
16.6.3.14	projectPointOnLine	118
16.6.3.15	projectPointOnPlane	119
16.6.3.16	projectPointOnTriangle	119
16.6.3.17	segmentSegmentIntersection	119
16.6.3.18	triangleArea	120
16.7	graph Namespace Reference	121
16.7.1	Detailed Description	122
16.7.2	Typedef Documentation	122
16.7.2.1	edge_identity_t	122
16.7.2.2	vertex_identity_t	122
16.7.3	Function Documentation	123
16.7.3.1	copy_symmetric	123
16.7.4	Variable Documentation	123
16.7.4.1	vertex_counter	123
16.8	parallel Namespace Reference	124
16.8.1	Detailed Description	125
16.8.2	Function Documentation	125
16.8.2.1	threadEval	125
16.9	shape Namespace Reference	126
16.9.1	Detailed Description	126
16.9.2	Function Documentation	126
16.9.2.1	cube	126
16.9.2.2	cylinder	127
16.9.2.3	disk	127

16.9.2.4	parallelepiped	127
16.9.2.5	sphere	128
16.10	solver Namespace Reference	129
16.10.1	Detailed Description	129
16.10.2	General information	129
16.10.3	Module Usage	130
16.10.4	Enumeration Type Documentation	131
16.10.4.1	SolvingMethod	131
16.10.4.2	ToleranceType	132
16.11	std Namespace Reference	133
16.11.1	Detailed Description	133
16.11.2	Function Documentation	133
16.11.2.1	swap	133
16.12	storage Namespace Reference	134
16.12.1	Detailed Description	136
16.12.2	Principle	136
16.12.3	Defining the serialization	136
16.12.3.1	File versionning	137
16.12.4	Using the VVEStorage object	137
16.12.5	Enumeration Type Documentation	138
16.12.5.1	Options	138
16.12.5.2	STORAGE_ERROR	139
16.12.5.3	TypeCheckingOption	139
16.12.6	Function Documentation	140
16.12.6.1	convert_type	140
16.12.6.2	find_type	140
16.12.6.3	isStrictConversion	140
16.12.6.4	serialization	141
16.12.6.5	typeid_to_name	141
16.12.6.6	typename_to_id	141
16.12.6.7	versionNumber	141
16.13	template_utils Namespace Reference	142
16.13.1	Detailed Description	142
16.14	test Namespace Reference	143

16.14.1 Detailed Description	143
16.15tissue Namespace Reference	144
16.15.1 Detailed Description	145
16.15.2 Typedef Documentation	145
16.15.2.1 cell_graph	145
16.15.2.2 tissue_graph	145
16.15.2.3 vertex	146
16.15.3 Enumeration Type Documentation	146
16.15.3.1 CELL_DIVISION_ALGORITHM	146
16.15.4 Function Documentation	146
16.15.4.1 cellPinching	146
16.15.4.2 findDivisionPoints	148
16.15.4.3 findDivisionPoints	149
16.15.4.4 findDivisionPoints	150
16.16tissue_model Namespace Reference	153
16.16.1 Detailed Description	153
16.16.2 Variable Documentation	153
16.16.2.1 epsilon	153
16.17util Namespace Reference	154
16.17.1 Detailed Description	159
16.17.2 Function Documentation	159
16.17.2.1 absoluteDir	159
16.17.2.2 approx	160
16.17.2.3 clamp	160
16.17.2.4 convertHSVtoRGB	160
16.17.2.5 demangle	161
16.17.2.6 gaussRan	161
16.17.2.7 gaussRan	162
16.17.2.8 greater	162
16.17.2.9 less	162
16.17.2.10make_range	163
16.17.2.11make_range	163
16.17.2.12maximum	163
16.17.2.13minimum	164

16.17.2.14	<code>notequal</code>	164
16.17.2.15	<code>operator*</code>	164
16.17.2.16	<code>operator*</code>	165
16.17.2.17	<code>operator<<</code>	165
16.17.2.18	<code>operator>></code>	165
16.17.2.19	<code>qdemangle</code>	166
16.17.2.20	<code>ran</code>	166
16.17.2.21	<code>ran</code>	166
16.17.2.22	<code>random</code>	167
16.17.2.23	<code>random</code>	167
16.17.2.24	<code>random</code>	167
16.17.2.25	<code>range_c</code>	167
16.17.2.26	<code>range_closed</code>	168
16.17.2.27	<code>range_open</code>	168
16.17.2.28	<code>ran</code>	168
16.17.2.29	<code>ran_time</code>	169
16.17.2.30	<code>ie</code>	169
16.18	<code>util::Shapes</code> Namespace Reference	170
16.18.1	Detailed Description	170
16.19	<code>vvcomplex</code> Namespace Reference	171
16.19.1	Detailed Description	173
16.19.2	General Information	173
16.19.3	2D Cell Complex Usage	174
16.19.3.1	Definition of a 2D Cell Complex	174
16.19.3.2	Creation and Edition of a 2D Cell Complex	175
16.19.3.3	Requirements on the Model using 2D Cell Complexes	176
16.19.3.4	Implementing a Cell Division Algorithm	177
16.19.4	Extending the Cell Complex Class	177
16.19.5	Example	177
16.19.6	Typedef Documentation	180
16.19.6.1	<code>cell</code>	180
16.19.6.2	<code>cell_edge</code>	180
16.19.6.3	<code>cell_graph</code>	180
16.19.6.4	<code>cell_junction_edge</code>	180

16.19.6.5	complex_graph	180
16.19.6.6	const_cell_edge	181
16.19.6.7	const_cell_junction_edge	181
16.19.6.8	const_junction_cell_edge	181
16.19.6.9	const_wall	181
16.19.6.10	junction	181
16.19.6.11	junction_cell_edge	181
16.19.6.12	wall	181
16.19.6.13	wall_graph	181
16.19.7	Function Documentation	182
16.19.7.1	FindCenter	182
16.19.7.2	findCenter	182
16.19.7.3	findDivisionPoints	183
16.19.7.4	FindOppositeWall	184
16.19.7.5	FindWallMin	185
16.19.7.6	testDivisionOnVertices	185
17	Class Documentation	187
17.1	graph::_EmptyEdgeContent Class Reference	187
17.1.1	Detailed Description	187
17.2	graph::Arc< EdgeContent > Struct Template Reference	188
17.2.1	Detailed Description	189
17.2.2	Member Typedef Documentation	189
17.2.2.1	identity_t	189
17.2.3	Constructor & Destructor Documentation	190
17.2.3.1	Arc	190
17.2.3.2	Arc	190
17.2.3.3	Arc	190
17.2.3.4	~Arc	190
17.2.4	Member Function Documentation	191
17.2.4.1	inv	191
17.2.4.2	isNull	191
17.2.4.3	operator bool	191
17.2.4.4	operator edge_t	191

17.2.4.5	operator*	192
17.2.4.6	operator-	192
17.2.4.7	operator->	192
17.2.4.8	source	192
17.2.4.9	sync	193
17.2.4.10	target	193
17.3	util::Shapes::BSplineSurface Class Reference	194
17.3.1	Detailed Description	196
17.3.2	Member Function Documentation	196
17.3.2.1	BoundingBox	196
17.3.2.2	Draw	196
17.3.2.3	IsTextured	200
17.3.2.4	Load	200
17.3.2.5	Recompute	201
17.3.2.6	Reread	201
17.3.2.7	TextureId	201
17.3.2.8	Transform	202
17.4	util::Buffer< T, size > Class Template Reference	203
17.4.1	Detailed Description	203
17.4.2	Constructor & Destructor Documentation	203
17.4.2.1	Buffer	203
17.4.2.2	~Buffer	204
17.4.3	Member Function Documentation	204
17.4.3.1	c_data	204
17.4.3.2	operator[]	204
17.5	algorithms::TriangleSurface::Cell Struct Reference	205
17.5.1	Detailed Description	205
17.5.2	Member Data Documentation	205
17.5.2.1	id	205
17.5.2.2	normal	205
17.6	tissue::CellPinchingParams Struct Reference	206
17.6.1	Detailed Description	206
17.6.2	Constructor & Destructor Documentation	207
17.6.2.1	CellPinchingParams	207

17.6.2.2	CellPinchingParams	207
17.6.3	Member Data Documentation	207
17.6.3.1	cellMaxPinch	207
17.6.3.2	cellPinch	207
17.7	cell_system::CellSystem< Complex, MyModel > Class Template Reference	208
17.7.1	Detailed Description	211
17.7.2	Member Typedef Documentation	212
17.7.2.1	division_rules_t	212
17.7.2.2	label_change_rules_t	212
17.7.3	Constructor & Destructor Documentation	212
17.7.3.1	CellSystem	212
17.7.4	Member Function Documentation	212
17.7.4.1	label	212
17.7.4.2	readMechanicParam	213
17.7.4.3	readParam	214
17.7.4.4	readRules	214
17.7.4.5	readSymbols	216
17.7.4.6	registerSymbol	217
17.7.4.7	serialize	217
17.7.4.8	step	218
17.7.4.9	step_cellsystem	218
17.7.4.10	step_cellsystem_division	219
17.7.4.11	step_cellsystem_meca	219
17.7.4.12	updateCellsArea	221
17.7.4.13	version	221
17.7.4.14	versionNumber	222
17.7.5	Member Data Documentation	222
17.7.5.1	current_div	222
17.7.5.2	damping	222
17.7.5.3	division_rules	222
17.7.5.4	dt	223
17.7.5.5	ke	223
17.7.5.6	ki	223

17.7.5.7	label_change_rules	223
17.7.5.8	label_names	224
17.7.5.9	label_numbers	224
17.7.5.10	mass	224
17.7.5.11	pressure	224
17.7.5.12	restLength	224
17.7.5.13	showInterval	225
17.7.5.14	showTime	225
17.7.5.15	stability	225
17.7.5.16	T	225
17.7.5.17	time	226
17.7.5.18	viewRulesApplication	226
17.7.5.19	viewRulesDefinitions	226
17.8	cell_system::CellSystemCell Class Reference	227
17.8.1	Detailed Description	227
17.8.2	Member Function Documentation	227
17.8.2.1	serialize	227
17.8.3	Member Data Documentation	228
17.8.3.1	area	228
17.8.3.2	label	228
17.8.3.3	pos	228
17.9	cell_system::CellSystemDivisionParams Class Reference	229
17.9.1	Detailed Description	230
17.9.2	Constructor & Destructor Documentation	230
17.9.2.1	CellSystemDivisionParams	230
17.9.2.2	CellSystemDivisionParams	230
17.9.3	Member Data Documentation	230
17.9.3.1	angle	230
17.9.3.2	direction	231
17.9.3.3	epsilon	231
17.9.3.4	minCellWall	231
17.9.3.5	pinching	231
17.9.3.6	pinchingParam	231
17.9.3.7	ratio	231

17.10	cell_system::CellSystemJunction Class Reference	233
17.10.1	Detailed Description	233
17.10.2	Member Function Documentation	233
17.10.2.1	serialize	233
17.10.3	Member Data Documentation	234
17.10.3.1	force	234
17.10.3.2	pos	234
17.10.3.3	velocity	234
17.11	util::CircIterator< ForwardIterator > Class Template Reference	235
17.11.1	Detailed Description	236
17.11.2	Constructor & Destructor Documentation	236
17.11.2.1	CircIterator	236
17.11.2.2	CircIterator	236
17.11.2.3	CircIterator	237
17.11.3	Member Function Documentation	237
17.11.3.1	end	237
17.11.3.2	isInitIterator	237
17.12	tissue::ClosestMidAlgoParams Struct Reference	238
17.12.1	Detailed Description	238
17.12.2	Constructor & Destructor Documentation	239
17.12.2.1	ClosestMidAlgoParams	239
17.12.2.2	ClosestMidAlgoParams	239
17.12.3	Member Data Documentation	239
17.12.3.1	cellWallMin	239
17.12.3.2	strictCellWallMin	239
17.13	tissue::ClosestWallAlgoParams Struct Reference	240
17.13.1	Detailed Description	240
17.13.2	Constructor & Destructor Documentation	241
17.13.2.1	ClosestWallAlgoParams	241
17.13.2.2	ClosestWallAlgoParams	241
17.13.3	Member Data Documentation	241
17.13.3.1	cellWallMin	241
17.13.3.2	strictCellWallMin	241
17.14	util::Color< T > Class Template Reference	242

17.14.1 Detailed Description	244
17.14.2 Constructor & Destructor Documentation	244
17.14.2.1 Color	244
17.14.2.2 Color	244
17.14.3 Member Function Documentation	244
17.14.3.1 a	244
17.14.3.2 a	245
17.14.3.3 a	245
17.14.3.4 b	245
17.14.3.5 b	245
17.14.3.6 b	245
17.14.3.7 g	246
17.14.3.8 g	246
17.14.3.9 g	246
17.14.3.10operator=	246
17.14.3.11operator=	247
17.14.3.12r	247
17.14.3.13r	247
17.14.3.14r	247
17.15bspline_tissue_model::TissueModel< RealModel, TissueClass >::CompareSize Struct Reference	248
17.15.1 Detailed Description	248
17.16tissue_model::TissueModel< RealModel, TissueClass >::CompareSize Struct Reference	249
17.16.1 Detailed Description	249
17.17util::Contour Class Reference	250
17.17.1 Detailed Description	251
17.17.2 Constructor & Destructor Documentation	251
17.17.2.1 Contour	251
17.17.2.2 Contour	251
17.17.3 Member Function Documentation	252
17.17.3.1 getMax	252
17.17.3.2 getMin	252
17.17.3.3 length	252

17.17.3.4 normal	253
17.17.3.5 operator()	253
17.17.3.6 operator=	254
17.17.3.7 reread	254
17.17.3.8 tangent	261
17.17.3.9 travel	261
17.18storage::ConvertType< From, To > Struct Template Reference	263
17.18.1 Detailed Description	263
17.18.2 Member Function Documentation	263
17.18.2.1 operator()	263
17.19graph::CountedContent< Content > Struct Template Reference . . .	264
17.19.1 Detailed Description	264
17.19.2 Member Data Documentation	264
17.19.2.1 count	264
17.20util::CrossProductType< 2, T > Struct Template Reference	265
17.20.1 Detailed Description	265
17.21util::CrossProductType< 3, T > Struct Template Reference	266
17.21.1 Detailed Description	266
17.22util::GraphInset::Data Struct Reference	267
17.22.1 Detailed Description	267
17.22.2 Member Data Documentation	267
17.22.2.1 changed	267
17.22.2.2 time	267
17.22.2.3 values	268
17.23vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::division_ result_t Class Reference	269
17.23.1 Detailed Description	269
17.24vvcomplex::DivisionData< JunctionContent > Class Template Refer- ence	270
17.24.1 Detailed Description	271
17.24.2 Member Typedef Documentation	271
17.24.2.1 junction	271
17.24.3 Constructor & Destructor Documentation	271
17.24.3.1 DivisionData	271

17.24.3.2 DivisionData	272
17.24.3.3 DivisionData	272
17.24.4 Member Function Documentation	272
17.24.4.1 operator bool	272
17.24.4.2 operator!	273
17.24.5 Member Data Documentation	273
17.24.5.1 divide_at_u1	273
17.24.5.2 divide_at_v1	273
17.24.5.3 pu	273
17.24.5.4 pv	274
17.24.5.5 u1	274
17.24.5.6 v1	274
17.25cell_system::DivisionParams Class Reference	276
17.25.1 Detailed Description	276
17.25.2 Member Data Documentation	276
17.25.2.1 angle	276
17.25.2.2 left_label	277
17.25.2.3 ratio	277
17.25.2.4 right_label	277
17.26graph::Edge< EdgeContent > Class Template Reference	278
17.26.1 Detailed Description	280
17.26.2 Member Typedef Documentation	280
17.26.2.1 content_t	280
17.26.2.2 identity_t	280
17.26.2.3 pointer	280
17.26.3 Constructor & Destructor Documentation	281
17.26.3.1 Edge	281
17.26.3.2 Edge	281
17.26.3.3 Edge	281
17.26.4 Member Function Documentation	282
17.26.4.1 clear	282
17.26.4.2 isNull	282
17.26.4.3 operator bool	282
17.26.4.4 operator!=	282

17.26.4.5 operator*	283
17.26.4.6 operator->	283
17.26.4.7 operator->*	283
17.26.4.8 operator->*	284
17.26.4.9 operator<	284
17.26.4.10operator<=	284
17.26.4.11operator=	285
17.26.4.12operator==	285
17.26.4.13operator>	285
17.26.4.14operator>=	286
17.26.4.15serialize	286
17.26.4.16source	286
17.26.4.17target	287
17.26.5 Member Data Documentation	287
17.26.5.1 _content	287
17.26.5.2 _source	287
17.26.5.3 _target	287
17.27util::FileObject Class Reference	289
17.27.1 Detailed Description	290
17.27.2 Constructor & Destructor Documentation	290
17.27.2.1 FileObject	290
17.27.2.2 FileObject	290
17.27.2.3 FileObject	291
17.27.2.4 FileObject	291
17.27.3 Member Function Documentation	291
17.27.3.1 getFilename	291
17.27.3.2 modified	291
17.27.3.3 operator=	292
17.27.3.4 reread	292
17.27.3.5 setFilename	292
17.27.3.6 setFilename	292
17.27.3.7 setFilename	293
17.28storage::Frame Struct Reference	294
17.28.1 Detailed Description	294

17.29util::Function Class Reference	295
17.29.1 Detailed Description	296
17.29.2 Constructor & Destructor Documentation	297
17.29.2.1 Function	297
17.29.2.2 Function	297
17.29.3 Member Function Documentation	298
17.29.3.1 getMax	298
17.29.3.2 getMin	298
17.29.3.3 normalizeX	298
17.29.3.4 normalizeY	298
17.29.3.5 operator()	299
17.29.3.6 operator=	300
17.29.3.7 reread	301
17.29.3.8 scaleX	304
17.29.3.9 scaleY	304
17.29.3.10scaleY	304
17.29.3.11scaleY	305
17.29.3.12setSamples	305
17.29.3.13shiftX	305
17.29.3.14shiftX	305
17.29.3.15shiftY	306
17.29.3.16shiftY	306
17.30util::GraphInset Class Reference	307
17.30.1 Detailed Description	311
17.30.2 Member Typedef Documentation	311
17.30.2.1 iterator	311
17.30.2.2 pointer	311
17.30.2.3 reference	311
17.30.2.4 reverse_iterator	311
17.30.2.5 value_type	312
17.30.3 Constructor & Destructor Documentation	312
17.30.3.1 GraphInset	312
17.30.4 Member Function Documentation	312
17.30.4.1 addData	312

17.30.4.2	addData	312
17.30.4.3	addData	313
17.30.4.4	addData	313
17.30.4.5	addData	313
17.30.4.6	addData	314
17.30.4.7	addData	314
17.30.4.8	addData	314
17.30.4.9	check	315
17.30.4.10	cleanData	316
17.30.4.11	clear	316
17.30.4.12	closeGraph	316
17.30.4.13	draw	317
17.30.4.14	drawFrame	318
17.30.4.15	drawWithName	319
17.30.4.16	graphClosed	319
17.30.4.17	numberOfLines	319
17.30.4.18	openGraph	319
17.30.4.19	readParms	320
17.30.4.20	setNumberOfLines	320
17.30.4.21	setTime	321
17.30.4.22	size	321
17.30.4.23	writeCSV	321
17.30.5	Member Data Documentation	322
17.30.5.1	_numberOfLines	322
17.30.5.2	autoOpenClose	322
17.30.5.3	backColor	322
17.30.5.4	backgroundOffset	322
17.30.5.5	changeSourceColor	323
17.30.5.6	changeSourceOffset	323
17.30.5.7	changeSourceWidth	323
17.30.5.8	closed	323
17.30.5.9	contourColor	323
17.30.5.10	contourOffset	323
17.30.5.11	contourWidth	323

17.30.5.12	data	324
17.30.5.13	initialized	324
17.30.5.14	keepAllData	324
17.30.5.15	linesColor	324
17.30.5.16	linesMinMaxView	324
17.30.5.17	linesOffset	324
17.30.5.18	linesWidth	324
17.30.5.19	showAll	325
17.30.5.20	showIndices	325
17.30.5.21	time	325
17.30.5.22	timeSpan	325
17.30.5.23	windowPosition	325
17.30.5.24	windowSize	325
17.31	complex_factory::HexFiller< Complex, Model > Struct Template Reference	327
17.31.1	Detailed Description	327
17.32	vvcomplex::InModelDivisionParam Class Reference	328
17.32.1	Detailed Description	328
17.33	algorithms::Insert< vvgraph, do_checks > Class Template Reference	329
17.33.1	Detailed Description	329
17.34	util::KeyFramer Class Reference	330
17.34.1	Detailed Description	331
17.34.2	Member Function Documentation	331
17.34.2.1	reread	331
17.35	util::Materials::Material Struct Reference	332
17.35.1	Detailed Description	332
17.36	util::Materials Class Reference	333
17.36.1	Detailed Description	334
17.36.2	Constructor & Destructor Documentation	334
17.36.2.1	Materials	334
17.36.3	Member Function Documentation	334
17.36.3.1	blend	334
17.36.3.2	getMaterial	335
17.36.3.3	reread	336

17.36.3.4 useMaterial	337
17.37util::Matrix< nRows, nCols, T > Class Template Reference	339
17.37.1 Detailed Description	344
17.37.2 Constructor & Destructor Documentation	345
17.37.2.1 Matrix	345
17.37.2.2 Matrix	345
17.37.2.3 Matrix	345
17.37.2.4 Matrix	345
17.37.2.5 Matrix	346
17.37.2.6 Matrix	346
17.37.2.7 Matrix	347
17.37.3 Member Function Documentation	347
17.37.3.1 c_data	347
17.37.3.2 data	348
17.37.3.3 identity	348
17.37.3.4 nbColumns	348
17.37.3.5 nbRows	348
17.37.3.6 operator()	349
17.37.3.7 operator()	349
17.37.3.8 operator*	349
17.37.3.9 operator*	349
17.37.3.10operator+	350
17.37.3.11operator-	350
17.37.3.12operator/	350
17.37.3.13operator=	351
17.37.3.14operator[]	351
17.37.3.15operator[]	352
17.37.3.16operator~	352
17.37.3.17rotation	352
17.37.3.18rotation	353
17.37.3.19size	353
17.37.3.20trace	353
17.37.3.21zero	354
17.37.4 Friends And Related Function Documentation	354

17.37.4.1 cofactor	354
17.37.4.2 det	354
17.37.4.3 det	354
17.37.4.4 det	354
17.37.4.5 det	355
17.37.4.6 inverse	355
17.37.4.7 inverse	355
17.37.4.8 inverse	355
17.37.4.9 inverse	355
17.37.4.10map	355
17.37.4.11map	356
17.37.4.12map	356
17.37.4.13map	357
17.37.4.14map	357
17.37.4.15map	357
17.37.4.16map	358
17.37.4.17map	358
17.37.4.18map	359
17.37.4.19map	359
17.37.4.20map	360
17.37.4.21map	360
17.37.4.22norm	360
17.37.4.23normsq	361
17.37.4.24operator*	361
17.37.4.25operator*	361
17.37.4.26transpose	361
17.38Model Class Reference	362
17.38.1 Detailed Description	367
17.38.2 Constructor & Destructor Documentation	367
17.38.2.1 Model	367
17.38.2.2 ~Model	368
17.38.3 Member Function Documentation	368
17.38.3.1 actions	368
17.38.3.2 addAction	368

17.38.3.3	addAction	369
17.38.3.4	addItem	369
17.38.3.5	addItem	369
17.38.3.6	addItem	369
17.38.3.7	animationPeriod	370
17.38.3.8	arguments	370
17.38.3.9	changedExitCode	370
17.38.3.10	draw	371
17.38.3.11	draw	371
17.38.3.12	drawWithNames	371
17.38.3.13	drawWithNames	371
17.38.3.14	fileRegister	372
17.38.3.15	fileUnregister	372
17.38.3.16	finalizeDraw	372
17.38.3.17	finalizeDraw	372
17.38.3.18	finalizePrint	373
17.38.3.19	helpString	373
17.38.3.20	initDraw	373
17.38.3.21	initDraw	374
17.38.3.22	initPrint	374
17.38.3.23	insertAction	374
17.38.3.24	insertActions	375
17.38.3.25	loadingSnapshot	375
17.38.3.26	loadingSnapshot	375
17.38.3.27	loadSnapshot	375
17.38.3.28	loadSnapshot	376
17.38.3.29	menuItemActionInserted	376
17.38.3.30	menuItemActionRemoved	376
17.38.3.31	menuItem	376
17.38.3.32	modifiedFiles	377
17.38.3.33	modifiedFiles	377
17.38.3.34	postDraw	377
17.38.3.35	postDraw	378
17.38.3.36	postSelection	378

17.38.3.37preDraw	378
17.38.3.38preDraw	379
17.38.3.39print	379
17.38.3.40registerFile	379
17.38.3.41removeAction	379
17.38.3.42removeMenuItem	380
17.38.3.43removeMenuItem	380
17.38.3.44removeMenuItem	380
17.38.3.45reread	381
17.38.3.46restart	381
17.38.3.47restartModel	381
17.38.3.48run	381
17.38.3.49runModel	382
17.38.3.50saveNextSnapshot	382
17.38.3.51saveSnapshot	382
17.38.3.52savingNextSnapshot	382
17.38.3.53savingScreenshot	382
17.38.3.54savingSnapshot	383
17.38.3.55screenshot	383
17.38.3.56serialize	383
17.38.3.57setAnimationPeriod	384
17.38.3.58setExitCode	384
17.38.3.59setStatusMessage	384
17.38.3.60statusMessage	384
17.38.3.61step	385
17.38.3.62stop	385
17.38.3.63stopModel	385
17.38.3.64unregisterFile	385
17.38.3.65version	386
17.38.3.66versionNumber	386
17.39graph::VVGrahp< VertexContent, EdgeContent, compact >::neighbor_t Struct Reference	387
17.39.1 Detailed Description	387
17.39.2 Member Typedef Documentation	388

17.39.2.1 edge_list_t	388
17.39.3 Constructor & Destructor Documentation	388
17.39.3.1 neighbor_t	388
17.39.4 Member Function Documentation	388
17.39.4.1 operator==	388
17.39.5 Member Data Documentation	389
17.39.5.1 target	389
17.40complex_factory::ObjReaderError Class Reference	390
17.40.1 Detailed Description	390
17.40.2 Member Enumeration Documentation	391
17.40.2.1 Type	391
17.40.3 Member Function Documentation	391
17.40.3.1 operator bool	391
17.40.3.2 string	391
17.40.3.3 type	391
17.41util::Palette Class Reference	392
17.41.1 Detailed Description	393
17.41.2 Constructor & Destructor Documentation	393
17.41.2.1 Palette	393
17.41.2.2 Palette	394
17.41.3 Member Function Documentation	394
17.41.3.1 blend	394
17.41.3.2 blend	394
17.41.3.3 getColor	395
17.41.3.4 getColor	395
17.41.3.5 reread	396
17.41.3.6 select	396
17.41.3.7 selectColor	397
17.41.3.8 useColor	398
17.42ParallelControler Class Reference	399
17.42.1 Detailed Description	399
17.43util::Parms Class Reference	400
17.43.1 Detailed Description	402
17.43.2 Constructor & Destructor Documentation	403

17.43.2.1 Pargs	403
17.43.3 Member Function Documentation	404
17.43.3.1 all	404
17.43.3.2 all	404
17.43.3.3 all	404
17.43.3.4 all	405
17.43.3.5 all	405
17.43.3.6 all	405
17.43.3.7 all	406
17.43.3.8 all	406
17.43.3.9 all	407
17.43.3.10all	407
17.43.3.11all	408
17.43.3.12all	408
17.43.3.13all	408
17.43.3.14all	408
17.43.3.15all	409
17.43.3.16all	409
17.43.3.17sLoaded	410
17.43.3.18operator()	410
17.43.3.19operator()	411
17.43.3.20operator()	411
17.43.3.21operator()	411
17.43.3.22operator()	412
17.43.3.23operator()	412
17.43.3.24operator()	412
17.43.3.25operator()	413
17.43.3.26verboseLevel	414
17.44util::Point< T > Class Template Reference	415
17.44.1 Detailed Description	416
17.44.2 Constructor & Destructor Documentation	416
17.44.2.1 Point	416
17.44.2.2 ~Point	416
17.44.3 Member Function Documentation	417

17.44.3.1 c_data	417
17.44.3.2 operator*=	417
17.44.3.3 operator-	417
17.44.3.4 operator/	417
17.44.3.5 operator/=	418
17.44.3.6 proj	418
17.44.3.7 proj_length	418
17.45algorithms::TriangleSurface::Position Struct Reference	419
17.45.1 Detailed Description	419
17.46geometry::Quaternion Class Reference	420
17.46.1 Detailed Description	422
17.46.2 Constructor & Destructor Documentation	422
17.46.2.1 Quaternion	422
17.46.2.2 Quaternion	422
17.46.2.3 Quaternion	422
17.46.2.4 Quaternion	423
17.46.2.5 Quaternion	423
17.46.2.6 Quaternion	423
17.46.3 Member Function Documentation	424
17.46.3.1 angle	424
17.46.3.2 axis	424
17.46.3.3 conjugate	425
17.46.3.4 inverse	425
17.46.3.5 inverseRotate	425
17.46.3.6 operator*	425
17.46.3.7 operator*=	426
17.46.3.8 operator*=	426
17.46.3.9 operator+	426
17.46.3.10operator+=	426
17.46.3.11operator/	427
17.46.3.12operator/=	427
17.46.3.13operator=	427
17.46.3.14rotate	427
17.46.3.15setAxisAngle	428

17.46.3.16setMatrix	428
17.46.3.17setMatrix	428
17.46.3.18w	429
17.46.3.19w	429
17.47util::range< Iterator > Class Template Reference	430
17.47.1 Detailed Description	431
17.47.2 Constructor & Destructor Documentation	431
17.47.2.1 range	431
17.47.2.2 range	431
17.47.2.3 range	432
17.47.2.4 range	432
17.47.2.5 range	432
17.47.3 Member Function Documentation	432
17.47.3.1 begin	432
17.47.3.2 cbegin	433
17.47.3.3 cend	433
17.47.3.4 end	433
17.47.3.5 operator=	433
17.47.3.6 operator=	434
17.47.3.7 setBegin	434
17.47.3.8 setEnd	434
17.47.3.9 size	434
17.48util::refpair< T, U > Class Template Reference	435
17.48.1 Detailed Description	435
17.48.2 Constructor & Destructor Documentation	435
17.48.2.1 refpair	435
17.48.2.2 refpair	436
17.48.3 Member Function Documentation	436
17.48.3.1 operator=	436
17.48.4 Member Data Documentation	436
17.48.4.1 first	436
17.48.4.2 second	436
17.49graph::VVGrahp< VertexContent, EdgeContent, compact >::search_ result_t< Neighborhood, Iterator > Struct Template Reference	438

17.49.1 Detailed Description	438
17.49.2 Constructor & Destructor Documentation	439
17.49.2.1 search_result_t	439
17.49.2.2 search_result_t	439
17.49.2.3 search_result_t	439
17.49.3 Member Function Documentation	440
17.49.3.1 operator bool	440
17.49.4 Member Data Documentation	440
17.49.4.1 found	440
17.49.4.2 it	440
17.49.4.3 neighborhood	441
17.50graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator > Struct Template Reference	442
17.50.1 Detailed Description	443
17.50.2 Constructor & Destructor Documentation	443
17.50.2.1 search_result_t	443
17.50.2.2 search_result_t	443
17.50.2.3 search_result_t	444
17.50.3 Member Function Documentation	444
17.50.3.1 operator bool	444
17.50.4 Member Data Documentation	444
17.50.4.1 found	444
17.50.4.2 it	445
17.50.4.3 neighborhood	445
17.51util::SelectMemberIterator< Iterator, T, member, Reference, Pointer > Class Template Reference	447
17.51.1 Detailed Description	449
17.51.2 Member Typedef Documentation	450
17.51.2.1 base_iterator	450
17.51.2.2 difference_type	450
17.51.2.3 iterator_category	450
17.51.2.4 pointer	450
17.51.2.5 reference	450
17.51.2.6 value_type	451

17.51.3 Constructor & Destructor Documentation	451
17.51.3.1 SelectMemberIterator	451
17.51.3.2 SelectMemberIterator	451
17.51.3.3 SelectMemberIterator	451
17.51.4 Member Function Documentation	452
17.51.4.1 base	452
17.51.4.2 operator*	452
17.51.4.3 operator*	452
17.51.4.4 operator++	453
17.51.4.5 operator++	453
17.51.4.6 operator+=	453
17.51.4.7 operator--	453
17.51.4.8 operator--	454
17.51.4.9 operator-=	454
17.51.4.10operator->	454
17.51.4.11operator->	454
17.51.4.12operator=	455
17.51.5 Friends And Related Function Documentation	455
17.51.5.1 operator-	455
17.51.6 Member Data Documentation	455
17.51.6.1 it	455
17.52util::set_vector< T, Equal, Alloc > Struct Template Reference	457
17.52.1 Detailed Description	457
17.53tissue::ShortWallAlgoParams Struct Reference	459
17.53.1 Detailed Description	460
17.53.2 Constructor & Destructor Documentation	460
17.53.2.1 ShortWallAlgoParams	460
17.53.2.2 ShortWallAlgoParams	460
17.53.3 Member Data Documentation	460
17.53.3.1 cellWallMin	460
17.53.3.2 cellWallSample	460
17.53.3.3 dx	461
17.53.3.4 strictCellWallMin	461

17.54	<code>graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::single_neighborhood_t< VertexSrcContent, VertexTgtContent, EdgeSrcContent ></code> Struct Template Reference	462
17.54.1	Detailed Description	463
17.54.2	Member Function Documentation	463
17.54.2.1	<code>operator==</code>	463
17.54.3	Member Data Documentation	464
17.54.3.1	<code>edges</code>	464
17.54.3.2	<code>flagged</code>	464
17.54.3.3	<code>in_edges</code>	464
17.55	<code>graph::VVGraph< VertexContent, EdgeContent, compact >::single_neighborhood_t</code> Struct Reference	465
17.55.1	Detailed Description	465
17.55.2	Member Function Documentation	466
17.55.2.1	<code>operator==</code>	466
17.55.3	Member Data Documentation	466
17.55.3.1	<code>edges</code>	466
17.55.3.2	<code>flagged</code>	466
17.55.3.3	<code>in_edges</code>	466
17.56	<code>solver::Solver< nb_vars, identifier ></code> Class Template Reference	468
17.56.1	Detailed Description	473
17.56.2	Member Typedef Documentation	473
17.56.2.1	<code>EdgeInternals</code>	473
17.56.2.2	<code>Mat</code>	473
17.56.2.3	<code>tag_t</code>	473
17.56.2.4	<code>Vec</code>	474
17.56.2.5	<code>VertexInternals</code>	474
17.56.3	Constructor & Destructor Documentation	474
17.56.3.1	<code>Solver</code>	474
17.56.3.2	<code>Solver</code>	474
17.56.4	Member Function Documentation	475
17.56.4.1	<code>FindPartials</code>	475
17.56.4.2	<code>initialize</code>	476
17.56.4.3	<code>operator()</code>	477
17.56.4.4	<code>readParms</code>	478

17.56.4.5 solveAdaptiveCrankNicholson	480
17.56.4.6 solveAdaptiveEuler	485
17.56.4.7 solveAdaptiveRungeKutta	486
17.56.4.8 solveEuler	489
17.56.4.9 solveFixedpoint	490
17.56.4.10 solveMidpoint	491
17.56.4.11 solveRungeKutta	492
17.56.5 Member Data Documentation	494
17.56.5.1 AEulerHighTol	494
17.56.5.2 AEulerIncDt	494
17.56.5.3 AEulerLowTol	494
17.56.5.4 AEulerResDt	494
17.56.5.5 AEulerResTol	494
17.56.5.6 AEulerTolType	495
17.56.5.7 ARungeHighTol	495
17.56.5.8 ARungeIncDt	495
17.56.5.9 ARungeLowTol	495
17.56.5.10 ARungeResDt	495
17.56.5.11 ARungeResTol	496
17.56.5.12 ARungeTolType	496
17.56.5.13 ConjGradMaxSteps	496
17.56.5.14 ConjGradTol	496
17.56.5.15 ConjGradTolType	496
17.56.5.16 ConstNbPartials	497
17.56.5.17 CRAvgCPU	497
17.56.5.18 CRIncDt	497
17.56.5.19 CRMinCPU	497
17.56.5.20 CRResDt	497
17.56.5.21 current_method	498
17.56.5.22 dt	498
17.56.5.23 Dx	498
17.56.5.24 EulerDt	498
17.56.5.25 FixedPointDt	499
17.56.5.26 FixedPointMaxSteps	499

17.56.5.27FixedPointTol	499
17.56.5.28FixedPointTolType	499
17.56.5.29InitialDt	499
17.56.5.30initialized	500
17.56.5.31MaxDt	500
17.56.5.32method	500
17.56.5.33MidPointDt	500
17.56.5.34MinDt	501
17.56.5.35NewtMaxSteps	501
17.56.5.36NewtTol	501
17.56.5.37NewtTolType	501
17.56.5.38pdt	501
17.56.5.39peff	502
17.56.5.40PrintMatrix	502
17.56.5.41PrintStats	502
17.56.5.42RungeKuttaDt	502
17.56.5.43step	502
17.57complex_factory::SquareFiller< Complex, Model > Struct Template Reference	504
17.57.1 Detailed Description	504
17.58util::Tensor< T > Class Template Reference	505
17.58.1 Detailed Description	505
17.58.2 Constructor & Destructor Documentation	505
17.58.2.1 Tensor	505
17.58.2.2 ~Tensor	506
17.58.3 Member Function Documentation	506
17.58.3.1 angle_theta	506
17.58.3.2 grow	506
17.58.3.3 scale_s	507
17.58.3.4 scale_t	507
17.58.3.5 set	507
17.59util::Texture1D Class Reference	508
17.59.1 Detailed Description	509
17.59.2 Constructor & Destructor Documentation	509

17.59.2.1 Texture1D	509
17.59.2.2 Texture1D	509
17.59.2.3 Texture1D	510
17.59.2.4 ~Texture1D	510
17.59.3 Member Function Documentation	511
17.59.3.1 bind	511
17.59.3.2 blend	511
17.59.3.3 clamp	511
17.59.3.4 decal	511
17.59.3.5 filter	512
17.59.3.6 modulate	512
17.59.3.7 operator=	512
17.59.3.8 replace	513
17.60util::Texture2D Class Reference	514
17.60.1 Detailed Description	515
17.60.2 Constructor & Destructor Documentation	515
17.60.2.1 Texture2D	515
17.60.2.2 Texture2D	515
17.60.2.3 Texture2D	516
17.60.2.4 ~Texture2D	517
17.60.3 Member Function Documentation	517
17.60.3.1 bind	517
17.60.3.2 blend	517
17.60.3.3 clamp	517
17.60.3.4 decal	518
17.60.3.5 filter	518
17.60.3.6 modulate	518
17.60.3.7 operator=	519
17.60.3.8 replace	519
17.61parallel::ThreadEval Class Reference	520
17.61.1 Detailed Description	520
17.62tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, com- pact, LeafClass > Class Template Reference	521

17.62.1 Detailed Description	525
17.62.2 Member Typedef Documentation	525
17.62.2.1 division_data	525
17.62.3 Constructor & Destructor Documentation	526
17.62.3.1 Tissue	526
17.62.3.2 Tissue	526
17.62.3.3 Tissue	527
17.62.4 Member Function Documentation	528
17.62.4.1 calcQuads	528
17.62.4.2 divideCell	529
17.62.4.3 divideCell	530
17.62.4.4 divideCell	530
17.62.4.5 divideCell	531
17.62.4.6 divideCell	532
17.62.4.7 divideCell	533
17.62.4.8 divideCell	533
17.62.4.9 drawCell	534
17.62.4.10drawCell	535
17.62.4.11drawCellContour	538
17.62.4.12drawSimplifiedCell	539
17.62.4.13drawWalledCell	539
17.62.4.14drawWalledCell	540
17.62.4.15getCellPinchingParams	543
17.62.4.16getClosestMidAlgoParams	544
17.62.4.17getCloseWallAlgoParams	544
17.62.4.18getShortWallAlgoParams	545
17.62.4.19postDraw	546
17.62.4.20preDraw	546
17.62.4.21readParms	547
17.62.4.22readViewParms	548
17.62.4.23setCellPinchingParams	549
17.62.4.24valueCenterColor	550
17.62.4.25valueColor	551
17.62.5 Member Data Documentation	552

17.62.5.1 blending	552
17.62.5.2 cellDivAlg	552
17.62.5.3 cellMaxPinch	553
17.62.5.4 cellPinch	553
17.62.5.5 cellWallCorner	554
17.62.5.6 cellWallMin	554
17.62.5.7 cellWallSample	555
17.62.5.8 cellWallWidth	555
17.62.5.9 center_blending	556
17.62.5.10colorBegin	556
17.62.5.11colorCenter	557
17.62.5.12colorEnd	557
17.62.5.13contourColor	558
17.62.5.14drawBorders	558
17.62.5.15drawInsides	559
17.62.5.16palette	559
17.62.5.17sampleDx	560
17.62.5.18strictCellWallMin	560
17.63tissue_model::TissueModel< RealModel, TissueClass > Struct Template Reference	561
17.63.1 Detailed Description	563
17.63.2 Member Typedef Documentation	564
17.63.2.1 ordered_cells_t	564
17.63.3 Constructor & Destructor Documentation	564
17.63.3.1 TissueModel	564
17.63.4 Member Function Documentation	564
17.63.4.1 cellFromId	564
17.63.4.2 draw	565
17.63.4.3 drawWithNames	566
17.63.4.4 getCellCenterColor	566
17.63.4.5 getCellColor	567
17.63.4.6 initDraw	567
17.63.4.7 initialize	567
17.63.4.8 modifiedFiles	567

17.63.4.9 postDraw	568
17.63.4.10preDraw	568
17.63.4.11readParam	569
17.63.4.12readTissueParam	569
17.63.4.13step	569
17.63.4.14updateCellsArea	570
17.63.5 Member Data Documentation	570
17.63.5.1 backColor	570
17.63.5.2 drawNeighborhood	570
17.63.5.3 palette	571
17.63.5.4 rex	571
17.63.5.5 T	571
17.64bspline_tissue_model::TissueModel< RealModel, TissueClass > Class Template Reference	572
17.64.1 Detailed Description	575
17.64.2 Member Typedef Documentation	575
17.64.2.1 ordered_cells_t	575
17.64.3 Constructor & Destructor Documentation	575
17.64.3.1 TissueModel	575
17.64.4 Member Function Documentation	576
17.64.4.1 cellFromId	576
17.64.4.2 draw	576
17.64.4.3 drawWithNames	577
17.64.4.4 getCellCenterColor	578
17.64.4.5 getCellColor	578
17.64.4.6 initDraw	578
17.64.4.7 initialize	579
17.64.4.8 initTissue	579
17.64.4.9 modifiedFiles	580
17.64.4.10postDraw	580
17.64.4.11preDraw	580
17.64.4.12readParam	581
17.64.4.13readTissueParam	581
17.64.4.14registerFiles	582

17.64.4.15SetPos	582
17.64.4.16SetPos	582
17.64.4.17step	583
17.64.4.18updateCellsArea	583
17.64.4.19updatePositions	583
17.64.5 Member Data Documentation	584
17.64.5.1 backColor	584
17.64.5.2 cellInitWalls	584
17.64.5.3 drawNeighborhood	584
17.64.5.4 dt	585
17.64.5.5 growthStartTime	585
17.64.5.6 leaf	585
17.64.5.7 leafs	585
17.64.5.8 palette	586
17.64.5.9 T	586
17.64.5.10time	586
17.65algorithms::TriangleGrowth Class Reference	588
17.65.1 Detailed Description	589
17.65.2 Member Function Documentation	589
17.65.2.1 contour	589
17.65.2.2 endTime	590
17.65.2.3 errorMsg	590
17.65.2.4 init	590
17.65.2.5 readOBJs	593
17.65.2.6 readOBJs	593
17.65.2.7 setTime	601
17.65.2.8 size	601
17.65.2.9 startTime	601
17.65.2.10surface	602
17.65.2.11surface	602
17.65.2.12time	602
17.65.2.13timePos	603
17.65.2.14valid	603
17.66algorithms::TriangleSurface Class Reference	604

17.66.1 Detailed Description	606
17.66.2 Member Typedef Documentation	606
17.66.2.1 CellComplex	606
17.66.3 Member Function Documentation	606
17.66.3.1 center3d	606
17.66.3.2 centerPosition	607
17.66.3.3 clear	607
17.66.3.4 contour	607
17.66.3.5 normal	609
17.66.3.6 point3d	609
17.66.3.7 position	609
17.66.3.8 position	610
17.66.3.9 smoothNormal	611
17.66.3.10vector	611
17.66.3.11vector	612
17.66.3.12vector3d	612
17.66.4 Member Data Documentation	612
17.66.4.1 faces	612
17.66.4.2 pmin	612
17.66.4.3 S	613
17.66.4.4 time	613
17.66.4.5 vertices	613
17.67storage::TypeTraits< T > Struct Template Reference	614
17.67.1 Detailed Description	614
17.67.2 Member Data Documentation	614
17.67.2.1 id	614
17.67.2.2 name	614
17.67.2.3 type	615
17.68util::Vector< dim, T > Class Template Reference	616
17.68.1 Detailed Description	625
17.68.2 Constructor & Destructor Documentation	625
17.68.2.1 Vector	625
17.68.2.2 Vector	625
17.68.2.3 Vector	626

17.68.2.4 Vector	626
17.68.2.5 Vector	627
17.68.2.6 Vector	627
17.68.2.7 Vector	627
17.68.3 Member Function Documentation	627
17.68.3.1 begin	627
17.68.3.2 begin	628
17.68.3.3 c_data	628
17.68.3.4 cross	628
17.68.3.5 data	629
17.68.3.6 end	629
17.68.3.7 end	629
17.68.3.8 i	629
17.68.3.9 i	629
17.68.3.10	630
17.68.3.11j	630
17.68.3.12j	630
17.68.3.13j	631
17.68.3.14k	631
17.68.3.15k	631
17.68.3.16k	631
17.68.3.17	631
17.68.3.18	632
17.68.3.19	632
17.68.3.20mult	632
17.68.3.21norm	632
17.68.3.22normalize	633
17.68.3.23normalized	633
17.68.3.24normsq	633
17.68.3.25operator!=	633
17.68.3.26operator*	634
17.68.3.27operator*	634
17.68.3.28operator*=	634
17.68.3.29operator*=	635

17.68.3.30operator+	635
17.68.3.31operator+=	635
17.68.3.32operator-	635
17.68.3.33operator-	636
17.68.3.34operator-=	636
17.68.3.35operator/	636
17.68.3.36operator/	636
17.68.3.37operator/=	637
17.68.3.38operator/=	637
17.68.3.39operator/=	637
17.68.3.40operator<	638
17.68.3.41operator<=	638
17.68.3.42operator=	638
17.68.3.43operator=	639
17.68.3.44operator==	639
17.68.3.45operator>	639
17.68.3.46operator>=	639
17.68.3.47operator[]	640
17.68.3.48operator[]	640
17.68.3.49projectXY	640
17.68.3.50set	641
17.68.3.51set	641
17.68.3.52set	641
17.68.3.53set	641
17.68.3.54size	642
17.68.3.55	642
17.68.3.56	642
17.68.3.57	642
17.68.3.58x	643
17.68.3.59x	643
17.68.3.60x	643
17.68.3.61y	643
17.68.3.62y	644
17.68.3.63y	644

17.68.3.64z	644
17.68.3.65z	644
17.68.3.66z	645
17.68.4 Friends And Related Function Documentation	645
17.68.4.1 angle	645
17.68.4.2 angle	645
17.68.4.3 angle	645
17.68.4.4 angle	646
17.68.4.5 angle	646
17.68.4.6 divide	646
17.68.4.7 map	647
17.68.4.8 map	647
17.68.4.9 map	647
17.68.4.10map	648
17.68.4.11map	648
17.68.4.12map	648
17.68.4.13map	649
17.68.4.14max	649
17.68.4.15min	649
17.68.4.16multiply	650
17.68.4.17norm	650
17.68.4.18norm	650
17.68.4.19normalized	650
17.68.4.20normalized	651
17.68.4.21normsq	651
17.68.4.22normsq	651
17.68.4.23operator%	652
17.68.4.24operator%	652
17.68.4.25operator*	652
17.68.4.26operator+	652
17.68.4.27operator+	653
17.68.4.28operator-	653
17.68.4.29operator-	653
17.68.4.30operator^	653

17.68.4.3 <code>operator^</code>	654
17.68.4.3.2 <code>operator^</code>	654
17.68.4.3.3 <code>orthogonal</code>	654
17.69 <code>algorithms::TriangleSurface::Vertex</code> Struct Reference	656
17.69.1 Detailed Description	656
17.69.2 Member Data Documentation	656
17.69.2.1 <code>id</code>	656
17.69.2.2 <code>normal</code>	656
17.69.2.3 <code>pos</code>	656
17.70 <code>graph::Vertex< VertexContent ></code> Class Template Reference	657
17.70.1 Detailed Description	660
17.70.2 Member Typedef Documentation	661
17.70.2.1 <code>content_t</code>	661
17.70.2.2 <code>counted_content_t</code>	661
17.70.2.3 <code>identity_t</code>	661
17.70.2.4 <code>pointer</code>	661
17.70.2.5 <code>real_pointer</code>	661
17.70.3 Constructor & Destructor Documentation	662
17.70.3.1 <code>Vertex</code>	662
17.70.3.2 <code>Vertex</code>	662
17.70.3.3 <code>Vertex</code>	663
17.70.3.4 <code>Vertex</code>	663
17.70.3.5 <code>Vertex</code>	664
17.70.3.6 <code>Vertex</code>	664
17.70.3.7 <code>~Vertex</code>	664
17.70.4 Member Function Documentation	665
17.70.4.1 <code>acquire</code>	665
17.70.4.2 <code>content</code>	665
17.70.4.3 <code>id</code>	665
17.70.4.4 <code>isNull</code>	666
17.70.4.5 <code>isWeakRef</code>	667
17.70.4.6 <code>num</code>	667
17.70.4.7 <code>operator bool</code>	667
17.70.4.8 <code>operator!=</code>	667

17.70.4.9 operator!=	667
17.70.4.10operator*	668
17.70.4.11operator->	668
17.70.4.12operator->*	668
17.70.4.13operator->*	669
17.70.4.14operator<	669
17.70.4.15operator<=	669
17.70.4.16operator=	670
17.70.4.17operator==	670
17.70.4.18operator==	670
17.70.4.19operator>	671
17.70.4.20operator>=	671
17.70.4.21release	671
17.70.4.22serialize	672
17.70.4.23weakRef	672
17.70.5 Member Data Documentation	672
17.70.5.1 _content	672
17.70.5.2 cache	673
17.70.5.3 cache_source	673
17.70.5.4 null	674
17.71 Viewer Class Reference	676
17.71.1 Detailed Description	680
17.71.2 Constructor & Destructor Documentation	680
17.71.2.1 Viewer	680
17.71.2.2 ~Viewer	681
17.71.3 Member Function Documentation	681
17.71.3.1 addToolBar	681
17.71.3.2 addToolBar	681
17.71.3.3 addViewer	682
17.71.3.4 addViewer	682
17.71.3.5 cameraRecordingStart	682
17.71.3.6 cameraRecordingStop	682
17.71.3.7 configOpenGL	682
17.71.3.8 createViewer	683

17.71.3.9 createViewer	683
17.71.3.10 createViewer	683
17.71.3.11 domElement	683
17.71.3.12 draw	684
17.71.3.13 drawLight	684
17.71.3.14 drawWithNames	685
17.71.3.15 helpMenu	685
17.71.3.16 helpString	685
17.71.3.17 index	686
17.71.3.18 init	686
17.71.3.19 initFromDOMEElement	687
17.71.3.20 keyPressEvent	688
17.71.3.21 mainWindow	688
17.71.3.22 menuBar	689
17.71.3.23 model	689
17.71.3.24 modelRecordingStart	689
17.71.3.25 modelRecordingStop	690
17.71.3.26 newViewer	690
17.71.3.27 openOpenGLConfig	690
17.71.3.28 openScreenshotFormatDialog	690
17.71.3.29 postDraw	690
17.71.3.30 postSelection	691
17.71.3.31 preDraw	691
17.71.3.32 saveScreenshot	691
17.71.3.33 saveScreenshot	692
17.71.3.34 screenshotCounter	692
17.71.3.35 screenshotFileName	692
17.71.3.36 screenshotFormat	692
17.71.3.37 screenshotQuality	693
17.71.3.38 setIndex	693
17.71.3.39 setModel	693
17.71.3.40 setScreenshotCounter	694
17.71.3.41 setScreenshotFileName	694
17.71.3.42 setScreenshotFormat	694

17.71.3.43	setScreenshotQuality	695
17.71.3.44	setStatusMessage	695
17.71.3.45	showEvent	695
17.71.3.46	startRecordingCameraAnimation	695
17.71.3.47	startRecordingModelAnimation	696
17.71.3.48	statusMessage	696
17.71.3.49	stepDrawFinished	696
17.71.3.50	stopRecordingCameraAnimation	696
17.71.3.51	stopRecordingModelAnimation	696
17.71.3.52	updateAll	697
17.71.3.53	wasInitialized	697
17.72	graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact > Class Template Reference	698
17.72.1	Detailed Description	712
17.72.2	Performance notes	712
17.72.3	Member Typedef Documentation	713
17.72.3.1	circ_neighbor_iterator1	713
17.72.3.2	circ_neighbor_iterator1_range	713
17.72.3.3	circ_neighbor_iterator2	714
17.72.3.4	circ_neighbor_iterator2_range	714
17.72.3.5	const_circ_neighbor_iterator1	714
17.72.3.6	const_circ_neighbor_iterator1_range	714
17.72.3.7	const_circ_neighbor_iterator2	715
17.72.3.8	const_circ_neighbor_iterator2_range	715
17.72.3.9	const_edge1_t	715
17.72.3.10	const_edge2_t	715
17.72.3.11	const_iterator1	716
17.72.3.12	const_iterator2	716
17.72.3.13	const_neighbor1_found_t	716
17.72.3.14	const_neighbor2_found_t	717
17.72.3.15	const_neighbor_iterator1	717
17.72.3.16	const_neighbor_iterator1_range	717
17.72.3.17	const_neighbor_iterator2	717
17.72.3.18	const_neighbor_iterator2_range	718

17.72.3.19	const_range_vertex1	718
17.72.3.20	const_range_vertex2	718
17.72.3.21	edge1_t	718
17.72.3.22	edge2_t	719
17.72.3.23	edge_list1_t	719
17.72.3.24	edge_list2_t	719
17.72.3.25	in_edges1_t	719
17.72.3.26	in_edges2_t	720
17.72.3.27	neighbor_iterator1	720
17.72.3.28	neighbor_iterator1_range	720
17.72.3.29	neighbor_iterator2	720
17.72.3.30	neighbor_iterator2_range	721
17.72.3.31	int_const_neighbor_iterator1	721
17.72.3.32	int_const_neighbor_iterator2	721
17.72.3.33	int_neighbor_iterator1	722
17.72.3.34	int_neighbor_iterator2	722
17.72.3.35	iterator1	722
17.72.3.36	iterator2	722
17.72.3.37	lookup1_t	723
17.72.3.38	lookup2_t	723
17.72.3.39	neighbor1_found_t	723
17.72.3.40	neighbor1_t	724
17.72.3.41	neighbor2_found_t	724
17.72.3.42	neighbor2_t	724
17.72.3.43	neighbor_iterator1	724
17.72.3.44	neighbor_iterator1_range	725
17.72.3.45	neighbor_iterator2	725
17.72.3.46	neighbor_iterator2_range	725
17.72.3.47	neighborhood1_t	725
17.72.3.48	neighborhood1_value_type	726
17.72.3.49	neighborhood2_t	726
17.72.3.50	neighborhood2_value_type	726
17.72.3.51	range_vertex1	726
17.72.3.52	range_vertex2	727

17.72.3.53single_neighborhood1_t	727
17.72.3.54single_neighborhood2_t	727
17.72.3.55size_type	727
17.72.3.56vertex1_t	728
17.72.3.57vertex2_t	728
17.72.4 Constructor & Destructor Documentation	728
17.72.4.1 VVBiGraph	728
17.72.4.2 VVBiGraph	728
17.72.5 Member Function Documentation	729
17.72.5.1 any_vertex1	729
17.72.5.2 any_vertex2	729
17.72.5.3 anyIn	730
17.72.5.4 anyIn	730
17.72.5.5 begin_vertex1	731
17.72.5.6 begin_vertex1	731
17.72.5.7 begin_vertex2	731
17.72.5.8 begin_vertex2	732
17.72.5.9 clear	732
17.72.5.10clear	732
17.72.5.11clear	733
17.72.5.12clear	733
17.72.5.13clear	733
17.72.5.14contains	734
17.72.5.15contains	734
17.72.5.16count	734
17.72.5.17count	735
17.72.5.18edge	735
17.72.5.19edge	735
17.72.5.20edge	736
17.72.5.21edge	736
17.72.5.22empty	736
17.72.5.23empty	737
17.72.5.24empty	737
17.72.5.25end_vertex1	737

17.72.5.26end_vertex1	738
17.72.5.27end_vertex2	738
17.72.5.28end_vertex2	738
17.72.5.29erase	739
17.72.5.30erase	739
17.72.5.31erase	739
17.72.5.32erase	740
17.72.5.33eraseAllEdges	740
17.72.5.34eraseAllEdges	741
17.72.5.35eraseAllEdges	741
17.72.5.36eraseAllEdges	742
17.72.5.37eraseAllEdges	743
17.72.5.38eraseEdge	743
17.72.5.39eraseEdge	743
17.72.5.40eraseEdge	743
17.72.5.41eraseEdge	744
17.72.5.42eraseEdge	744
17.72.5.43eraseEdge	745
17.72.5.44find	745
17.72.5.45find	746
17.72.5.46find	746
17.72.5.47find	746
17.72.5.48findIn	747
17.72.5.49findIn	747
17.72.5.50findIn	748
17.72.5.51findIn	748
17.72.5.52findInVertex	749
17.72.5.53findInVertex	749
17.72.5.54findInVertex	750
17.72.5.55findInVertex	750
17.72.5.56findVertex	751
17.72.5.57findVertex	752
17.72.5.58findVertex	752
17.72.5.59findVertex	753

17.72.5.60flag	754
17.72.5.61flag	754
17.72.5.62flagged	755
17.72.5.63flagged	755
17.72.5.64get_vertex1	755
17.72.5.65get_vertex2	756
17.72.5.66AnyIn	756
17.72.5.67AnyIn	757
17.72.5.68Empty	757
17.72.5.69Empty	757
17.72.5.70Neighbors	758
17.72.5.71Neighbors	758
17.72.5.72insert	759
17.72.5.73insert	759
17.72.5.74insert	759
17.72.5.75insert	759
17.72.5.76insert	760
17.72.5.77insert	760
17.72.5.78insertEdge	761
17.72.5.79insertEdge	761
17.72.5.80insertInEdge	762
17.72.5.81insertInEdge	762
17.72.5.82Valence	763
17.72.5.83Valence	763
17.72.5.84nb_vertices1	764
17.72.5.85nb_vertices2	764
17.72.5.86neighbors	764
17.72.5.87neighbors	764
17.72.5.88neighbors	765
17.72.5.89neighbors	765
17.72.5.90neighbors	766
17.72.5.91neighbors	766
17.72.5.92neighbors	767
17.72.5.93neighbors	767

17.72.5.94	nextTo	768
17.72.5.95	nextTo	768
17.72.5.96	no_vertex1	769
17.72.5.97	no_vertex2	769
17.72.5.98	operator=	770
17.72.5.99	operator==	771
17.72.5.100	prevTo	773
17.72.5.101	prevTo	773
17.72.5.102	reference	774
17.72.5.103	reference	774
17.72.5.104	replace	775
17.72.5.105	replace	775
17.72.5.106	size	776
17.72.5.107	source	777
17.72.5.108	source	777
17.72.5.109	spliceAfter	777
17.72.5.110	spliceAfter	778
17.72.5.111	spliceBefore	779
17.72.5.112	spliceBefore	779
17.72.5.113	swap	780
17.72.5.114	target	780
17.72.5.115	target	781
17.72.5.116	undoInEdge	781
17.72.5.117	undoInEdge	781
17.72.5.118	updateEdgeCache	782
17.72.5.119	updateEdgeCache	782
17.72.5.120	updateVertexCache	783
17.72.5.121	updateVertexCache	783
17.72.5.122	valence	784
17.72.5.123	valence	784
17.72.5.124	vertices1	784
17.72.5.125	vertices1	785
17.72.5.126	vertices2	785
17.72.5.127	vertices2	785

17.72.6 Member Data Documentation	786
17.72.6.1 graph_id	786
17.72.6.2 lookup1	786
17.72.6.3 lookup2	787
17.72.6.4 neighborhood1	787
17.72.6.5 neighborhood2	788
17.73 vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCell- Content, compact, LeafClass > Class Template Reference	790
17.73.1 Detailed Description	795
17.73.2 Member Typedef Documentation	796
17.73.2.1 cell	796
17.73.2.2 cell_arc	796
17.73.2.3 cell_edge	796
17.73.2.4 cell_graph	797
17.73.2.5 cell_junction_edge	797
17.73.2.6 complex_graph	797
17.73.2.7 const_cell_edge	798
17.73.2.8 const_cell_junction_edge	798
17.73.2.9 const_junction_cell_edge	798
17.73.2.10 const_wall	799
17.73.2.11 division_data	799
17.73.2.12 junction	799
17.73.2.13 junction_cell_edge	800
17.73.2.14 wall	800
17.73.2.15 wall_arc	800
17.73.2.16 wall_graph	801
17.73.3 Constructor & Destructor Documentation	801
17.73.3.1 VVComplex	801
17.73.3.2 VVComplex	802
17.73.3.3 ~VVComplex	802
17.73.4 Member Function Documentation	803
17.73.4.1 addCell	803
17.73.4.2 addCell	804

17.73.4.3 addCell	804
17.73.4.4 addCell	805
17.73.4.5 addCell	806
17.73.4.6 addCell	806
17.73.4.7 addCell	807
17.73.4.8 addCell	807
17.73.4.9 addCellExtra	808
17.73.4.10addCellUnsorted	808
17.73.4.11adjacentCell	810
17.73.4.12adjacentCells	810
17.73.4.13border	811
17.73.4.14border	811
17.73.4.15border	812
17.73.4.16border	812
17.73.4.17clear	813
17.73.4.18clearOldGraphs	813
17.73.4.19connectFromJunctions	814
17.73.4.20deleteCell	815
17.73.4.21deleteCellInGraphs	815
17.73.4.22deleteJunction	816
17.73.4.23deleteJunctionExtra	817
17.73.4.24divideCell	817
17.73.4.25divideCell	818
17.73.4.26divideCell	818
17.73.4.27divideCell	819
17.73.4.28divideCell	820
17.73.4.29divideCell	820
17.73.4.30interfaceWall	822
17.73.4.31interfaceWallSpan	822
17.73.4.32mergeCells	823
17.73.4.33reconnectGraphs	824
17.73.4.34saveSubgraphs	824
17.73.4.35serialize	825
17.73.4.36splitWall	826

17.73.4.37splitWall	826
17.73.4.38splitWallExtra	827
17.73.5 Member Data Documentation	827
17.73.5.1 C	827
17.73.5.2 model	828
17.73.5.3 OldC	829
17.73.5.4 OldS	829
17.73.5.5 OldW	830
17.73.5.6 S	830
17.73.5.7 save_parameters	831
17.73.5.8 W	832
17.74vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact > Class Template Reference	833
17.74.1 Detailed Description	836
17.74.2 Member Typedef Documentation	837
17.74.2.1 const_iterator1	837
17.74.2.2 const_iterator2	837
17.74.2.3 division_data	837
17.74.2.4 iterator1	837
17.74.2.5 iterator2	838
17.74.2.6 size_type	838
17.74.3 Member Function Documentation	838
17.74.3.1 addCell	838
17.74.3.2 addCell	839
17.74.3.3 addCell	839
17.74.3.4 addCell	840
17.74.3.5 addCell	840
17.74.3.6 addCell	841
17.74.3.7 addCell	841
17.74.3.8 adjacentCell	842
17.74.3.9 adjacentCells	842
17.74.3.10any_cell	843
17.74.3.11any_junction	843

17.74.3.12border	844
17.74.3.13border	844
17.74.3.14border	844
17.74.3.15deleteCell	845
17.74.3.16divideCell	846
17.74.3.17get_cell	847
17.74.3.18get_junction	848
17.74.3.19interfaceWall	848
17.74.3.20interfaceWallSpan	849
17.74.3.21mergeCells	849
17.74.3.22nb_cells	850
17.74.3.23nb_junctions	851
17.74.3.24no_cell	851
17.74.3.25no_junction	851
17.74.3.26splitWall	851
17.74.3.27splitWall	852
17.75storage::VVEStorage Class Reference	853
17.75.1 Detailed Description	856
17.75.2 Member Typedef Documentation	857
17.75.2.1 reference_t	857
17.75.3 Constructor & Destructor Documentation	857
17.75.3.1 ~VVEStorage	857
17.75.4 Member Function Documentation	857
17.75.4.1 checkNextField	857
17.75.4.2 clear	857
17.75.4.3 clearLastError	857
17.75.4.4 endCompound	858
17.75.4.5 endReference	858
17.75.4.6 field	858
17.75.4.7 field	859
17.75.4.8 field	859
17.75.4.9 field	859
17.75.4.10field	860
17.75.4.11field	860

17.75.4.12field	860
17.75.4.13field	860
17.75.4.14field	860
17.75.4.15field	860
17.75.4.16field	860
17.75.4.17field	860
17.75.4.18field	860
17.75.4.19field	861
17.75.4.20field	861
17.75.4.21field	861
17.75.4.22field	861
17.75.4.23field	861
17.75.4.24field	861
17.75.4.25field	861
17.75.4.26filename	861
17.75.4.27fileVersion	862
17.75.4.28ignore	862
17.75.4.29lastError	862
17.75.4.30lastErrorString	862
17.75.4.31reader	863
17.75.4.32reference	863
17.75.4.33reference	863
17.75.4.34serialize	864
17.75.4.35setLastError	864
17.75.4.36setOption	865
17.75.4.37startCompound	865
17.75.4.38startReference	865
17.75.4.39version	866
17.75.4.40writer	866
17.76storage::old::VVEStorage_BINReader Class Reference	867
17.76.1 Detailed Description	868
17.76.2 Member Typedef Documentation	868
17.76.2.1 reference_t	868
17.76.3 Member Function Documentation	869

17.76.3.1 checkNextField	869
17.76.3.2 endCompound	869
17.76.3.3 endReference	869
17.76.3.4 filename	869
17.76.3.5 reader	870
17.76.3.6 serialize	870
17.76.3.7 setLastError	871
17.76.3.8 setOption	872
17.76.3.9 startCompound	872
17.76.3.10startReference	873
17.76.3.11writer	874
17.77storage::VVEStorage_BINReader Class Reference	875
17.77.1 Detailed Description	877
17.77.2 Member Typedef Documentation	877
17.77.2.1 reference_t	877
17.77.3 Member Function Documentation	877
17.77.3.1 checkNextField	877
17.77.3.2 endCompound	877
17.77.3.3 endReference	878
17.77.3.4 filename	878
17.77.3.5 ignore	878
17.77.3.6 reader	879
17.77.3.7 serialize	879
17.77.3.8 setLastError	881
17.77.3.9 setOption	881
17.77.3.10startCompound	882
17.77.3.11startReference	882
17.77.3.12writer	883
17.77.4 Member Data Documentation	884
17.77.4.1 frame_stack	884
17.77.4.2 top_level_compounds	884
17.78storage::VVEStorage_BINWriter Class Reference	885
17.78.1 Detailed Description	886
17.78.2 Member Typedef Documentation	886

17.78.2.1 reference_t	886
17.78.3 Member Function Documentation	887
17.78.3.1 checkNextField	887
17.78.3.2 endCompound	887
17.78.3.3 endReference	888
17.78.3.4 filename	888
17.78.3.5 reader	888
17.78.3.6 serialize	888
17.78.3.7 startCompound	890
17.78.3.8 startReference	890
17.78.3.9 writer	891
17.78.4 Member Data Documentation	891
17.78.4.1 frame_stack	891
17.79storage::VVEStorage_XMLReader Class Reference	892
17.79.1 Detailed Description	893
17.79.2 Member Typedef Documentation	893
17.79.2.1 reference_t	893
17.79.3 Member Function Documentation	894
17.79.3.1 checkNextField	894
17.79.3.2 endCompound	894
17.79.3.3 endReference	894
17.79.3.4 filename	895
17.79.3.5 reader	895
17.79.3.6 serialize	895
17.79.3.7 setLastError	896
17.79.3.8 setOption	897
17.79.3.9 startCompound	897
17.79.3.10startReference	898
17.79.3.11writer	899
17.80storage::VVEStorage_XMLWriter Class Reference	900
17.80.1 Detailed Description	901
17.80.2 Member Typedef Documentation	901
17.80.2.1 reference_t	901
17.80.3 Member Function Documentation	902

17.80.3.1 checkNextField	902
17.80.3.2 endCompound	902
17.80.3.3 endReference	902
17.80.3.4 filename	903
17.80.3.5 reader	903
17.80.3.6 serialize	903
17.80.3.7 setOption	904
17.80.3.8 startCompound	905
17.80.3.9 startReference	905
17.80.3.10writer	906
17.81 graph::VVGraph< VertexContent, EdgeContent, compact > Class Template Reference	907
17.81.1 Detailed Description	915
17.81.2 Performance notes	915
17.81.3 Member Typedef Documentation	916
17.81.3.1 arc_t	916
17.81.3.2 circ_neighbor_iterator	916
17.81.3.3 circ_neighbor_iterator_range	917
17.81.3.4 const_circ_neighbor_iterator	917
17.81.3.5 const_circ_neighbor_iterator_range	917
17.81.3.6 const_edge_t	917
17.81.3.7 const_iterator	917
17.81.3.8 const_neighbor_found_t	918
17.81.3.9 const_neighbor_iterator	918
17.81.3.10const_neighbor_iterator_range	918
17.81.3.11edge_list_t	918
17.81.3.12edge_t	918
17.81.3.13in_edges_t	919
17.81.3.14neighbor_iterator	919
17.81.3.15neighbor_iterator_range	919
17.81.3.16nt_const_neighbor_iterator	919
17.81.3.17nt_neighbor_iterator	919
17.81.3.18iterator	920
17.81.3.19lookup_t	920

17.81.3.20neighbor_found_t	920
17.81.3.21neighbor_iterator	920
17.81.3.22neighbor_iterator_range	921
17.81.3.23neighborhood_t	921
17.81.3.24neighborhood_value_type	921
17.81.3.25size_type	921
17.81.3.26value_type	921
17.81.3.27vertex_t	922
17.81.4 Constructor & Destructor Documentation	922
17.81.4.1 VVGraph	922
17.81.4.2 VVGraph	922
17.81.5 Member Function Documentation	922
17.81.5.1 any	922
17.81.5.2 anyIn	923
17.81.5.3 arc	924
17.81.5.4 begin	925
17.81.5.5 begin	925
17.81.5.6 clear	925
17.81.5.7 clear	926
17.81.5.8 clear	926
17.81.5.9 contains	927
17.81.5.10count	927
17.81.5.11edge	927
17.81.5.12edge	928
17.81.5.13empty	929
17.81.5.14empty	930
17.81.5.15end	930
17.81.5.16end	930
17.81.5.17erase	930
17.81.5.18erase	931
17.81.5.19eraseAllEdges	932
17.81.5.20eraseAllEdges	933
17.81.5.21eraseAllEdges	933
17.81.5.22eraseEdge	934

17.81.5.23	eraseEdge	934
17.81.5.24	eraseEdge	935
17.81.5.25	find	936
17.81.5.26	find	936
17.81.5.27	findIn	936
17.81.5.28	findIn	937
17.81.5.29	findInVertex	938
17.81.5.30	findInVertex	939
17.81.5.31	findVertex	940
17.81.5.32	findVertex	941
17.81.5.33	flag	942
17.81.5.34	flagged	943
17.81.5.35	AnyIn	943
17.81.5.36	Empty	944
17.81.5.37	Neighbors	945
17.81.5.38	insert	945
17.81.5.39	insert	946
17.81.5.40	insert	946
17.81.5.41	insertEdge	947
17.81.5.42	insertEdges	948
17.81.5.43	insertInEdge	948
17.81.5.44	Valence	949
17.81.5.45	neighbors	950
17.81.5.46	neighbors	950
17.81.5.47	neighbors	951
17.81.5.48	neighbors	952
17.81.5.49	nextTo	952
17.81.5.50	operator=	953
17.81.5.51	operator==	955
17.81.5.52	operator[]	956
17.81.5.53	prevTo	957
17.81.5.54	reference	957
17.81.5.55	replace	958
17.81.5.56	size	960

17.81.5.57source	960
17.81.5.58spliceAfter	960
17.81.5.59spliceAfter	961
17.81.5.60spliceBefore	963
17.81.5.61spliceBefore	963
17.81.5.62store_vertices	965
17.81.5.63subgraph	965
17.81.5.64swap	967
17.81.5.65target	967
17.81.5.66undoInEdge	968
17.81.5.67updateEdgeCache	968
17.81.5.68updateVertexCache	969
17.81.5.69valence	969
17.81.6 Member Data Documentation	970
17.81.6.1 graph_id	970
17.81.6.2 lookup	970
17.81.6.3 neighborhood	971
17.82util::WatchDog Class Reference	973
17.82.1 Detailed Description	974
17.82.2 Constructor & Destructor Documentation	974
17.82.2.1 WatchDog	974
17.82.3 Member Function Documentation	975
17.82.3.1 addObject	975
17.82.3.2 addObject	975
17.82.3.3 changeFilename	975
17.82.3.4 guarded	976
17.82.3.5 removeObject	976
17.82.3.6 removeObject	976
17.82.3.7 watch	977
17.82.4 Member Data Documentation	977
17.82.4.1 fileObjects	977
17.82.4.2 model	978
17.82.4.3 names	978
17.83graph::WeakVertex< VertexContent > Class Template Reference	979

17.83.1 Detailed Description	980
17.83.2 Member Typedef Documentation	980
17.83.2.1 content_t	980
17.83.2.2 identity_t	980
17.83.2.3 pointer	981
17.83.2.4 strong_ref	981
17.83.3 Constructor & Destructor Documentation	981
17.83.3.1 WeakVertex	981
17.83.3.2 WeakVertex	981
17.83.3.3 WeakVertex	981
17.83.3.4 WeakVertex	982
17.83.4 Member Function Documentation	982
17.83.4.1 isNull	982
17.83.4.2 operator=	982
18 File Documentation	985
18.1 doc/examples/context_menu.cpp File Reference	985
18.1.1 Detailed Description	985
18.2 doc/examples/hello_world.cpp File Reference	986
18.2.1 Detailed Description	986
18.3 doc/examples/select.cpp File Reference	987
18.3.1 Detailed Description	987
18.4 doc/examples/simple_model.cpp File Reference	988
18.4.1 Detailed Description	988
18.5 vvelib/algorithms/cellsystem.h File Reference	989
18.5.1 Detailed Description	990
18.6 vvelib/algorithms/complex.h File Reference	991
18.6.1 Detailed Description	995
18.6.2 Define Documentation	995
18.6.2.1 EXPORT_COMPLEX_EDGES	995
18.6.2.2 EXPORT_COMPLEX_EDGES_PREFIX	995
18.6.2.3 EXPORT_COMPLEX_GRAPHS	996
18.6.2.4 EXPORT_COMPLEX_GRAPHS_PREFIX	996
18.6.2.5 EXPORT_COMPLEX_TYPES	996

18.6.2.6	EXPORT_COMPLEX_TYPES_PREFIX	997
18.6.2.7	EXPORT_COMPLEX_VERTICES	997
18.6.2.8	EXPORT_COMPLEX_VERTICES_PREFIX	997
18.6.2.9	IMPORT_COMPLEX_EDGES	998
18.6.2.10	IMPORT_COMPLEX_GRAPHS	998
18.6.2.11	IMPORT_COMPLEX_MODEL	998
18.6.2.12	IMPORT_COMPLEX_TYPES	999
18.6.2.13	IMPORT_COMPLEX_VERTICES	999
18.7	vvelib/storage/complex.h File Reference	1000
18.7.1	Detailed Description	1000
18.8	vvelib/algorithms/draw_graphs.h File Reference	1001
18.8.1	Detailed Description	1001
18.9	vvelib/algorithms/graph.h File Reference	1002
18.9.1	Detailed Description	1002
18.10	vvelib/storage/graph.h File Reference	1004
18.10.1	Detailed Description	1004
18.11	vvelib/algorithms/insert.h File Reference	1005
18.11.1	Detailed Description	1005
18.12	vvelib/algorithms/parallel.h File Reference	1006
18.12.1	Detailed Description	1007
18.13	vvelib/algorithms/solver.h File Reference	1008
18.13.1	Detailed Description	1009
18.14	vvelib/algorithms/split.h File Reference	1010
18.14.1	Detailed Description	1010
18.15	vvelib/algorithms/tissue.h File Reference	1011
18.15.1	Detailed Description	1012
18.16	vvelib/algorithms/triangle_growth.h File Reference	1013
18.16.1	Detailed Description	1014
18.17	vvelib/bspline_tissue_model.h File Reference	1015
18.17.1	Detailed Description	1016
18.17.2	Define Documentation	1019
18.17.2.1	EndModel	1019
18.17.2.2	StartModel	1019
18.17.3	Typedef Documentation	1019

18.17.3.1 Color	1019
18.18 vvelib/factory/complex_grid.h File Reference	1020
18.18.1 Detailed Description	1021
18.19 vvelib/factory/grid.h File Reference	1022
18.19.1 Detailed Description	1022
18.20 vvelib/factory/objreader.h File Reference	1023
18.20.1 Detailed Description	1024
18.21 vvelib/geometry/area.h File Reference	1025
18.21.1 Detailed Description	1025
18.22 vvelib/geometry/coordinates.h File Reference	1026
18.22.1 Detailed Description	1026
18.23 vvelib/geometry/geometry.h File Reference	1027
18.23.1 Detailed Description	1027
18.24 vvelib/geometry/intersection.h File Reference	1028
18.24.1 Detailed Description	1029
18.25 vvelib/geometry/projection.h File Reference	1030
18.25.1 Detailed Description	1030
18.26 vvelib/geometry/quaternion.h File Reference	1031
18.26.1 Detailed Description	1031
18.27 vvelib/graph/edge.h File Reference	1032
18.27.1 Detailed Description	1032
18.28 vvelib/graph/vertex.h File Reference	1033
18.28.1 Detailed Description	1034
18.29 vvelib/graph/vvbigraph.h File Reference	1035
18.29.1 Detailed Description	1036
18.30 vvelib/graph/vvgraph.h File Reference	1037
18.30.1 Detailed Description	1038
18.31 vvelib/model.h File Reference	1039
18.31.1 Detailed Description	1039
18.32 vvelib/shape/quadric.h File Reference	1040
18.32.1 Detailed Description	1040
18.33 vvelib/storage/storage.h File Reference	1041
18.33.1 Detailed Description	1042
18.34 vvelib/storage/storage_xml.h File Reference	1043

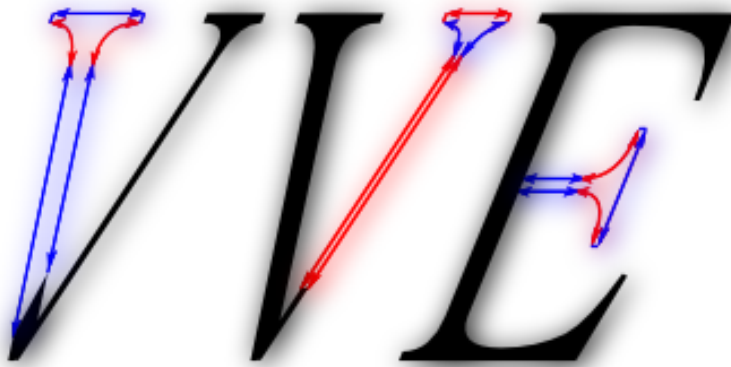
18.34.1 Detailed Description	1043
18.35 vvelib/storage/types.h File Reference	1044
18.35.1 Detailed Description	1045
18.35.2 Define Documentation	1045
18.35.2.1 FOR_ALL_TYPEIDS	1045
18.35.2.2 FOR_ALL_TYPES	1046
18.35.2.3 FOR_ALL_TYPES_NOSTRING	1046
18.36 vvelib/storage/vector.h File Reference	1048
18.36.1 Detailed Description	1048
18.37 vvelib/util/vector.h File Reference	1049
18.37.1 Detailed Description	1050
18.38 vvelib/test/graph.vvh File Reference	1051
18.38.1 Detailed Description	1052
18.38.2 Define Documentation	1052
18.38.2.1 CHECK_NEIGHBORS_BIGGRAPH	1052
18.38.2.2 CHECK_NEIGHBORS_GRAPH	1052
18.38.2.3 CHECK_SYMMETRIC_GRAPH	1052
18.39 vvelib/test/test.h File Reference	1053
18.39.1 Detailed Description	1054
18.39.2 Define Documentation	1054
18.39.2.1 CHECK_EQUAL	1054
18.39.2.2 CHECK_FLOAT	1054
18.39.2.3 CHECK_FLOAT_DIFF	1055
18.39.2.4 CHECK_NOTEQUAL	1055
18.39.2.5 CHECK_ZERO	1055
18.39.2.6 TEST_FCT	1055
18.40 vvelib/tissue_model.h File Reference	1056
18.40.1 Detailed Description	1057
18.41 vvelib/util/assert.h File Reference	1060
18.41.1 Detailed Description	1060
18.41.2 Define Documentation	1060
18.41.2.1 vvassert	1060
18.41.2.2 vvassert_msg	1061
18.42 vvelib/util/buffer.h File Reference	1062

18.42.1 Detailed Description	1062
18.43 vvelib/util/clamp.h File Reference	1063
18.43.1 Detailed Description	1063
18.44 vvelib/util/color.h File Reference	1064
18.44.1 Detailed Description	1065
18.45 vvelib/util/contour.h File Reference	1066
18.45.1 Detailed Description	1066
18.46 vvelib/util/dir.h File Reference	1067
18.46.1 Detailed Description	1067
18.47 vvelib/util/forall.h File Reference	1068
18.47.1 Detailed Description	1069
18.47.2 Define Documentation	1069
18.47.2.1 forall	1069
18.47.2.2 forall_named	1071
18.47.2.3 forall_range	1072
18.47.2.4 forall_reverse	1072
18.48 vvelib/util/function.h File Reference	1073
18.48.1 Detailed Description	1073
18.49 vvelib/util/glerrorcheck.h File Reference	1074
18.49.1 Detailed Description	1074
18.49.2 Define Documentation	1074
18.49.2.1 OPENGGL_ERROR_CHECK	1074
18.50 vvelib/util/graph_inset.h File Reference	1075
18.50.1 Detailed Description	1075
18.51 vvelib/util/leaf_class.h File Reference	1076
18.51.1 Detailed Description	1076
18.52 vvelib/util/materials.h File Reference	1077
18.52.1 Detailed Description	1077
18.53 vvelib/util/member_iterator.h File Reference	1078
18.53.1 Detailed Description	1078
18.54 vvelib/util/palette.h File Reference	1079
18.54.1 Detailed Description	1079
18.55 vvelib/util/parms.h File Reference	1080
18.55.1 Detailed Description	1080

18.56 vvelib/util/point.h File Reference	1081
18.56.1 Detailed Description	1081
18.57 vvelib/util/random.h File Reference	1082
18.57.1 Detailed Description	1083
18.58 vvelib/util/range.h File Reference	1084
18.58.1 Detailed Description	1085
18.59 vvelib/util/static_assert.h File Reference	1086
18.59.1 Detailed Description	1086
18.59.2 Define Documentation	1086
18.59.2.1 STATIC_ASSERT	1086
18.60 vvelib/util/tensor.h File Reference	1088
18.60.1 Detailed Description	1088
18.61 vvelib/util/texture.h File Reference	1089
18.61.1 Detailed Description	1089
18.62 vvelib/util/tie.h File Reference	1090
18.62.1 Detailed Description	1090
18.63 vvelib/util/watchdog.h File Reference	1091
18.63.1 Detailed Description	1091

Chapter 1

VVE documentation



This site presents the VVE documentation. You will find here

1. An [introduction to VVE](#)
2. How to use the [model helpers](#)
3. Some [VVE examples](#)
4. A manual on [how to create your own VVE libraries](#)
5. At last how to [install VVE](#) on your machine

Chapter 2

Creating libraries for VVE

Chapter 3

VVE examples

Chapter 4

VVE model helpers

4.1 Table of content

- [Basics of model helpers](#)
 - [What is a model helper?](#)
 - [Commands and methods found in all helpers](#)
- [Existing model helpers](#)
 - [vvelib/tissue_model.h](#)
 - [vvelib/bspline_tissue_model.h](#)

4.2 Basics of model helpers

4.2.1 What is a model helper?

To allow for as much flexibility as possible, VVE allows the user to parametrize a lot of types. While this is essential to develop new ideas and complex models, simple models become cumbersome to write.

Model helpers predefine the main data structures, provide initialization, default drawing functions and more. In some extreme case (like the [Cell-System model helper](#)) the smallest possible model can be three line long. In some other case, it is a bit longer to get any useful model.

4.2.2 Commands and methods found in all helpers

The template to use helpers is always the same:

```
#define VertexAttributes \  
    double a; \  
    double b;  
  
#include <helper_model.h>  
  
StartModel  
  
void initialize()  
{  
    // Custom initialization  
}  
  
void readParam(util::Parms& parms)  
{  
    // Read custom parameters  
}  
  
void step()  
{  
    // Redefining the default step  
    ParentClass::step(); // Call the default step function  
}
```

EndModel

From within the model, there will always be a set of variables and vertex attributes predefined. When adding your own attributes, be careful to choose name not clashing with already defined attributes. There are only two class names common to all the helpers:

- `ModelClass` is defined after the `StartModel` and is the name of the model class being defined
- `ParentClass` is defined within the model (i.e. in between `StartModel` and `EndModel`) and can be used to call the original method when you define your own.

In addition to these functions, all methods of the [Model](#) class can be overloaded to personalize the model.

4.3 Existing model helpers

4.3.1 Cell-System model helper

4.3.2 Growing tissue model helper

Chapter 5

Installing VVE

- [Linux](#)
- [Windows](#)
 - [From binary distribution](#)
 - [From source](#)
 - [Post-installation specific to L-Studio ATE](#)

5.1 Linux

Linux installation is, for now, only from source. You will need:

- VLab (optional but recommended)
- Qt 4.2 or greater
- CMake
- g++ 4.2

First, you have to compile the **QGLViewer** provided (it is slightly modified from the original). It is compiled using qmake, a readme is provided by the **QGLViewer** development team. You will want to install **QGLViewer** in your system.

Note

To simplify the following explanation, I will refer to the VVE source directory using `~vve` and the VVE build directory using `~vve_build`. You will replace these by whatever directory you chose.

To install VVE, create a build directory outside the VVE source directory. Then, go inside `~vve_build` and launch the command: `ccmake ~vve`

Configure a first time by pressing the 'c' key.

You will see a lot of options. If everything is installed in your system (including **QGLViewer**), the only options you will want to set are:

- `CMAKE_BUILD_TYPE` - set it to `Release`
- `CMAKE_INSTALL_PREFIX` - set it to whatever directory you want to install it into

If **QGLViewer** is not installed in your system but locally, set `QGLVIEWER_DIR` to the directory containing the **QGLViewer** directory (which contains the header files) and `QGLVIEWER_LIBRARY_DIR` to the directory containing `libQGLViewer.so`.

Now, reconfigure by pressing 'c' and, if everything is setup correctly, you will be able to generate the makefiles by pressing 'g'.

`ccmake` will exit by itself, leaving you into your build directory. From then, simply type:

```
$ make -j2
$ su
# make install
```

Note

The \$ and # signs indicate respectively the user and root prompt, you should not type them.

Now, setup a `VVEDIR` environment variable to whatever directory you installed VVE in and make sure `$VVEDIR/bin` is in the `$PATH` and `$VVEDIR/lib` in `$LD_LIBRARY_PATH` (or in the system library lookup paths).

You are setup and can compile/run VVE models in VLab (and even outside).

5.2 Windows

5.2.1 From binary distribution

To install VVE on windows (using LStudio) you will first need to install MinGW32 with g++-4.2 and Qt 4.3 for MingW32. The binary archive provides you with all the needed software.

1. Install MinGW using the installer
2. Unpack the content of `mingw_update.zip` into the install directory of MinGW (by default `C:\mingw`)
3. Install Qt 4.3 for MinGW
4. Unpack `Lstudio.exe.zip` into your LStudio `bin` directory (this operation will replace your LStudio.exe, you might want to save it first).
5. Unpack `vve.zip` into `C:\` (or any other directory, but `C:\` is recommended)
6. Copy `vveinterpreter.exe` and `QGLViewer2.dll` (found in the `bin` directory of VVE) into the `bin` directory of LStudio
7. Setup your environment variables:
 - `VVEDIR` should contain the installation directory of VVE (by default `C:\VVE`)
 - `QMAKESPEC` must be set to `win32-g++`
 - `Q4DIR` should contain the installation directory of Qt (by default `C:\Qt\4.3`)
 - Make sure `Q4DIR%\bin`, `Q4DIR%\lib` and `C:\mingw\bin` (or wherever the `bin` directory of MinGW is) are in your `PATH` environment variable

You are setup and can compile/run VVE models in LStudio.

5.2.2 From source

To install from source, start by the steps 1 to 4 and 7 of the binary installation. In addition, you will need to install `cmake` for windows. You can download it at <http://www.cmake.org>

Note

During this process, I will assume you want to install VVE into `C:\VVE`. If this is not the case, change the directories accordingly.

The first project to compile is **QGLViewer**. It is compiled using `qmake`. Once it is compile, install it into `C:\VVE`. At the end, you should have `QGLViewer.dll` in `C:\VVE\lib` and a **QGLViewer** directory into `C:\VVE\include`. Next, copy `QGLViewer.dll` into the `bin` directory of LStudio.

Then, launch `CMake`. Select the source directory of VVE and a build directory outside the source directory. Validate and select a MinGW project in the list of project types `CMake` propose. The variable to setup are:

- `CMAKE_BUILD_TYPE` - set it to `Release`
- `CMAKE_INSTALL_PREFIX` - set it to `C:\VVE`
- `QGLVIEWER_DIR` - set it to `C:\VVE\include`
- `QGLVIEWER_LIBRARY_DIR` - set it to `C:\VVE\lib`

Click the `configure` button, then the `OK` button. The window should disappear without error message. Next, launch a window command prompt and change the directory for your build directory. If your build directory is `C:\vve_build`, then type:

```
$ cd C:\\vve_build
$ c:
$ mingw32-make
$ mingw32-make install
```

Note

the `$` sign indicate the prompt, you should not type it

At last, copy the `vveinterpreter.exe` file into your LStudio `bin` directory.

You are setup and can compile/run VVE models in LStudio.

5.3 Post-installation specific to L-Studio ATE

L-Studio ATE comes with its own version of MinGW. However, the version it comes with is too old to run VVE. Also, when L-Studio launches any command (i.e. to run or compile) it completely override the `PATH` the user set up and points to its own `MINGW`. You should prevent it by editing the file `L-studio-VERSION\lpfg\bin\mingw.bat` and replace its content by:

```
IF "%MINGW%" == "" set MINGW=%LPFGPATH%\MinGW
set PATH=%PATH%;%MINGW%\bin
set MINGWMAKE=%MINGW%\bin\mingw32-make
```


Chapter 6

VVE introduction

6.1 Table of content

- [Structure of a model](#)
- [Model compilation](#)
- [VVE constructs](#)
 - [The forall loop](#)
 - [Defining your own graph](#)
 - [Lookup methods](#)
 - [Edition methods](#)
- [Advanced user interaction with a Viewer](#)
 - [Creating a context menu](#)
 - [Selecting a 3D object](#)

6.2 Structure of a model

VVE provides a complete environment to create a model.

Each model consist in a single dynamically loaded library describing the model and being loaded at run time. This allows for quicker interaction, as you don't need to rerun the program each time but more simply reload the library anytime a modification is made.

Once the model is created and compiled as a library, the VVE interpreter will take control of it to run and render it and allow a certain number of basic user interactions.

A VVE model consists in up to two classes:

1. A mandatory [Model](#) class that describes the model and its representation;
2. An optional [Viewer](#) class that implements any advanced user interactions.

To run a model, the VVE interpreter creates a [Model](#) object, a [Viewer](#) object and link them (i.e. the [Viewer](#) object knows about the [Model](#) object).

Each model begins with the inclusion of the main VVE header:

```
#include <vve.h>
```

Then, the user typically defines the main model class, inheriting the [Model](#) class.

```
class MyModel : public Model  
{
```


The constructor of the model has to accept a **QObject** as argument and pass it to its base class. The constructor is also the place to initialize all the variables of the model.

```
MyModel(QObject* parent)
    : Model(parent)
{
    // Initialization of the model
}
```

For this trivial model, the only method that has to be provided is the [Model::step](#) method. It should describe the evolution of the model for one step.

```
void step()
{
    // Compute the evolution of the model
}

};
```

At last, the user has to tell the system which class is to be used as model.

```
DEFINE_MODEL(MyModel);
```

All the methods that can be overridden to specify the model are documented in the [Model](#) class.

This minimal model is the do not have any rendering (which does not mean you cannot have any output, just not any OpenGL output in the VVE interpreter). This is, however, the minimum valid model.

To have a better feel about the structure of the model, let's create a Hello World model.

So the beginning includes the VVE main header, creates the class, its constructor and the (empty) step method.

```
#include <vve.h>

class HelloWorldModel : public Model
{
    HelloWorldModel(QObject *parent)
        : Model(parent)
    { }

    void step()
    { }
```

Now, let's look at the draw method:

```
void draw( Viewer* viewer )
{
    viewer->drawText(5, 5, "Hello world!");
}
```

It uses the **QGLViewer** class, whose documentation can be found here:

<http://artis.imag.fr/Membres/Gilles.Debunne/QGLViewer/refManual/hierarchy.html>

It simply draws the text "Hello World!" at coordinates (5,5) in the OpenGL screen.

At last, the usual ending:

```
};

DEFINE_MODEL(HelloWorldModel);
```

6.3 Model compilation

6.3.1 Table of content

- [General intructions](#)
- [Adding dependencies](#)
- [Changing the model name](#)
- [Changing the compilation options](#)

6.3.2 General intructions

The compilation of the model is defined in the file called `Makefile`. The makefile starts with the inclusion of a compile-time generated makefile to allow for multi-plateform makefiles. Every makefile should contain:

```
include $(shell vveinterpreter --makefile)
```

A makefile containing only that line will be able to compile a model consisting of only one file called `model.cpp`.

Then, you can:

- compile `model.vve` with the command:
`run`
- run the model with the command:
`standalone`

- remove any compiled file using:
clean

Note

On windows, you might have to replace `make` by `mingw32-make` or `gmake` depending on your installation of MinGW.

6.3.3 Adding dependencies

You can add other files to compile by specifying the variable `EXTRA_SRCS`. This variable takes the name of the extra files **without their extension**. For example, if you want to add the files `bsurface.cpp` and `key_framer.cpp` to the compilation, your Makefile will be:

```
EXTRA_SRCS=bsurface key_framer
include $(shell vveinterpreter --makefile)
```

If the files `bsurface.h` or `key_framer.h` exist, they will automatically be added as dependencies for the compilation of the model.

You can also add header dependencies by modifying the variable `EXTRA_HEADERS`. For example, if you have a header `mytemplate.h` you want to add, you change your makefile to be:

```
EXTRA_HEADERS=mytemplate.h
include $(shell vveinterpreter --makefile)
```

Now, the compilation of your model will be conditioned to the modification of this file too.

6.3.4 Changing the model name

You might not want to call your model `model`. In that case, just define the variable `MODEL`:

```
MODEL=system
include $(shell vveinterpreter --makefile)
```

This will compile the file `system.cpp` into `system.vve`.

6.3.5 Changing the compilation options

By default, the compilation is performed using chosen optimized flags. This is great for running the model as fast as possible, but not for debugging. The options for debugging are prepared in a `CXXFLAGS_DEBUG` variable. To enable it you can redefine `CXXFLAGS` like this:

```
include $(shell vveinterpreter --makefile)
CXXFLAGS=$(CXXFLAGS_DEBUG)
```

You can also add options to the compilation or linking stage:

```
include $(shell vveinterpreter --makefile)
CXXFLAGS+=-WError
LD_SO_FLAGS+=-flag
LD_EXE_FLAGS+=-flag
```

This will add the "warnings as error" options to the compilation stage, and add some flag to the linking stage. The variable `LD_SO_FLAGS` is specific to the compilation of the dynamic library (i.e. shared object) and the `LD_EXE_FLAGS` to the compilation of the standalone model.

Note

adding options might make you makefile platform-dependant.

6.4 VVE constructs

VVE relies on a data structure called [VVGraph](#).

Most of the time a simulation will consists in editing one or more graphs. The graph and its related classes are defined in the graph namespace, which is made directly accessible by the `vve.h` header file (i.e. you don't need to use fully-qualified names to access the `graph::VVGraph`, `graph::Vertex` or `graph::Edge` classes). It also relies on a new loop construct: the `forall` loop, provided for the user's convenience.

6.4.1 The forall loop

First, VVE provides a macro to iterate on any STL container without dealing with the iterators. This macro is called `forall` and can be used like this:

```
std::vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
forall(int i, v)
{
    cout << i << " ";
}
```

This code will output "1 2 3 4".

Another way to iterate is by using references. This method allows you to modify the value you are iterating on:

```
std::vector<double> v;
for(int i = 0 ; i < 100 ; i++)
```

```

{
    v.push_back(i);
}
forall(double& d, v)
{
    d = sin(d*2*M_PI/100);
}

```

At the end of this code, `v` contains the values of \sin for $2k\pi/100$ with k an integer from 0 to 99.

The last useful construct with the `forall` loop is the iteration using constant references. This is useful to iterate without copying the objects but without modifying them either. Note that you can't iterate over STL sets using non-constant references!

```

std::set<double> s;
s.insert(20);
s.insert(10);
s.insert(100);
forall(const double& d, s)
{
    cout << d << " ";
}

```

This code will output "10 20 100 ".

The last construct will be very useful as the VVE graphs behaves almost like STL sets. Thus it will be most efficient to iterate over a graph using constant references:

```

vvgraph M;
// Filling in the graph
forall(const vertex& v, M)
{
    // Do something with the vertex
}

```

Note

As for any other loop, the `forall` loop do not keep the container iterated on alive. This means that, if a function returns a container by value, you have to store it in a local variable *before* iterating of it. Example:

If you have this function defined:

```
std::vector<int> build_vector();
```

And you try to iterate like this:

```

forall(int i, build_vector())
{
    // Do something
}

```

The result is undefined, as the vector created by `build_vector` do not exist after the evaluation of the function. The correct way to iterate on it is:

```
std::vector<int> v = build_vector();
forall(int i, v)
{
    // Do whatever
}
```

6.4.2 Defining your own graph

Before editing and exploring the structure of a graph, you have to define your own. A [graph::VVGraph](#) is a class template parametrized by the structure holding the label of vertexes and the structure holding the label of the edges. The usual first step is then to define a new type for your specialized graph:

```
typedef graph::VVGraph<MyVertexContent, MyEdgeContent> vvgraph;
```

Once you defined your own graph, you will want to create new types for the smart pointers for vertex and edge access. You have two ways of defining them: either directly using the [graph::Vertex](#) and [graph::Edge](#) templates, or using the types withing the graph you created. The recommended way (easier and less error-prone) is the second:

```
typedef vvgraph::vertex_t vertex;
typedef vvgraph::edge_t edge;
```

Now you have defined all the types you need to fully interact with your home-made graph.

Note

The types `vertex` and `edge` are smart pointers on the actual vertexes and edges. However they are very different smart pointers. First, a vertex can be created (and accessed) outside any graph. To reflect that property, the vertex type is a strong reference on the vertex data. This means that as long as you hold a vertex object, you can access it and the data it holds will be available, even if your removed that vertex from all existing graph. That means also you have to be careful not to create reference loops! The only way to have loop is to put a vertex object in the vertex data structure. If you avoid that (unusual) construct, you won't ever have references loops. Second, an edge do not exists outside a graph. Thus, the edge type is only a weak reference to the edge data. This means that, if you have an edge object and delete that edge from the graph, *you must not try to access that edge ever again*. Only the graph is responsible to allocating/deallocating edge data and you can't do anything to prevent the data from being destroyed if you delete an edge.

6.4.3 Lookup methods

Now, let's have a look on the methods provided to explore a graph. I will suppose in that section that all the types defined in the previous section (i.e. `vvgraph`, `vertex` and `edge`) are available.

6.4.3.1 Vertex lookup methods

First, the graph behaves exactly as a STL set of vertexes. So if you know to use a set, you know how to use graph (at least as far as vertexes are concerned). This means you can already iterate on it:

```
vvgraph M;
forall(const vertex& v, M) // Constant reference, because it behaves as a STL set
{
    // use v here
}
```

Now, if you want to use the vertex you iterate on, remember this is a smart pointer, so just act as if it was a pointer on the structure you defined:

```
struct MyVertexContent
{
    int a;
    double b;
};

typedef graph::VVGraph<MyVertexContent> vvgraph; // If the edge is not specified
an empty structure is used.
vvgraph M;

// ...

forall(const vertex& v, M)
{
    v->a = 1;
    v->b = 1.3;
}
```

Note that when iterating using constant references, the constant qualification applies to the smart pointer, not its content (this is due to the implementation of the [graph::Vertex](#) smart pointer, not a general C++ rule!). So you can freely modify the data of a vertex, even if holding a constant reference on a vertex. This is also true for the edges. Note, however, that there exists a constant edge smart pointer, that won't allow you to modify the edge. You will get one if you iterate over a constant graph. As a rule, remember that vertexes content are not considered part of the graph, so modifying them do not modify the graph itself. However, the edges *are* part of the graph, so modifying them modifies the graph.

If you just want one vertex, you can ask for one using the [graph::VVGraph::any](#) method:

```
vvgraph M;
if(!M.empty())
{
    vertex v = M.any();
    v->a = 0;
}
```

6.4.3.2 Neighbors lookup methods

The first basic lookup method for neighbors is the [graph::VVGraph::neighbors](#) method, that allows you to iterate over the neighbors of a vertex:

```
forall(const vertex& n, M.neighbors(v))
{
    n->a = v->a+1;
}
```

Once you get a neighbor of a vertex, you can also access the neighbors before and after it:

```
forall(const vertex& n, M.neighbors(v))
{
    vertex nn = M.nextTo(v, n);
    // nn is the neighbor of v after n
    vertex np = M.prevTo(v, n);
    // np is the neighbor of v before n
}
```

Two useful methods are [graph::VVGraph::valence](#), that gets the number of neighbors and [graph::VVGraph::anyIn](#) that returns a neighbor:

```
if(M.valence(v) > 0)
{
    vertex n = M.anyIn(v);
    n->a = v->a;
}
```

At last, it is possible to access the incoming edges, using the [graph::VVGraph::iNeighbors](#) method:

```
forall(const vertex& n, M.iNeighbors(v))
{
    M.edge(n, v)->a = 3; // Note that it is the edge n->v that exists, and not v->n
}
```

6.4.3.3 Edge lookup methods

The last category of lookups is for the edges. It is pretty simple as there are only three methods: [graph::VVGraph::edge](#), [graph::VVGraph::source](#) and [graph::VVGraph::target](#). They will allow you to interact with the edge objects:

```
vvgraph M;
if(!M.empty())
{
    vertex v = M.any();
    if(M.valence(v) > 0)
    {
        vertex n = M.anyIn(v);
        edge e = M.edge(v, n);
        assert(v == M.source(e));
        assert(n == M.target(e));
    }
}
```

6.4.4 Edition methods

A graph is constructed entirely using vertexes. The edge being accessed only if you need to label them. Therefore, there are only two categories of edition methods: vertexes and neighbors.

6.4.4.1 Vertex edition methods

You can add a vertex is using either the `graph::VVGraph::addVertex` method, which create and add a vertex, or the `graph::VVGraph::insert` graph, that insert an existing vertex in the graph. The recommended way is the use on `graph::VVGraph::insert`, rather than `graph::VVGraph::addVertex`. So you can start populating a graph like that:

```
vvgraph M;
for(int i = 0 ; i < 100 ; ++i)
{
    vertex v;    // Creates a new vertex object
    v->a = i;    // Initialize the vertex
    M.insert(v); // Insert the vertex in M
}
```

You can also remove a vertex from the graph using the `graph::VVGraph::erase` method. As a weird way to clear a graph you can do:

```
vvgraph M;
// ...
while(!M.empty())
{
    M.erase(M.any());
}
```

Note that removing a vertex from a graph remove all edges involving that vertex, incoming or outgoing.

At last, you can remove all vertexes at once, using the `graph::VVGraph::clear()` method:

```
vvgraph M;
M.clear();
```

6.4.4.2 Neighbors edition methods

Once you have at least two vertexes in a graph, you can start adding neighbors to them. Note that, to be valid in any neighborhood-related method, a vertex has to be first inserted into the graph.

The first edition method for the neighborhood is the `graph::VVGraph::insertIn` method, that allows you to add a vertex into the neighborhood of another one without specifying where:

```
vvgraph M;
vertex v1, v2;
M.insert(v1);
M.insert(v2);
M.insertIn(v1, v2); // Insert v2 in the neighborhood of v1
```

Once the neighborhood of `v1` is not empty anymore, you can use `graph::VVGraph::spliceAfter` and `graph::VVGraph::spliceBefore` to insert new neighbors at specific positions:

```
vertex v3, v4;
M.insert(v3);
M.insert(v4);
M.spliceBefore(v1, v2, v3); // Splice v3 in v1 before v2
M.spliceAfter(v1, v2, v4);  // Splice v4 in v1 after v2
```

In the end, the neighbors of v_1 are ordered as $v_3 < v_2 < v_4 < v_3$.

You can also remove an edge using the `graph::VVGraph::eraseEdge` method. For example, to remove all outgoing edges from a vertex you can do:

```
while(M.valence(v) != 0)
{
    M.eraseEdge(v, M.anyIn(v));
}
```

At last, you can clear all edges (incoming and outgoing) of a vertex using the `graph::VVGraph::clear(const vertex_t&)` method:

```
M.clear(v);
```

6.5 Advanced user interaction with a Viewer

VVE allows the user to inherit its `Viewer` class, extending the Qt widget as needed.

The first step to extend the `Viewer` is to create your own viewer class inheriting the VVE `Viewer`:

```
class MyViewer : public Viewer
{
public:
    MyViewer(QWidget* parent)
        : Viewer(parent)
    { }
};
```

and to tell VVE you want to use this viewer class:

```
DEFINE_VIEWER(MyViewer);
```

The viewer can be extended using all the regular Qt `QWidget` extensions and the `QGLViewer` ones. You will find the documentation:

- for Qt: <http://doc.trolltech.com/>
- for `QGLViewer`: <http://artis.imag.fr/Membres/Gilles.Debunne/QGLViewer/refM>

Here I will describe some common extensions.

6.5.1 Creating a context menu

Creating a context menu relies on the Qt `QContextMenuEvent`, `QMenu` and `QAction` classes. First, we need to include them:

```
#include <QContextMenuEvent>
#include <QAction>
#include <QMenu>
```

Then, we define our class, and add the `Q_OBJECT` macro as we will need to specify Qt slots:

```
#include <iostream>

using namespace std;

class MyViewer : public Viewer
{
    Q_OBJECT
```

The menu items should be defined as actions (or action groups if required):

```
    QAction *firstAct, *secondAct, *thirdAct;
```

And the actions should be created in the constructor and connected to the slots:

```
public:
    MyViewer(QWidget* parent)
        : Viewer(parent)
        , firstAct(new QAction("First action", this))
        , secondAct(new QAction("Second action", this))
        , thirdAct(new QAction("Third action", this))
    {
        connect(firstAct, SIGNAL(activated()), SLOT(firstActivated()));
        connect(secondAct, SIGNAL(activated()), SLOT(secondActivated()));
        connect(thirdAct, SIGNAL(activated()), SLOT(thirdActivated()));
    }
```

Now, the slots have to be defined:

```
public slots:
    void firstActivated()
    {
        cout << "First action selected" << endl;
    }

    void secondActivated()
    {
        cout << "Second action selected" << endl;
    }

    void thirdActivated()
    {
        cout << "Third action selected" << endl;
    }
```

Then, the `contextMenuEvent` method has to be overridden to create and show the menu:

```
protected:
    void contextMenuEvent(QContextMenuEvent* event)
```

```

{
    QMenu *popup = new QMenu(this);
    popup->addAction(firstAct);
    popup->addAction(secondAct);
    popup->addAction(thirdAct);
    popup->popup(event->globalPos());
}
};

```

At last, as we defined slots, we will need Qt to generate the moc file and we will need to include it. Also, we need to tell VVE we want to use this class as viewer:

```

#include "model.moc"

DEFINE_VIEWER(MyViewer);

```

The whole can be found in [context_menu.cpp](#)

Just don't forget to add the moc file as dependency in your project!

6.5.2 Selecting a 3D object

The 3D object selection uses both OpenGL naming system and the **QGLViewer** selection helpers. The example will draw a grid of 10 by 10 squares using a graph and color the square in gray. When a square is selected (i.e. by pressing Alt and clicking on the square) it is drawn in red. Reselecting the square draw it in gray again. You will find the full code in the file [select.cpp](#)

The first step is to define the data we need for our vertex. In our case, we need the position of the graph (in 3D) and its status (i.e. selected or not). The position will be a [util::Vector](#), so we need to include it.

```

#include <vve.h>
#include <util/vector.h>

typedef util::Vector<3, double> Point3d;

struct VertexContent
{
    VertexContent()
        : selected(false)
    {}

    Point3d pos;
    bool selected;
};

```

Note

The default constructor specifies the vertex is, by default, not selected. The [util::Vector](#) class specifies already a default constructor setting the coordinates to 0, so we don't need to respecify it.

Next, we define our graph and vertex types:

```
typedef graph::VVGraph<VertexContent> vvgraph;
typedef vvgraph::vertex_t vertex;
```

Then, the model class, its constructor (filling in the graph) and an empty step method (remember the step method is mandatory):

```
class SelectModel : public Model
{
public:
    vvgraph S;

    SelectModel(QObject *parent)
        : Model(parent)
    {
        for(double i = 0 ; i < 10 ; i++)
        {
            for(double j = 0 ; j < 10 ; j++)
            {
                vertex v; // Create a vertex
                v->pos.x() = i;
                v->pos.y() = j;
                S.insert(v);
            }
        }
    }

    void step() { }
```

Now, just for presentation purpose, we want to setup the size of our scene in the viewer, so that the camera will be nicely placed. We also want a black background.

```
void preDraw( Viewer* viewer )
{
    glClearColor(0,0,0,0);
    viewer->setSceneBoundingBox(qglviewer::Vec(0,0,0), qglviewer::Vec(10, 10, 0))
    ;
}
```

The next function just draw a square at the position of the vertex. It is extracted as we will need both to actually draw the square and to define the shape of the cells for the selection.

```
void drawCell(const vertex& v)
{
    Point3d c1 = v->pos;
    Point3d c2 = v->pos + Point3d(1, 0, 0);
    Point3d c3 = v->pos + Point3d(1, 1, 0);
    Point3d c4 = v->pos + Point3d(0, 1, 0);
    glBegin(GL_QUADS);
    glVertex3dv(c1.c_data());
    glVertex3dv(c2.c_data());
```

```

        glVertex3dv(c3.c_data());
        glVertex3dv(c4.c_data());
        glEnd();
    }

```

Now, the draw method draw the squares, setting the color to red or gray depending on the state:

```

void draw()
{
    forall(const vertex& v, S)
    {
        if(v->selected)
        {
            glColor3f(1, 0, 0);
        }
        else
        {
            glColor3f(0.5, 0.5, 0.5);
        }
        drawCell(v);
    }
}

```

And now, the most important method for the selection: [Model::drawWithNames](#). This method should draw all the selectable items, setting the name with `glPushName` and `glPopName`. To name the vertexes, the method `graph::Vertex::label` is very convenient, as you are guaranteed a unique name for each vertex of the same type (be careful though that vertexes of different type may have the same label).

```

void drawWithNames()
{
    forall(const vertex& v, S)
    {
        glPushName((int)v.label());
        drawCell(v);
        glPopName();
    }
}
};

```

The last step is to define the viewer and what to do with the selection. This is done by overriding the [Viewer::postSelection](#) method. We will also use a way to obtain a weak pointer on a vertex. If you hold a valid label of a vertex (and know its type), you can then construct a vertex using this label. Be careful though, as there is no checking of the validity of the label. Also, the reference you will hold is weak, so you must not store it!

```

class SelectViewer : public Viewer
{
public:
    SelectViewer(QWidget *parent)
        : Viewer(parent)
    { }
}

```

```
protected:
void postSelection(const QPoint& p)
{
    int s = selectedName();
    if(s != -1)
    {
        // Obtain a weak pointer on the vertex
        // Be careful, if s is not the label of a vertex, this could crash
        vertex v((vertex::identity_t)s);
        v->selected ^= true;
    }
}
};
```

And don't forget to declare the model and viewer:

```
DEFINE_MODEL(SelectModel);
DEFINE_VIEWER(SelectViewer);
```


Chapter 7

VVE libraries

Chapter 8

Deprecated List

Member **vvcomplex::FindCenter**(const typename VVComplex::cell &c, VVComplex &T)

You should use **vvcomplex::findCenter** instead and update yourself the position of the cell.

Chapter 9

Bug List

Member `geometry::pointInTriangle`(const Point2d &p, const Point2d &p1, const Point2d &p2, const Point2d &p3) Not working properly at all !

Member `util::Contour::Contour`(std::string filename) If the specified contains an incomplete or improper specification, the result of construction will be unknown. Most probably, the object will unuseable; however, this is not currently reported.

Member `util::Function::Function`(std::string filename) If the specified file contains an incomplete or improper specification, the result of construction will be unknown. Most probably, the object will unuseable; however, this is not currently reported.

Chapter 10

Directory Hierarchy

10.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

doc	60
examples	61
vvelib	71
algorithms	57
factory	62
geometry	63
graph	64
shape	65
storage	66
config	59
test	67
util	68

Chapter 11

Namespace Index

11.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

algorithms (Namespace containing various useful algorithms)	73
bspline_tissue_model (Namespace containing the base class for bspline tissue model helper)	79
cell_system (Namespace defining the cell system division algorithm and model)	80
complex_factory (Contains functions to initialise a complex)	86
factory (Contains predefined ways to initialize a graph)	102
geometry (Algorithmic geometry)	111
graph (Contains all the classes related to the graphs)	121
parallel (Define a thread pool for multi-threading computation)	124
shape (Define a set of shape to be drawn using OpenGL)	126
solver (Implement a generic solver of ODE on a graph)	129
std (STL namespace)	133
storage (Namespace containing all functions and classes related to VVE persistence)	134
template_utils (Namespace for meta-programming utils using templates) . . .	142
test (Namespace containing test facilities)	143
tissue (The tissue module handle cell division in a cell tissue)	144
tissue_model (Namespace containing the base class for the tissue model helper)	153
util (Various utility classes for generic programming)	154
util::Shapes (This namespace contains various shapes)	170
vvcomplex (Definition of classes and functions for 2d cell complexes)	171

Chapter 12

Class Index

12.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

graph::_EmptyEdgeContent	187
graph::Arc< EdgeContent >	188
util::Shapes::BSplineSurface	194
util::Buffer< T, size >	203
algorithms::TriangleSurface::Cell	205
tissue::CellPinchingParams	206
tissue::ClosestMidAlgoParams	238
tissue::ClosestWallAlgoParams	240
tissue::ShortWallAlgoParams	459
cell_system::CellSystemCell	227
cell_system::CellSystemDivisionParams	229
cell_system::CellSystemJunction	233
util::CircIterator< ForwardIterator >	235
bspline_tissue_model::TissueModel< RealModel, TissueClass >::CompareSize	248
tissue_model::TissueModel< RealModel, TissueClass >::CompareSize	249
storage::ConvertType< From, To >	263
graph::CountedContent< Content >	264
util::CrossProductType< 2, T >	265
util::CrossProductType< 3, T >	266
util::GraphInset::Data	267
vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunc- tionContent, JunctionCellContent, compact >::division_result_t . .	269
vvcomplex::DivisionData< JunctionContent >	270
cell_system::DivisionParams	276
graph::Edge< EdgeContent >	278
storage::Frame	294
util::GraphInset	307
complex_factory::HexFiller< Complex, Model >	327

vvcomplex::InModelDivisionParam	328
algorithms::Insert< vvgraph, do_checks >	329
util::Materials::Material	332
util::Matrix< nRows, nCols, T >	339
graph::VVGraph< VertexContent, EdgeContent, compact >::neighbor_t	387
complex_factory::ObjReaderError	390
ParallelControler	399
util::Parms	400
algorithms::TriangleSurface::Position	419
QGLViewer[external]	
Viewer	676
QObject[external]	
util::FileObject	289
Model	362
bspline_tissue_model::TissueModel< RealModel, TissueClass >	572
cell_system::CellSystem< Complex, MyModel >	208
cell_system::CellSystem< TissueClass, RealModel >	208
tissue_model::TissueModel< RealModel, TissueClass >	561
util::Contour	250
util::Function	295
util::KeyFramer	330
util::Materials	333
util::Palette	392
util::WatchDog	973
util::range< Iterator >	430
util::refpair< T, U >	435
graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >	438
graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >	442
util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >	447
util::set_vector< T, Equal, Alloc >	457
graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::single_neighborhood_t< VertexSrcContent, VertexTgtContent, EdgeSrcContent >	462
graph::VVGraph< VertexContent, EdgeContent, compact >::single_neighborhood_t	465
solver::Solver< nb_vars, identifier >	468
complex_factory::SquareFiller< Complex, Model >	504
util::Tensor< T >	505
util::Texture1D	508
util::Texture2D	514
parallel::ThreadEval	520
algorithms::TriangleGrowth	588
algorithms::TriangleSurface	604
storage::TypeTraits< T >	614
util::Vector< dim, T >	616
geometry::Quaternion	420

util::Vector< 3, T >	616
util::Point< T >	415
util::Vector< 4, T >	616
util::Color< T >	242
algorithms::TriangleSurface::Vertex	656
graph::Vertex< VertexContent >	657
graph::Vertex< VERTEX_ARGS >	657
graph::WeakVertex< VertexContent >	979
graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >	698
graph::VVBiGraph< COMPLEX_ARGS >	698
vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >	833
vvcomplex::VVComplexGraph< RESOLVE_LEAF_- CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_- ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >	833
vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallCon- tent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >	790
vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_- ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_- COMPLEX_TEMPLATE_ARGS >)>	790
tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >	521
storage::VVEStorage	853
storage::old::VVEStorage_BINReader	867
storage::VVEStorage_BINReader	875
storage::VVEStorage_BINWriter	885
storage::VVEStorage_XMLReader	892
storage::VVEStorage_XMLWriter	900
graph::VVGraph< VertexContent, EdgeContent, compact >	907
graph::VVGraph< JunctionContent, graph::_EmptyEdgeContent, false >	907
graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_- EmptyEdgeContent, false >	907

Chapter 13

Class Index

13.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

graph::_EmptyEdgeContent (Empty class used as default for edge content)	187
graph::Arc< EdgeContent > (Type of a undirected edge (or arc))	188
util::Shapes::BSplineSurface (Class describing a bspline surface)	194
util::Buffer< T, size > (A ranged checked array)	203
algorithms::TriangleSurface::Cell (Type of a cell, i.e)	205
tissue::CellPinchingParams (Parameters of the cell pinching algorithm)	206
cell_system::CellSystem< Complex, MyModel > (Full cell-system model)	208
cell_system::CellSystemCell (Content of a vertex for the 2D cell system)	227
cell_system::CellSystemDivisionParams (Parameters for cell-system division)	229
cell_system::CellSystemJunction (Content of a vertex for the 2D cell system)	233
util::CircIterator< ForwardIterator > (Creates a circular iterator from a range of forward iterators)	235
tissue::ClosestMidAlgoParams (Parameters for the closest mid)	238
tissue::ClosestWallAlgoParams (Parameters for the closest wall algorithm)	240
util::Color< T > (A utility class to encapsulate color data)	242
bspline_tissue_model::TissueModel< RealModel, TissueClass >::CompareSize (Operator class to compare the size of cells)	248
tissue_model::TissueModel< RealModel, TissueClass >::CompareSize (Operator class to compare the size of cells)	249
util::Contour (Contour utility class)	250
storage::ConvertType< From, To > (Convert object of type from to object of type to)	263
graph::CountedContent< Content > (Type of the reference counted content)	264
util::CrossProductType< 2, T > (For 2d -> a scalar)	265
util::CrossProductType< 3, T > (For 3d -> a vector)	266
util::GraphInset::Data (Data structure used to store the time stamped values)	267

vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::division_result_t (Describe the result of a cell division)	269
vvcomplex::DivisionData< JunctionContent > (Class holding the data needed to actually divide a cell)	270
cell_system::DivisionParams (Structure storing the right hand side of a division rule)	276
graph::Edge< EdgeContent > (Edge of a vv graph)	278
util::FileObject (This class is the base class for any object that might be watched by the WatchDog)	289
storage::Frame (Data holding properties of a frame)	294
util::Function (A utility class for functions)	295
util::GraphInset (Class used to draw a graph of time stamped data in the model window)	307
complex_factory::HexFiller< Complex, Model > (For internal use only!)	327
vvcomplex::InModelDivisionParam (Class used to define the division parameters from within the model class)	328
algorithms::Insert< vvgraph, do_checks > (Insert a new vertex on an edge)	329
util::KeyFramer (Class describing the evolution of a bspline surface through time)	330
util::Materials::Material (The material data structure)	332
util::Materials (A utility class for materials)	333
util::Matrix< nRows, nCols, T > (Class representing a fixed-size matrix)	339
Model (Simulation class)	362
graph::VVGraph< VertexContent, EdgeContent, compact >::neighbor_t (Structure maintaining the data for a single neighbor)	387
complex_factory::ObjReaderError (Error object returned by the complex_factory::objreader functions)	390
util::Palette (A utility class for palettes)	392
ParallelControler (This class is used to control how many threads are used for parallel processing)	399
util::Parms (A utility class to parse L-Studio like parameter files)	400
util::Point< T > (A 3D point/vector class)	415
algorithms::TriangleSurface::Position (Structure describing the position of a point on a triangulated surface, relatively to this surface)	419
geometry::Quaternion (Implements the quaternion operations)	420
util::range< Iterator > (Class representing a range of values from two iterators)	430
util::refpair< T, U > (Class used to hold references for the util::tie() function)	435
graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator > (Type of the result of the search for a vertex in a neighborhood)	438
graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content, compact >::search_result_t< Neighborhood, Iterator > (Type of the result of the search for a vertex in a neighborhood)	442
util::SelectMemberIterator< Iterator, T, member, Reference, Pointer > (Iterate over a container of structure, dereferencing only a member of it)	447

util::set_vector< T, Equal, Alloc > (Define a class using (small) vectors as unordered sets)	457
tissue::ShortWallAlgoParams (Parameters for the shortest wall algorithm) . .	459
graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::single_neighborhood_t< VertexSrcContent, VertexTgtContent, EdgeSrcContent > (Type of the neighborhood of a vertex)	462
graph::VVGraph< VertexContent, EdgeContent, compact >::single_neighborhood_t (Type of the neighborhood of a vertex)	465
solver::Solver< nb_vars, identifier > (Implement a set of solvers for ODE on a graph)	468
complex_factory::SquareFiller< Complex, Model > (For internal use only!)	504
util::Tensor< T > (A growth tensor class)	505
util::Texture1D (A utility class for one-dimensional textures)	508
util::Texture2D (A utility class for two-dimensional textures)	514
parallel::ThreadEval (Base class for function evaluated in a thread)	520
tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > (Class handling the development and representation of a cell tissue)	521
tissue_model::TissueModel< RealModel, TissueClass > (Base class for the tissue_model helper)	561
bspline_tissue_model::TissueModel< RealModel, TissueClass > (Base class for the bspline tissue model helper)	572
algorithms::TriangleGrowth (Growth description using a set of triangles moving)	588
algorithms::TriangleSurface (Class representing a triangulated surface) . . .	604
storage::TypeTraits< T > (Defines main traits for types directly handled) . .	614
util::Vector< dim, T > (Vector class supporting all classic classic vector operations)	616
algorithms::TriangleSurface::Vertex (Type of a vertex, i.e)	656
graph::Vertex< VertexContent > (Vertex of a vv graph)	657
Viewer (Viewer widget)	676
graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact > (Class representing a VV graph)	698
vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > (Class handling the representation and development of 2D cell complex)	790
vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact > (Definition of the bipartite graph with specialized methods for the complex graph) . . .	833
storage::VVEStorage (Abstract base class defining the interactions with the persistence module)	853
storage::old::VVEStorage_BINReader (Implementation reading an BIN file)	867
storage::VVEStorage_BINReader (Implementation reading an BIN file) . . .	875
storage::VVEStorage_BINWriter (Implementation writing an BIN file) . . .	885
storage::VVEStorage_XMLReader (Implementation reading an XML file) .	892
storage::VVEStorage_XMLWriter (Implementation writing an XML file) . .	900

[graph::VVGraph< VertexContent, EdgeContent, compact >](#) (Class representing a VV graph) 907
[util::WatchDog](#) (Watch the modifications of file objects for you) 973
[graph::WeakVertex< VertexContent >](#) (Weak pointer on a vertex) 979

Chapter 14

File Index

14.1 File List

Here is a list of all documented files with brief descriptions:

doc/ compilation.h	??
doc/ construct.h	??
doc/ CreateLib.h	??
doc/ Examples.h	??
doc/ gui.h	??
doc/ helpers.h	??
doc/ Install.h	??
doc/ intro.h	??
doc/ main_doc.h	??
doc/ VVelib.h	??
doc/examples/ context_menu.cpp (Example of a viewer that creates a context menu)	985
doc/examples/ hello_world.cpp (Simple Hello World example)	986
doc/examples/ select.cpp (Example with 3D object selection)	987
doc/examples/ simple_model.cpp (Minimal model file)	988
vvelib/ bspline_tissue_model.h (This files include the bspline tissue model helper)	1015
vvelib/ cellsystem_model.h	??
vvelib/ model.h (Defines the Model class)	1039
vvelib/ model_.cpp	??
vvelib/ seashell_model.h	??
vvelib/ tissue_model.h (This files include the tissue model helper)	1056
vvelib/ viewer.cpp	??
vvelib/ viewer.h	??
vvelib/ vve.h	??
vvelib/algorithms/ cellsystem.cpp	??
vvelib/algorithms/ cellsystem.h (Include the definitions necessary for the cell system division algorithm and model)	989
vvelib/algorithms/ complex.cpp	??

vvelib/algorithms/ complex.h (Definition of a 2D cell complex)	991
vvelib/algorithms/ draw_connections.h	??
vvelib/algorithms/ draw_graphs.h (This file contains algorithms to draw graphs)	1001
vvelib/algorithms/ graph.h (Implement common algorithms on graphs)	1002
vvelib/algorithms/ insert.h (Defines the algorithms::Insert class template) . . .	1005
vvelib/algorithms/ parallel.cpp	??
vvelib/algorithms/ parallel.h (Defines the help class for parallel execution) . .	1006
vvelib/algorithms/ parallel_impl.h	??
vvelib/algorithms/ solver.h (Defines the solver namespace)	1008
vvelib/algorithms/ split.h (Defines the algorithms::split function)	1010
vvelib/algorithms/ tissue.h (Definition of the tissue algorithm)	1011
vvelib/algorithms/ triangle_growth.cpp	??
vvelib/algorithms/ triangle_growth.h (This file contains the classes needed to describe the growth of a tissue from a triangular mesh growing through time)	1013
vvelib/factory/ complex_grid.h (Functions to initialise a complex as a regular grid)	1020
vvelib/factory/ grid.h (Contains factories to generate grids)	1022
vvelib/factory/ objreader.cpp	??
vvelib/factory/ objreader.h (Contains factories to generate a cell complex from an obj file)	1023
vvelib/geometry/ area.cpp	??
vvelib/geometry/ area.h (Algorithms for areas of various shapes)	1025
vvelib/geometry/ coordinates.cpp	??
vvelib/geometry/ coordinates.h (Includes functions to convert between different type of coordinate systems)	1026
vvelib/geometry/ geometry.h (Common definitions and utilities for all geometry algorithms)	1027
vvelib/geometry/ intersection.cpp	??
vvelib/geometry/ intersection.h (Algorithms to compute intersections)	1028
vvelib/geometry/ projection.cpp	??
vvelib/geometry/ projection.h (Algorithms to project on lines/planes/)	1030
vvelib/geometry/ quaternion.cpp	??
vvelib/geometry/ quaternion.h (Implements the quaternion object)	1031
vvelib/graph/ edge.h (Define the graph::Edge class to be used with the graph::VVGraph class)	1032
vvelib/graph/ vertex.cpp	??
vvelib/graph/ vertex.h (This file contain the definition of the graph::Vertex class)	1033
vvelib/graph/ vvbigraph.h (Contain the definition of the VVBiGraph template class for bipartite graphs)	1035
vvelib/graph/ vvgraph.cpp	??
vvelib/graph/ vvgraph.h (Contain the definition of the VVGraph template class)	1037
vvelib/shape/ polyhedron.cpp	??
vvelib/shape/ polyhedron.h	??
vvelib/shape/ quadric.cpp	??
vvelib/shape/ quadric.h (Include the definitions to draw spheres, cylinders and disks)	1040

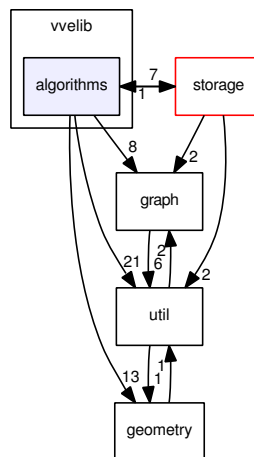
vvelib/storage/complex.h	1000
vvelib/storage/fwd.h	??
vvelib/storage/graph.h	1004
vvelib/storage/matrix.h	??
vvelib/storage/qglviewer.cpp	??
vvelib/storage/qglviewer.h	??
vvelib/storage/qt.h	??
vvelib/storage/stl.h	??
vvelib/storage/storage.cpp	??
vvelib/storage/storage.h (This file contains the base class and functions to handle storage)	1041
vvelib/storage/storage_bin.cpp	??
vvelib/storage/storage_bin.h	??
vvelib/storage/storage_xml.cpp	??
vvelib/storage/storage_xml.h (This file contains the BIN implementation of the storage class)	1043
vvelib/storage/types.cpp	??
vvelib/storage/types.h (This file contains all the utilities to convert from type to type)	1044
vvelib/storage/vector.h	1048
vvelib/storage/config/endianness.cpp	??
vvelib/storage/config/sizeof.cpp	??
vvelib/test/complex.vvh	??
vvelib/test/graph.vvh (File containing macros and functions to test properties on graphs)	1051
vvelib/test/test.cpp	??
vvelib/test/test.h (File containing macros and functions to for main testing)	1053
vvelib/util/assert.cpp	??
vvelib/util/assert.h (Graphical (or textual) assertion utility)	1060
vvelib/util/bsurface.cpp	??
vvelib/util/bsurface.h	??
vvelib/util/buffer.h (Defines the util::Buffer class)	1062
vvelib/util/circ_iterator.h	??
vvelib/util/clamp.h (Defines the util::clamp function)	1063
vvelib/util/color.cpp	??
vvelib/util/color.h (Defines the util::Color class template)	1064
vvelib/util/contour.cpp	??
vvelib/util/contour.h (Defines the util::Contour class)	1066
vvelib/util/dir.cpp	??
vvelib/util/dir.h (Defines functions related to directory handling)	1067
vvelib/util/features.h	??
vvelib/util/floats.h	??
vvelib/util/forall.h (This file contains the defines the forall loops)	1068
vvelib/util/function.cpp	??
vvelib/util/function.h (Defines the util::Function class)	1073
vvelib/util/gl.h	??
vvelib/util/glerrorcheck.h (Define the OPENGGL_ERROR_CHECK macro)	1074
vvelib/util/glexth.h	??
vvelib/util/glu.h	??
vvelib/util/graph_inset.cpp	??

vvelib/util/graph_inset.h (Defines the util::GraphInset class)	1075
vvelib/util/hashmap.h	??
vvelib/util/hashset.h	??
vvelib/util/key_framer.cpp	??
vvelib/util/key_framer.h	??
vvelib/util/leaf_class.h (Define templates and macros to figure out, at compile time, what class is the leaf class of a chain of inheritance)	1076
vvelib/util/mangling.cpp	??
vvelib/util/mangling.h	??
vvelib/util/materials.cpp	??
vvelib/util/materials.h (Defines the util::Materials class)	1077
vvelib/util/matrix.h	??
vvelib/util/matrix_impl.h	??
vvelib/util/member_iterator.h (Defines the util::SelectMemberIterator class template)	1078
vvelib/util/memory.h	??
vvelib/util/minmax.h	??
vvelib/util/palette.cpp	??
vvelib/util/palette.h (Defines the util::Palette class)	1079
vvelib/util/parms.cpp	??
vvelib/util/parms.h (Defines the util::Parms class)	1080
vvelib/util/point.h (Defines the (deprecated) util::Point class template)	1081
vvelib/util/random.cpp	??
vvelib/util/random.h (Defines various functions to generate random numbers)	1082
vvelib/util/range.h (Defines the range util template and various functions to deal with numerical range)	1084
vvelib/util/set_vector.h	??
vvelib/util/static_assert.h (Define the STATIC_ASSERT macro)	1086
vvelib/util/tensor.h (Defines the util::Tensor class template)	1088
vvelib/util/texture.cpp	??
vvelib/util/texture.h (Defines the util::Texture1D and util::Texture2D classes)	1089
vvelib/util/tie.h (Defines the util::tie function)	1090
vvelib/util/unorderedmap.h	??
vvelib/util/unorderedset.h	??
vvelib/util/vector.h (Defines the util::Vector class template)	1049
vvelib/util/watchdog.cpp	??
vvelib/util/watchdog.h (Defines the util::WatchDog and util::FileObject classes)	1091

Chapter 15

Directory Documentation

15.1 vvelib/algorithms/ Directory Reference



Files

- file **cellsystem.cpp**
- file [cellsystem.h](#)

Include the definitions necessary for the cell system division algorithm and model.

- file **complex.cpp**
- file [complex.h](#)

Definition of a 2D cell complex.

- file **draw_connections.h**
- file [draw_graphs.h](#)

This file contains algorithms to draw graphs.

- file [graph.h](#)

Implement common algorithms on graphs.

- file [insert.h](#)

Defines the [algorithms::Insert](#) class template.

- file **parallel.cpp**

- file [parallel.h](#)

Defines the [help](#) class for parallel execution.

- file **parallel_impl.h**

- file [solver.h](#)

Defines the solver namespace.

- file [split.h](#)

Defines the [algorithms::split](#) function.

- file [tissue.h](#)

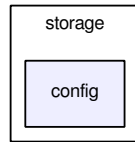
Definition of the tissue algorithm.

- file **triangle_growth.cpp**

- file [triangle_growth.h](#)

This file contains the classes needed to describe the growth of a tissue from a triangular mesh growing through time.

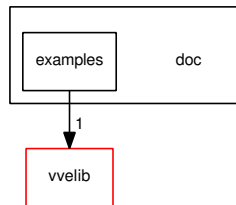
15.2 vvelib/storage/config/ Directory Reference



Files

- file **endianness.cpp**
- file **sizeof.cpp**

15.3 doc/ Directory Reference



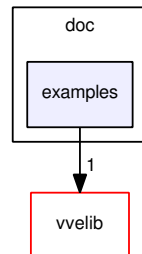
Directories

- directory [examples](#)

Files

- file **compilation.h**
- file **construct.h**
- file **CreateLib.h**
- file **Examples.h**
- file **gui.h**
- file **helpers.h**
- file **Install.h**
- file **intro.h**
- file **main_doc.h**
- file **VVELib.h**

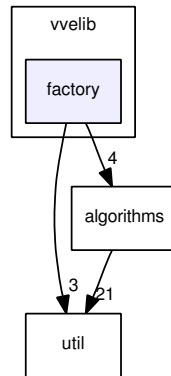
15.4 doc/examples/ Directory Reference



Files

- file [context_menu.cpp](#)
Example of a viewer that creates a context menu.
- file [hello_world.cpp](#)
Simple Hello World example.
- file [select.cpp](#)
Example with 3D object selection.
- file [simple_model.cpp](#)
Minimal model file.

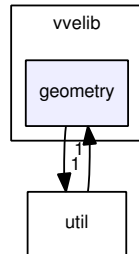
15.5 vvelib/factory/ Directory Reference



Files

- file [complex_grid.h](#)
Functions to initialise a complex as a regular grid.
- file [grid.h](#)
Contains factories to generate grids.
- file [objreader.cpp](#)
- file [objreader.h](#)
Contains factories to generate a cell complex from an obj file.

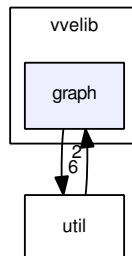
15.6 vvelib/geometry/ Directory Reference



Files

- file **area.cpp**
- file [area.h](#)
Algorithms for areas of various shapes.
- file **coordinates.cpp**
- file [coordinates.h](#)
Includes functions to convert between different type of coordinate systems.
- file [geometry.h](#)
Common definitions and utilities for all geometry algorithms.
- file **intersection.cpp**
- file [intersection.h](#)
Algorithms to compute intersections.
- file **projection.cpp**
- file [projection.h](#)
Algorithms to project on lines/planes/.
- file **quaternion.cpp**
- file [quaternion.h](#)
Implements the quaternion object.

15.7 vvelib/graph/ Directory Reference



Files

- file [edge.h](#)

Define the [graph::Edge](#) class to be used with the [graph::VVGra](#) class.

- file **vertex.cpp**
- file [vertex.h](#)

This file contain the definition of the [graph::Vertex](#) class.

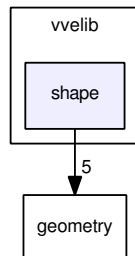
- file [vbigraph.h](#)

Contain the definition of the [VVB](#) template class for bipartite graphs.

- file **vvgraph.cpp**
- file [vvgraph.h](#)

Contain the definition of the [VVGra](#) template class.

15.8 vvelib/shape/ Directory Reference

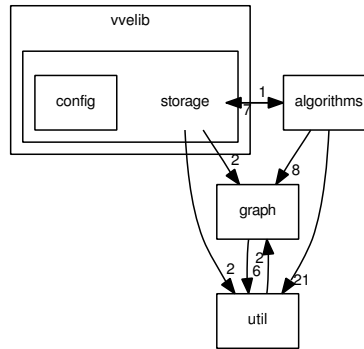


Files

- file **polyhedron.cpp**
- file **polyhedron.h**
- file **quadric.cpp**
- file [quadric.h](#)

Include the definitions to draw spheres, cylinders and disks.

15.9 vvelib/storage/ Directory Reference



Directories

- directory [config](#)

Files

- file [complex.h](#)
- file [fwd.h](#)
- file [graph.h](#)
- file [matrix.h](#)
- file [qglviewer.cpp](#)
- file [qglviewer.h](#)
- file [qt.h](#)
- file [stl.h](#)
- file [storage.cpp](#)
- file [storage.h](#)

This file contains the base class and functions to handle storage.

- file [storage_bin.cpp](#)
- file [storage_bin.h](#)
- file [storage_xml.cpp](#)
- file [storage_xml.h](#)

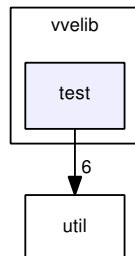
This file contains the BIN implementation of the storage class.

- file [types.cpp](#)
- file [types.h](#)

This file contains all the utilities to convert from type to type.

- file [vector.h](#)

15.10 vvelib/test/ Directory Reference



Files

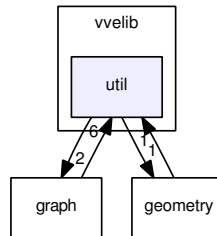
- file **complex.vvh**
- file [graph.vvh](#)

File containing macros and functions to test properties on graphs.

- file **test.cpp**
- file [test.h](#)

File containing macros and functions to for main testing.

15.11 vvelib/util/ Directory Reference



Files

- file **assert.cpp**
- file [assert.h](#)

Graphical (or textual) assertion utility.

- file **bsurface.cpp**
- file **bsurface.h**
- file [buffer.h](#)

Defines the [util::Buffer](#) class.

- file **circ_iterator.h**
- file [clamp.h](#)

Defines the [util::clamp](#) function.

- file **color.cpp**
- file [color.h](#)

Defines the [util::Color](#) class template.

- file **contour.cpp**
- file [contour.h](#)

Defines the [util::Contour](#) class.

- file **dir.cpp**
- file [dir.h](#)

Defines functions related to directory handling.

- file **features.h**
- file **floats.h**
- file [forall.h](#)

This file contains the defines the forall loops.

- file **function.cpp**

- file [function.h](#)

Defines the `util::Function` class.

- file `gl.h`
- file [glerrorcheck.h](#)

Define the `OPENGGL_ERROR_CHECK` macro.

- file `glext.h`
- file `glu.h`
- file `graph_inset.cpp`
- file [graph_inset.h](#)

Defines the `util::GraphInset` class.

- file `hashmap.h`
- file `hashset.h`
- file `key_framer.cpp`
- file `key_framer.h`
- file [leaf_class.h](#)

Define templates and macros to figure out, at compile time, what class is the leaf class of a chain of inheritance.

- file `mangling.cpp`
- file `mangling.h`
- file `materials.cpp`
- file [materials.h](#)

Defines the `util::Materials` class.

- file `matrix.h`
- file `matrix_impl.h`
- file [member_iterator.h](#)

Defines the `util::SelectMemberIterator` class template.

- file `memory.h`
- file `minmax.h`
- file `palette.cpp`
- file [palette.h](#)

Defines the `util::Palette` class.

- file `parms.cpp`
- file [parms.h](#)

Defines the `util::Parms` class.

- file [point.h](#)

Defines the (deprecated) `util::Point` class template.

- file `random.cpp`

- file [random.h](#)

Defines various functions to generate random numbers.

- file [range.h](#)

Defines the range util template and various functions to deal with numerical range.

- file **set_vector.h**

- file [static_assert.h](#)

Define the `STATIC_ASSERT` macro.

- file [tensor.h](#)

Defines the `util::Tensor` class template.

- file **texture.cpp**

- file [texture.h](#)

Defines the `util::Texture1D` and `util::Texture2D` classes.

- file [tie.h](#)

Defines the `util::tie` function.

- file **unorderedmap.h**

- file **unorderedset.h**

- file [vector.h](#)

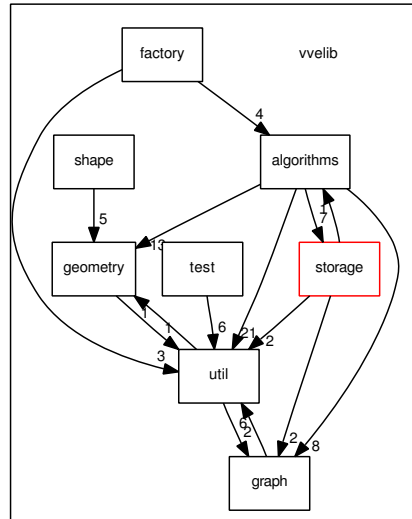
Defines the `util::Vector` class template.

- file **watchdog.cpp**

- file [watchdog.h](#)

Defines the `util::WatchDog` and `util::FileObject` classes.

15.12 vvelib/ Directory Reference



Directories

- directory [algorithms](#)
- directory [factory](#)
- directory [geometry](#)
- directory [graph](#)
- directory [shape](#)
- directory [storage](#)
- directory [test](#)
- directory [util](#)

Files

- file [bspline_tissue_model.h](#)
This files include the bspline [tissue](#) model helper.
- file [cellsystem_model.h](#)
- file [model.h](#)
Defines the [Model](#) class.
- file [model_.cpp](#)
- file [seashell_model.h](#)
- file [tissue_model.h](#)
This files include the tissue model helper.

- file **viewer.cpp**
- file **viewer.h**
- file **vve.h**

Chapter 16

Namespace Documentation

16.1 algorithms Namespace Reference

Namespace containing various useful algorithms.

Classes

- class [Insert](#)
Insert a new vertex on an edge.
- class [TriangleGrowth](#)
Growth description using a set of triangles moving.
- class [TriangleSurface](#)
Class representing a triangulated surface.

Functions

- `template<typename src_t , typename tgt_t , typename Model , typename Graph , typename iterator >
void _drawGraphConnections (Model *model, Viewer *viewer, const Graph &G, iterator(Graph::*begin)() const, iterator(Graph::*end)() const, bool order_neighbors)`
- `template<typename GraphType , typename Model >
void drawSymetricVVGra (Model *model, const GraphType &G)
This function draws a graph using lines between the vertices.`
- `template<typename Model , typename GraphType >
void drawVVBIGra (Model *model, Viewer *viewer, const GraphType &G, bool order_neighbors)`

Draw the connections of a bigraph using the drawArrow method of the viewer.

- template<typename Model , typename GraphType >
void [drawVVBigraphConnections2](#) ([Model](#) *model, [Viewer](#) *viewer, const GraphType &G, bool order_neighbors)

Draw the connections of a bigraph using the drawArrow method of the viewer.

- template<typename Model , typename GraphType >
void [drawVVGraohConnections](#) ([Model](#) *model, [Viewer](#) *viewer, const GraphType &G, bool order_neighbors)

Draw the connections of a graph using the drawArrow method of the viewer.

- template<class VertexContent , class EdgeContent , bool compact>
const [graph::Vertex](#)< VertexContent > & [insert](#) (const [graph::Vertex](#)< VertexContent > &a, const [graph::Vertex](#)< VertexContent > &b, [graph::VVGraph](#)< VertexContent, EdgeContent, compact > &S, const [graph::Vertex](#)< VertexContent > &x=[graph::Vertex](#)< VertexContent >(0))
- template<class Graph >
[Graph::edge_t](#) [insertAfter](#) (const typename [Graph::vertex_t](#) &v, const typename [Graph::vertex_t](#) &ref, const typename [Graph::vertex_t](#) &nv, [Graph](#) &S)

Splice nv after ref in v if ref is not null.

- template<class Graph >
[Graph::edge_t](#) [insertBefore](#) (const typename [Graph::vertex_t](#) &v, const typename [Graph::vertex_t](#) &ref, const typename [Graph::vertex_t](#) &nv, [Graph](#) &S)

Splice nv before ref in v if ref is not null.

- template<typename VertexContent , typename EdgeContent , bool compact, typename Model , typename tag_t >
void [shortest_paths_Dijkstra](#) (const std::unordered_set< [Vertex](#)< VertexContent > > &srcs, const [graph::VVGraph](#)< VertexContent, EdgeContent, compact > &m, [Model](#) &model, const tag_t &t)
- template<typename VertexContent , typename EdgeContent , bool compact, typename Model >
void [shortest_paths_Dijkstra](#) (const [Vertex](#)< VertexContent > &src, const [graph::VVGraph](#)< VertexContent, EdgeContent, compact > &m, const [Model](#) &model)
- template<typename VertexContent , typename EdgeContent , bool compact, typename Model , typename tag_t >
void [shortest_paths_Dijkstra](#) (const [Vertex](#)< VertexContent > &src, const [graph::VVGraph](#)< VertexContent, EdgeContent, compact > &m, [Model](#) &model, const tag_t &t)
- template<typename VVGraph , typename Model >
void [shortest_paths_FloydWarshall](#) (const [VVGraph](#) &m, const [Model](#) &model)

Compute the shortest path for all pair of vertices in the graph.

- template<typename VertexContent , typename EdgeContent , bool compact, typename Model , typename tag_t >

void **shortest_paths_FloydWarshall** (const [graph::VVGraph](#)< VertexContent, EdgeContent, compact > &m, [Model](#) &model, const tag_t &t)

- template<class vvgraph >
[vvgraph::vertex_t split](#) (const typename [vvgraph::vertex_t](#) &v, const typename [vvgraph::vertex_t](#) &n1, const typename [vvgraph::vertex_t](#) &n2, [vvgraph](#) &S)

Split the vertex v in two, attaching all the neighbors of v from n1 to n2 (included) to the new vertex.

16.1.1 Detailed Description

Namespace containing various useful algorithms.

16.1.2 Function Documentation

16.1.2.1 template<typename GraphType , typename Model > void algorithms::drawSymetricVVGraph (Model * model, const GraphType & G) [inline]

This function draws a graph using lines between the vertices.

It supposes the graph is symetric, i.e. if there is an edge from v1 to v2, there is another edge from v2 to v1.

These methods are required in the model:

```
Point3d position(const vertex& v);
Point3d normal(const vertex& v);
```

Definition at line 28 of file draw_graphs.h.

References [util::Vector< dim, T >::c_data\(\)](#), and [forall](#).

```
29  {
30      typedef typename GraphType::vertex_t vertex;
31      glBegin(GL_LINES);
32      forall(const vertex& v, G)
33      {
34          const geometry::Point3d& vpos = model->position(v);
35          const geometry::Point3d& normal = model->normal(v);
36          forall(const vertex& n, G.neighbors(v))
37          {
38              if (v<n)
39              {
40                  glNormal3dv(normal.c_data());
41                  glVertex3dv(vpos.c_data());
42                  glNormal3dv(model->normal(n).c_data());
43                  glVertex3dv(model->position(n).c_data());
44              }
45          }
46      }
47      glEnd();
48  }
```

16.1.2.2 **template<typename Model , typename GraphType > void algorithms::drawVVBiGraphConnections1 (Model * *model*, Viewer * *viewer*, const GraphType & *G*, bool *order_neighbors*) [inline]**

Draw the connections of a bigraph using the drawArrow method of the viewer.

This function draws the connections from the vertex1 to the vertex2.

This method is required in the model:

```
Point3d position(const vertex& v);
```

Definition at line 83 of file draw_connections.h.

```
87  {
88      typedef typename GraphType::vertex1_t vertex1;
89      typedef typename GraphType::vertex2_t vertex2;
```

16.1.2.3 **template<typename Model , typename GraphType > void algorithms::drawVVBiGraphConnections2 (Model * *model*, Viewer * *viewer*, const GraphType & *G*, bool *order_neighbors*) [inline]**

Draw the connections of a bigraph using the drawArrow method of the viewer.

This function draws the connections from the vertex2 to the vertex1.

This method is required in the model:

```
Point3d position(const vertex& v);
```

Definition at line 102 of file draw_connections.h.

```
106  {
107      typedef typename GraphType::vertex1_t vertex1;
108      typedef typename GraphType::vertex2_t vertex2;
```

16.1.2.4 **template<typename Model , typename GraphType > void algorithms::drawVVGraohConnections (Model * *model*, Viewer * *viewer*, const GraphType & *G*, bool *order_neighbors*) [inline]**

Draw the connections of a graph using the drawArrow method of the viewer.

This method is required in the model:

```
Point3d position(const vertex& v);
```

Definition at line 65 of file draw_connections.h.

```
69  {
70      typedef typename GraphType::vertex_t vertex;
```

16.1.2.5 `template<class Graph > Graph::edge_t algorithms::insertAfter (const
typename Graph::vertex_t & v, const typename Graph::vertex_t &
ref, const typename Graph::vertex_t & nv, Graph & S) [inline]`

Splice `nv` after `ref` in `v` if `ref` is not null.

Otherwise just insert the edge (`v`,`nv`) in `S`.

Definition at line 138 of file `insert.h`.

Referenced by `complex_factory::square_grid()`.

```
142 {
143     if(ref)
144         return S.spliceAfter(v, ref, nv);
145     return S.insertEdge(v, nv);
146 }
```

16.1.2.6 `template<class Graph > Graph::edge_t algorithms::insertBefore
(const typename Graph::vertex_t & v, const typename
Graph::vertex_t & ref, const typename Graph::vertex_t & nv, Graph
& S) [inline]`

Splice `nv` before `ref` in `v` if `ref` is not null.

Otherwise just insert the edge (`v`,`nv`) in `S`.

Definition at line 153 of file `insert.h`.

```
157 {
158     if(ref)
159         return S.spliceBefore(v, ref, nv);
160     return S.insertEdge(v, nv);
161 }
```

16.1.2.7 `template<typename VVGraph , typename Model > void
algorithms::shortest_paths_FloydWarshall (const VVGraph & m,
const Model & model) [inline]`

Compute the shortest path for all pair of vertices in the graph.

This version do not have a tag and will call all the required methods in the model without a tag (to use either as a default or if there is no other call to this function)

See also

`shortest_paths_FloydWarshall(const graph::VVGraph<VertexContent,EdgeContent>&,
Model&, const tag_t&)`

Definition at line 194 of file `graph.h`.

```
198 {
```

16.1.2.8 `template<class vvgraph > vvgraph::vertex_t algorithms::split (const typename vvgraph::vertex_t & v, const typename vvgraph::vertex_t & n1, const typename vvgraph::vertex_t & n2, vvgraph & S) [inline]`

Split the vertex `v` in two, attaching all the neighbors of `v` from `n1` to `n2` (included) to the new vertex.

The next vertex is replacing the range `n1-n2` in `v`.

All the edges content are copied using the `=` operator. This function assumes symmetric connectivity.

Definition at line 22 of file `split.h`.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::edge()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::eraseEdge()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insert()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdge()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::nextTo()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::replace()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::spliceAfter()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::spliceBefore()`.

```

26  {
27      typedef typename vvgraph::vertex_t vertex;
28      typedef typename vvgraph::edge_t edge;
29      vertex x;
30      vertex pv = v;
31      S.insert(x);
32      S.spliceBefore(v,n1,x);
33      S.insertEdge(x,v);
34      do
35      {
36          const vertex& n = S.nextTo(v,x);
37          typename edge::content_t ec = *S.edge(n,v);
38          *S.replace(n,v,x) = ec;
39          *S.spliceAfter(x,pv,n) = *S.edge(v,n);
40          pv = n;
41          S.eraseEdge(v,n);
42      } while(pv != n2);
43      return x;
44  }
```

16.2 bspline_tissue_model Namespace Reference

Namespace containing the base class for bspline tissue model helper.

Classes

- class [TissueModel](#)
Base class for the bspline tissue model helper.

Variables

- const double [epsilon](#) = 0.0001
Relative error for geometry.

16.2.1 Detailed Description

Namespace containing the base class for bspline tissue model helper.

16.2.2 Variable Documentation

16.2.2.1 const double bspline_tissue_model::epsilon = 0.0001

Relative error for geometry.

Definition at line 205 of file bspline_tissue_model.h.

Referenced by `tissue::findDivisionPoints()`, and `bspline_tissue_model::TissueModel<RealModel, TissueClass >::updateCellsArea()`.

16.3 cell_system Namespace Reference

Namespace defining the cell system division algorithm and model.

Classes

- class [CellSystem](#)
Full cell-system model.
- class [CellSystemCell](#)
Content of a vertex for the 2D cell system.
- class [CellSystemDivisionParams](#)
Parameters for cell-system division.
- class [CellSystemJunction](#)
Content of a vertex for the 2D cell system.
- class [DivisionParams](#)
Structure storing the right hand side of a division rule.

Typedefs

- typedef int [label_t](#)
Type of the label of a cell (for full cell-system only).

Functions

- template<class Complex >
[DivisionData](#)< typename Complex::junction_content > [findDivisionPoints](#)
(const typename Complex::cell &c, Complex &T, const [CellSystemDivisionParams](#) ¶ms)
Division algorithm for cell-system.
- template<class Complex >
[DivisionData](#)< typename Complex::junction_content > [findDivisionPoints](#)
(const typename Complex::cell &c, Point3d ¢er, Complex &T, const Point3d &direction)
Find the division points for a wall going through a given point.
- std::string [removeWhitespace](#) (std::string s)
Convenience function that removes white spaces before and after the string.

- `template<class Complex >`
`double volumeLeftCell (const typename Complex::cell &c, Complex &T, const Point3d ¢er, const DivisionData< typename Complex::junction_content > &result)`

Compute the volume of the left cell of a division.

16.3.1 Detailed Description

Namespace defining the cell system division algorithm and model.

16.3.2 Typedef Documentation

16.3.2.1 `typedef int cell_system::label_t`

Type of the label of a cell (for full cell-system only).

Definition at line 304 of file `cellsystem.h`.

16.3.3 Function Documentation

16.3.3.1 `template<class Complex > DivisionData<typename Complex::junction_content> cell_system::findDivisionPoints (const typename Complex::cell & c, Complex & T, const CellSystemDivisionParams & params) [inline]`

Division algorithm for cell-system.

See also

[CellSystemDivisionParams](#)

Definition at line 213 of file `cellsystem.h`.

References `cell_system::CellSystemDivisionParams::angle`, `tissue::cellPinching()`, `cell_system::CellSystemDivisionParams::direction`, `cell_system::CellSystemDivisionParams::epsilon`, `findDivisionPoints()`, `forall`, `IMPORT_COMPLEX_MODEL`, `IMPORT_COMPLEX_VERTICES`, `cell_system::CellSystemDivisionParams::minCellWall`, `cell_system::CellSystemDivisionParams::pinching`, `cell_system::CellSystemDivisionParams::pinchingParam`, `cell_system::CellSystemDivisionParams::ratio`, `vvcomplex::testDivisionOnVertices()`, and `volumeLeftCell()`.

```
215 {
216     IMPORT_COMPLEX_VERTICES(Complex);
217     IMPORT_COMPLEX_MODEL(Complex, T);
218     const Point3d& cell_normal = model.normal(c);
219     Point3d direction = util::Matrix<3,3,double>::rotation(cell_normal, params.an
```

```

        gle) * params.direction;
220 //      cout << "Z = " << params.direction << " - angle = " << params.angle << " =>
        direction = " << direction << endl;
221 //      cout << "Polygon: ";
222 //      forall(const vertex& j, S.neighbors(cell))
223 //      {
224 //          cout << model.vertexPosition(j) << "; ";
225 //      }
226 //      cout << endl;
227      Point3d center = model.position(c);
228      // First, find the division points going through the center
229      DivisionData<typename Complex::junction_content> result =
230          findDivisionPoints(c, center, T, direction);
231      if(!result)
232      {
233          return DivisionData<typename Complex::junction_content>();
234      }
235
236      // Then, if ratio != 0 then adjust volume by dichotomy
237      if(params.ratio)
238      {
239          // Volume of the cell
240          double total_volume = model.area(c);
241          double target_volume = total_volume*params.ratio;
242          double vol_epsilon = params.epsilon * target_volume;
243          double proj_min = HUGE_VAL, proj_max = -HUGE_VAL;
244          Point3d pos_max, pos_min;
245          forall(const junction& j, T.S.neighbors(c))
246          {
247              const Point3d& jpos = model.position(j);
248              double proj = (jpos - center) * direction;
249              if(proj > proj_max)
250              {
251                  proj_max = proj;
252                  pos_max = jpos;
253              }
254              if(proj < proj_min)
255              {
256                  proj_min = proj;
257                  pos_min = jpos;
258              }
259          }
260          double current_volume = volumeLeftCell(c, T, center, result);
261          Point3d delta;
262          if(current_volume < target_volume)
263          {
264              delta = proj_max*direction;
265          }
266          else
267          {
268              delta = -proj_min*direction;
269          }
270          cout << endl << endl;
271          cout << "Total volume = " << total_volume << " => target_volume = " << ta
rget_volume << endl;
272          cout << "Pos max = " << pos_max << " - Pos min = " << pos_min << endl;
273          cout << "delta = " << delta << endl;
274          while(std::abs(current_volume - target_volume) > vol_epsilon)
275          {
276              delta /= 2;
277              if(current_volume < target_volume)
278              {

```



```

279         center += delta;
280     }
281     else
282     {
283         center -= delta;
284     }
285     //     cout << "New delta = " << delta << " - new center = " << center <<
286     //     endl;
287     result = findDivisionPoints(c, center, T, direction);
288     current_volume = volumeLeftCell(c, T, center, result);
289     //     cout << "pu = " << result.pu << " - pv = " << result.pv << endl;
290 }
291 //     cout << "Volume = " << current_volume << endl;
292 }
293 if(params.pinchng)
294 {
295     tissue::cellPinching(c, T, result, params.pinchngParam);
296 }
297 testDivisionOnVertices(c, result, T, std::max(params.epsilon, params.minCellWa
298 ll));
299 return result;
300 }

```

16.3.3.2 template<class Complex > DivisionData<typename Complex::junction_content> cell_system::findDivisionPoints (const typename Complex::cell & c, Point3d & center, Complex & T, const Point3d & direction) [inline]

Find the division points for a wall going through a given point.

Parameters

- ← *c* Cell to be divided
- *center* Point defining the line of the division wall. At the end, center is the center of the division wall.
- ← *T* Complex containing the cell to divide
- ← *direction* Normal to the division wall

The division wall is a segment on the line going through *center* and with *direction* as normal.

Definition at line 127 of file cellsystem.h.

References forall, IMPORT_COMPLEX_MODEL, IMPORT_COMPLEX_VERTICES, geometry::planeLineIntersection(), vvcomplex::DivisionData< JunctionContent >::pu, vvcomplex::DivisionData< JunctionContent >::pv, vvcomplex::DivisionData< JunctionContent >::u1, and vvcomplex::DivisionData< JunctionContent >::v1.

Referenced by findDivisionPoints().

```

129 {
130     IMPORT_COMPLEX_VERTICES(Complex);
131     IMPORT_COMPLEX_MODEL(Complex, T);

```

```

132     DivisionData<typename Complex::junction_content> result;
133     //      cout << "Finding points for center: " << center << endl;
134     forall(const junction& v, T.S.neighbors(c))
135     {
136         const junction& n = T.S.nextTo(c, v);
137         const Point3d& p1 = model.position(v);
138         const Point3d& p2 = model.position(n);
139         Point3d u;
140         double s;
141         if(geometry::planeLineIntersection(u, s, center, direction, p1, p2) && s >=
142             0 && s <= 1)
143         {
144             //      cout << "Intersection between " << p1 << " and " << p2 << " at " << u <
145             //      < " with s = " << s << endl;
146             if((p1-center)*direction > 0)
147             {
148                 //      cout << "Found v1 at " << model.vertexPosition(v) << " with pv = " <<
149                 //      u << endl;
150                 if(result.v1)
151                 {
152                     cout << "Warning, finding v1 again" << endl;
153                 }
154                 result.v1 = v;
155                 result.pv = u;
156             }
157             else
158             {
159                 //      cout << "Found u1 at " << model.vertexPosition(v) << " with pu = " <<
160                 //      u << endl;
161                 if(result.u1)
162                 {
163                     cout << "Warning, finding v1 again" << endl;
164                 }
165                 result.u1 = v;
166                 result.pu = u;
167             }
168             if(result.v1 && result.u1)
169             {
170                 break;
171             }
172             if(result.u1 && result.v1)
173             {
174                 center = (result.pu+result.pv)/2;
175             }
176         }
177     }
178     return result;
179 }

```

16.3.3.3 `std::string cell_system::removeWhitespace (std::string s)`

Convenience function that removes white spaces before and after the string.

16.3.3.4 `template<class Complex > double cell_system::volumeLeftCell (const typename Complex::cell & c, Complex & T, const Point3d & center, const DivisionData< typename Complex::junction_content > & result) [inline]`

Compute the volume of the left cell of a division.

Do so with just the division data.

Parameters

c Cell being divided
T Complex containing the cell *c*
center A point on the division wall
result Division data being used
model [Model](#) is use

Definition at line 187 of file cellsystem.h.

References [IMPORT_COMPLEX_MODEL](#), [IMPORT_COMPLEX_VERTICES](#), [vvcomplex::DivisionData< JunctionContent >::pu](#), [vvcomplex::DivisionData< JunctionContent >::pv](#), [geometry::triangleArea\(\)](#), [vvcomplex::DivisionData< JunctionContent >::u1](#), and [vvcomplex::DivisionData< JunctionContent >::v1](#).

Referenced by [findDivisionPoints\(\)](#).

```

189  {
190      IMPORT_COMPLEX_VERTICES(Complex);
191      IMPORT_COMPLEX_MODEL(Complex, T);
192      double area = 0;
193      junction v = result.v1;
194      Point3d prev = result.pv;
195      do
196      {
197          v = T.S.nextTo(c, v);
198          const Point3d& cur = model.position(v);
199          area += geometry::triangleArea(center, prev, cur);
200          prev = cur;
201      } while(v != result.u1);
202      area += geometry::triangleArea(center, prev, result.pu);
203      return area;
204  }
```

16.4 complex_factory Namespace Reference

Contains functions to initialise a complex.

Classes

- struct [HexFiller](#)
For internal use only!
- class [ObjReaderError](#)
Error object returned by the [complex_factory::objreader](#) functions.
- struct [SquareFiller](#)
For internal use only!

Typedefs

- typedef [util::Vector](#)< 2, size_t > **Point2u**
- typedef [util::Vector](#)< 3, double > **Point3d**

Functions

- template<class Graph >
void [connectJunction](#) (const typename Graph::vertex_t &j, const typename Graph::vertex_t &nj, const typename Graph::vertex_t &pj, Graph &S, bool direct=true)
Connect j to nj knowing that p j is the junction before j in the current cell.
- template<typename Complex , class Model >
bool [hex_grid](#) (size_t N, size_t M, Complex &T, [Model](#) &model, const [Point3d](#) &bottom_left=[Point3d](#)(0, 0, 0), const [Point3d](#) &u=[Point3d](#)(1, 0, 0), const [Point3d](#) &v=[Point3d](#)(0, 1, 0))
Create a hexagonal grid with N lines and M columns.
- template<class CellComplex , class Model >
[ObjReaderError](#) [objreader](#) (std::string filename, CellComplex &T, [Model](#) &model)
Convenience methods that read a Wavefront OBJ file.
- template<class CellComplex , class Model >
[ObjReaderError](#) [objreader](#) (const **QString** &filename, CellComplex &T, [Model](#) &model)
Convenience methods that read a Wavefront OBJ file.

- `template<class CellComplex, class Model >`
`ObjReaderError objreader (QTextStream &content, CellComplex &T, Model &model)`
Read a Wavefront OBJ file describing a surface and return the complex representing it.
- `template<typename Complex, class Model >`
`bool square_grid (size_t N, size_t M, Complex &T, Model &model, const Point3d &bottom_left=Point3d(0, 0, 0), const Point3d &shift_right=Point3d(1, 0, 0), const Point3d &shift_up=Point3d(0, 1, 0))`
Create a square grid with N lines and M columns.
- `template<typename Complex, class Model >`
`SquareFiller< Complex, Model > squareFiller (const Point3d &bottom_left, const Point3d &shift_right, const Point3d &shift_up, Complex &T, Model &model)`
Create a square filler.

16.4.1 Detailed Description

Contains functions to initialise a complex.

16.4.2 Function Documentation

16.4.2.1 `template<class Graph > void complex_factory::connectJunction (const typename Graph::vertex_t &j, const typename Graph::vertex_t &nj, const typename Graph::vertex_t &pj, Graph &S, bool direct = true) [inline]`

Connect `j` to `nj` knowing that `pj` is the junction before `j` in the current cell.

Note

For internal use only!

Definition at line 275 of file `complex_grid.h`.

Referenced by `hex_grid()`.

```

280     {
281         typedef typename Graph::vertex_t vertex;
282         if (S.empty(j))
283         {
284             S.insertEdge(j, nj);
285             return;
286         }
287         if (S.edge(j, pj))
288         {
289             if (direct)

```

```

290         S.spliceBefore(j, pj, nj);
291     else
292         S.spliceAfter(j, pj, nj);
293     return;
294 }
295 const vertex& v = S.anyIn(j);
296 S.spliceBefore(j, v, nj);
297 }

```

16.4.2.2 `template<typename Complex, class Model> bool
complex_factory::hex_grid(size_t N, size_t M, Complex & T,
Model & model, const Point3d & bottom_left = Point3d(0, 0, 0),
const Point3d & u = Point3d(1, 0, 0), const Point3d & v =
Point3d(0, 1, 0)) [inline]`

Create a hexagonal grid with N lines and M columns.

Parameters

- N* Number of lines
- M* Number of columns
- G* Graph to initialize as a grid
- m* [Model](#) containing the initialization functions
- bottom_left* Position of the center of the bottom-left cell
- u* Vector describing the reference system. if $u=(1,0,0)$ and $v=(0,1,0)$, then the hexagon drawn would be of radius 1 and placed vertically.
- v* Vector describing the reference system.

Returns

True if successful

The model should contain this method:

```
void saveGridCellPosition(const cell& c, Point2u pos, Complex&)
```

- Initialize a newly created vertex at line *i* and column *j*. This method is called only on centers, before the junctions exist.

```
Point2u gridCellPosition(const cell& c, Complex&)
```

The neighbors are oriented in that order: bottom -> right -> top -> left

With the bottom-left corner being the one of coordinate (0,0).

The graph is cleared at the beginning of the process

Definition at line 333 of file `complex_grid.h`.

References `connectJunction()`, `forall`, `factory::hex_grid()`, `IMPORT_COMPLEX_TYPES`, `util::Vector< dim, T >::x()`, and `util::Vector< dim, T >::y()`.

```

337 {
338     IMPORT_COMPLEX_TYPES(Complex);
339     HexFiller<Complex,Model> sf(bottom_left, u, v, T, model);
340     if(!factory::hex_grid(N, M, T.C, sf))
341         return false;
342     size_t nb_corners = (M+1)*(N+1)-1;
343     // Bottom-left and top-left corners
344     std::vector<junction> bottom_corner(nb_corners,junction(0)), top_corner(nb_co
rners, junction(0));
345     size_t k = 0;
346     Point3d shift_top_left = bottom_left + v*0.5 - u*std::sqrt(3)/2;
347     Point3d shift_bottom_left = bottom_left - v*0.5 - u*std::sqrt(3)/2;
348     Point3d delta_pos_j = std::sqrt(3)*u;
349     Point3d delta_pos_i = 1.5*v;
350     Point3d odd_shift_pos = std::sqrt(3)/2*u;
351
352     // Grid of cells
353     std::vector<std::vector<cell> > cell_grid(N, std::vector<cell>(M, cell(0)));
354     forall(const cell& c, T.C)
355     {
356         Point2u ij = model.gridCellPosition(c, T);
357         cell_grid[ij.x()][ij.y()] = c;
358     }
359
360     for(size_t i = 0 ; i < N+1 ; ++i)
361         for(size_t j = 0 ; j < M+1 ; ++j, ++k)
362         {
363             if(i == N and j == M)
364                 continue;
365             junction top;
366             junction bottom;
367             T.W.insert(top); T.W.insert(bottom);
368             T.S.insert(top); T.S.insert(bottom);
369             Point3d delta_pos;
370             delta_pos = (double)j*delta_pos_j + (double)i*delta_pos_i;
371             if(i%2)
372                 delta_pos += odd_shift_pos;
373             else if(i == N)
374                 delta_pos += 2*odd_shift_pos;
375             Point3d pos_top;
376             if(i == N)
377             {
378                 pos_top = (double)j*delta_pos_j - v;
379             }
380             else
381                 pos_top = shift_top_left + delta_pos;
382             Point3d pos_bottom = shift_bottom_left + delta_pos;
383             model.setPosition(top, pos_top);
384             model.setPosition(bottom, pos_bottom);
385             bottom_corner[k] = bottom;
386             top_corner[k] = top;
387         }
388
389     // Connect everything
390     for(size_t i = 0 ; i < N ; ++i)
391         for(size_t j = 0 ; j < M ; ++j)
392         {
393             const cell& c = cell_grid[i][j];
394             T.S.insert(c);
395             const junction& c1 = TF_TOP_CORNER(i,j);
396             const junction& c2 = TF_BOTTOM_CORNER(i,j);
397             const junction& c3 = (i%2)?TF_TOP_CORNER(i-1,j+1):TF_TOP_CORNER(i-1,j);

```

```

398     const junction& c4 = TF_BOTTOM_CORNER(i, j+1);
399     const junction& c5 = TF_TOP_CORNER(i, j+1);
400     const junction& c6 = (i%2)?TF_BOTTOM_CORNER(i+1, (i==N-1)?j:j+1):TF_BOTTOM
    _CORNER(i+1, j);
401     // Connect the junctions to each other
402     connectJunction(c1, c2, c6, T.W);
403     connectJunction(c2, c3, c1, T.W);
404     connectJunction(c3, c4, c2, T.W);
405     connectJunction(c4, c5, c3, T.W);
406     connectJunction(c5, c6, c4, T.W);
407     connectJunction(c6, c1, c5, T.W);
408
409     if(i == 0)
410     {
411         connectJunction(c3, c2, c4, T.W, false);
412         connectJunction(c4, c3, c5, T.W, false);
413     }
414     if(j == 0)
415     {
416         if(i%2 == 0)
417         {
418             if(i != 0)
419                 connectJunction(c3, c2, c4, T.W, false);
420             if(i != N-1)
421                 connectJunction(c1, c6, c2, T.W, false);
422         }
423         connectJunction(c2, c1, c3, T.W, false);
424     }
425     if(i == N-1)
426     {
427         connectJunction(c1, c6, c2, T.W, false);
428         connectJunction(c6, c5, c1, T.W, false);
429     }
430     if(j == M-1)
431     {
432         if(i%2)
433         {
434             if(i != N-1)
435                 connectJunction(c6, c5, c1, T.W, false);
436             if(i != 0)
437                 connectJunction(c4, c3, c5, T.W, false);
438         }
439         connectJunction(c5, c4, c6, T.W, false);
440     }
441
442     // Connect the center to the junctions
443     T.S.insertEdge(c, c1);
444     T.S.insertEdge(c, c2);
445     T.S.spliceAfter(c, c2, c3);
446     T.S.spliceAfter(c, c3, c4);
447     T.S.spliceAfter(c, c4, c5);
448     T.S.spliceAfter(c, c5, c6);
449
450     // Connect the junctions to the center
451     if(i == 0 or j == 0)
452     {
453         T.S.insertEdge(c1, c);
454         T.S.insertEdge(c2, c);
455     }
456     else
457     {
458         const cell& nc = T.S.anyIn(c1);

```



```

459         T.S.spliceAfter(c1, nc, c);
460         T.S.spliceBefore(c2, nc, c);
461     }
462     {
463         const cell& nc = T.S.anyIn(c4);
464         if(nc)
465             T.S.spliceAfter(c3, nc, c);
466         else
467             T.S.insertEdge(c3, c);
468     }
469     T.S.insertEdge(c4, c);
470     T.S.insertEdge(c5, c);
471     T.S.insertEdge(c6, c);
472 }
473 return true;
474 }
```

16.4.2.3 `template<class CellComplex , class Model > ObjReaderError complex_factory::objreader (std::string filename, CellComplex & T, Model & model) [inline]`

Convenience methods that read a Wavefront OBJ file.

See also

[objreader\(QTextStream&,CellComplex&,Model&\)](#)

Definition at line 461 of file objreader.h.

References `QString::fromLocal8Bit()`, and `objreader()`.

```

462 {
463     QString fn = QString::fromLocal8Bit(filename.c_str());
464     return objreader(fn, T, model);
465 }
```

16.4.2.4 `template<class CellComplex , class Model > ObjReaderError complex_factory::objreader (const QString & filename, CellComplex & T, Model & model) [inline]`

Convenience methods that read a Wavefront OBJ file.

See also

[objreader\(QTextStream&,CellComplex&,Model&\)](#)

Definition at line 444 of file objreader.h.

References `objreader()`, and `QFile::open()`.

```

445 {
446     QFile f(filename);
```

```

447     if(!f.open(QIODevice::ReadOnly))
448     {
449         return ObjReaderError(ObjReaderError::CANNOT_OPEN_FILE, f.errorString());
450     }
451     QTextStream ts(&f);
452     return objreader(ts, T, model);
453 }
```

16.4.2.5 **template<class CellComplex, class Model> ObjReaderError complex_factory::objreader(QTextStream & *content*, CellComplex & T, Model & *model*) [inline]**

Read a Wavefront OBJ file describing a surface and return the complex representing it.

Parameters

content Stream containing the OBJ file
T Complex to fill with the OBJ file mesh
model Object containing needed methods

This version interpret vertices, faces and normals on vertices.

The `model` object should include, in addition to the methods required by the complex object, the following methods:

```
void setVertexId(const junction& v, int id);
```

Associate the identifier `id` with the vertex `v`. The identifiers for faces and junctions are independant. Typically, the id will be store in the vertex itself.

```
void setFaceId(const cell& f, int id);
```

Associate the identifier `id` with the face `f`. The identifiers for faces and junctions are independant. Typically, the id will be store in the face itself.

```
int vertexId(const junction& v);
```

Returns the identifier that was previously associated with the vertex `v`.

```
int faceId(const cell& f);
```

Returns the identifier that was previously associated with the face `f`.

```
void setNormal(const cell& c, const Point3d& normal);
```

Set the normal of the face `v` to `normal`. The normal is just for information, and do not have to be used.

```
void setNormal(const junction& v, const Point3d& normal);
```

Set the normal of the vertex `v` to `normal`. The normal is just for information, and do not have to be used.

Definition at line 126 of file `objreader.h`.

References `QTextStream::atEnd()`, `forall`, `forall_named`, `IMPORT_COMPLEX_TYPES`, `util::Vector< dim, T >::normalize()`, `QString::number()`, `QTextStream::readLine()`, `util::Vector< dim, T >::size()`, `QList::size()`, `QString::split()`, `QString::toInt()`, `util::Vector< dim, T >::x()`, `util::Vector< dim, T >::y()`, and `util::Vector< dim, T >::z()`.

Referenced by `objreader()`.

```

127 {
128     IMPORT_COMPLEX_TYPES(CellComplex);
129
130     std::vector<Point3d> vertices_pos;
131     std::vector<Point3d> normals;
132     std::vector<std::vector<int> > faces_idx;
133
134     QString token;
135
136     while(!content.atEnd())
137     {
138         content >> token;
139         QString buffer = content.readLine();
140         cerr << "Reading token '" << token.toStdString() << "' " << endl;
141         cerr << "Buffer: '" << buffer.toStdString() << "' " << endl;
142         if(token == QString("v"))
143         {
144             QList<QString> pos = buffer.split(' ', QString::SkipEmptyParts);
145             if(pos.size() != 3)
146             {
147                 QString msg = "Error, vertex " + QString::number(vertices_pos.size()) +
148
149                     " has " + QString::number(pos.size()) + " dimensions instead of 3";
150                 return ObjReaderError(ObjReaderError::BAD_NUMBER_OF_DIMENSIONS, msg);
151             }
152             Point3d p;
153             bool ok, ok_tmp;
154             p.x() = pos[0].toDouble(&ok);
155             p.y() = pos[1].toDouble(&ok_tmp);
156             ok &= ok_tmp;
157             p.z() = pos[2].toDouble(&ok_tmp);
158             ok &= ok_tmp;
159             if(!ok)
160             {
161                 QString msg = "Error on vertex " + QString::number(vertices_pos.size())
162
163                     + ": double value expected";
164                 return ObjReaderError(ObjReaderError::DOUBLE_EXPECTED, msg);
165             }
166             vertices_pos.push_back(p);
167         }
168         else if(token == QString("vn"))
169         {
170             QList<QString> pos = buffer.split(' ');
171             if(pos.size() != 3)
172             {
173                 QString msg = "Error, normal " + QString::number(normals.size()) + " ha
174 s " +

```

```

172         QString::number(pos.size()) + " dimensions instead of 3";
173         return ObjReaderError(ObjReaderError::BAD_NUMBER_OF_DIMENSIONS, msg);
174     }
175     Point3d p;
176     bool ok, ok_tmp;
177     p.x() = pos[0].toDouble(&ok);
178     p.y() = pos[1].toDouble(&ok_tmp);
179     ok &= ok_tmp;
180     p.z() = pos[2].toDouble(&ok_tmp);
181     ok &= ok_tmp;
182     if(!ok)
183     {
184         QString msg = "Error on normal " + QString::number(vertices_pos.size())
+ ": double value expected";
185         return ObjReaderError(ObjReaderError::DOUBLE_EXPECTED, msg);
186     }
187     normals.push_back(p);
188 }
189 else if(token == QString("f"))
190 {
191     std::vector<int> f;
192     QList<QString> face = buffer.split(' ', QString::SkipEmptyParts);
193     if(face.size() < 3)
194     {
195         QString msg = "Error, face " + QString::number(faces_idx.size()) + " ha
s less than 3 vertices.";
196         return ObjReaderError(ObjReaderError::INVALID_FACE, msg);
197     }
198     bool ok = true;
199     forall(QString& value, face)
200     {
201         int vn = value.toInt(&ok);
202         if(!ok)
203         {
204             QString msg = "Error on face " + QString::number(faces_idx.size()) +
": integer values expected";
205             return ObjReaderError(ObjReaderError::INTEGER_EXPECTED, msg);
206         }
207         f.push_back(vn);
208     }
209     faces_idx.push_back(f);
210 }
211 }
212 if(normals.size() > vertices_pos.size())
213 {
214     cerr << "Warning: more normals than vertices" << endl;
215 }
216 if(!normals.empty() and normals.size() < vertices_pos.size())
217 {
218     cerr << "Warning: not all vertices have a normal, ignoring the specified no
rmals." << endl;
219     normals.clear();
220 }
221 // First, add the vertices
222 std::vector<junction> vertices;
223 std::vector<cell> faces;
224 int id = 0;
225 forall(const Point3d& pos, vertices_pos)
226 {
227     junction v;
228     T.S.insert(v);
229     T.W.insert(v);

```

```
230     model.setPosition(v, pos);
231     vertices.push_back(v);
232     model.setVertexId(v, ++id);
233 }
234 // Then add the faces and connect the center to the junctions (in the
235 // correct order)
236 id = 0;
237 forall(const std::vector<int>& face_idx, faces_idx)
238 {
239     cell face;
240     junction pf(0);
241     T.S.insert(face);
242     T.C.insert(face);
243     Point3d center;
244     forall(int i, face_idx)
245     {
246         center += vertices_pos[i-1];
247         junction current = vertices[i-1];
248         if(pf)
249         {
250             T.S.spliceAfter(face, pf, current);
251         }
252         else
253             T.S.insertEdge(face, current);
254         pf = current;
255     }
256     center /= face_idx.size();
257     model.setPosition(face, center);
258     model.setFaceId(face, ++id);
259 }
260 // Connect the vertices to the faces
261 forall_named(const junction& j, T.S, junctions)
262 {
263     if(T.S.iValence(j) < 3)
264     {
265         // There is nothing to sort ...
266         forall(const cell& f, T.S.iNeighbors(j))
267         {
268             T.S.insertEdge(j, f);
269         }
270     }
271     else
272     {
273         const cell& f_init = T.S.iAnyIn(j);
274         cell pf = f_init;
275         T.S.insertEdge(j, f_init);
276         do
277         {
278             cell f(0);
279             const junction& m = T.S.prevTo(pf, j);
280             // Now, find the face that contains also m
281             forall(const cell& of, T.S.iNeighbors(j))
282             {
283                 if(of == pf)
284                     continue;
285                 if(T.S.edge(of, m))
286                 {
287                     f = of;
288                     break;
289                 }
290             }
291             if(!f or f == f_init)
```

```

292         {
293             break;
294         }
295         // Insert f after pf in j
296         T.S.spliceAfter(j, pf, f);
297         pf = f;
298     } while( T.S.iValence(j) != T.S.valence(j));
299     pf = f_init;
300     // If we didn't find all the neighbors, it means we have to go the
301     // other way around
302     while(T.S.iValence(j) != T.S.valence(j))
303     {
304         cell f(0);
305         const junction& m = T.S.nextTo(pf, j);
306         forall(const cell& of, T.S.iNeighbors(j))
307         {
308             if(of == pf)
309                 continue;
310             if(T.S.edge(of, m))
311             {
312                 f = of;
313                 break;
314             }
315         }
316         if(!f)
317             break;
318         T.S.spliceBefore(j, pf, f);
319         pf = f;
320     }
321     if(T.S.valence(j) != T.S.iValence(j))
322     {
323         cerr << "Error, vertex " << model.vertexId(j) << " was not properly con
nected to the faces" << endl;
324     }
325 }
326 }
327 // Connect the junctions
328 forall_named(const junction& j, T.S, junctions)
329 {
330     junction prev(0);
331     forall(const cell& c, T.S.neighbors(j))
332     {
333         const junction& jn = T.S.nextTo(c, j);
334         const junction& jp = T.S.prevTo(c, j);
335         if(prev.isNull())
336         {
337             T.W.insertEdge(j, jn);
338             prev = jn;
339         }
340         else if(prev != jn)
341         {
342             T.W.spliceAfter(j, prev, jn);
343         }
344         T.W.spliceAfter(j, jn, jp);
345         prev = jp;
346     }
347 }
348 // if the normals aren't specified, then compute them
349 if(normals.empty())
350 {
351     forall_named(const junction& v, T.S, junctions)
352     {

```

```

353     Point3d pos = model.position(v);
354     Point3d normal;
355     if(T.border(v))
356     {
357         // If this is a border index, first find the border
358         forall(const junction& n, T.W.neighbors(v))
359         {
360             const junction& m = T.W.nextTo(v,n);
361             if(T.border(v, m) and T.border(v, n))
362             {
363                 junction n1 = m;
364                 do
365                 {
366                     const junction& n2 = T.W.nextTo(v, n1);
367                     Point3d p1 = model.position(n1) - pos;
368                     Point3d p2 = model.position(n2) - pos;
369                     normal += p1^p2;
370                     n1 = n2;
371                 } while(n1 != n);
372                 break;
373             }
374         }
375     }
376     else
377     {
378         // If this is an inside vertex
379         forall(const junction& n, T.W.neighbors(v))
380         {
381             Point3d p1 = model.position(n) - pos;
382             Point3d p2 = model.position(T.W.nextTo(v, n)) - pos;
383             normal += p1^p2;
384         }
385     }
386     normal.normalize();
387     model.setNormal(v, normal);
388 }
389 }
390 else
391 {
392     int k = 0;
393     forall(const Point3d& n, normals)
394     {
395         junction v = vertices[k++];
396         model.setNormal(v, n);
397     }
398 }
399 // Compute normals for the faces
400 forall_named(const cell& f, T.S, cells)
401 {
402     Point3d pos = model.position(f);
403     Point3d normal;
404     forall(const junction& v, T.S.neighbors(f))
405     {
406         const junction& v1 = T.S.nextTo(f, v);
407         Point3d p1 = model.position(v) - pos;
408         Point3d p2 = model.position(v1) - pos;
409         normal += p1^p2;
410     }
411     normal.normalize();
412     model.setNormal(f, normal);
413 }
414 // At last, connect the cell graph

```

```

415     forall_named(const cell& f, T.S, cells)
416     {
417         cell pf(0);
418         forall(const junction& j, T.S.neighbors(f))
419         {
420             const cell& f2 = T.adjacentCell(f, j);
421             if(f2)
422             {
423                 if(pf)
424                 {
425                     T.C.spliceAfter(f, pf, f2);
426                 }
427                 else
428                 {
429                     T.C.insertEdge(f, f2);
430                 }
431                 pf = f2;
432             }
433         }
434     }
435     return ObjReaderError(ObjReaderError::NO_ERRORS);
436 }

```

16.4.2.6 `template<typename Complex, class Model> bool complex_factory::square_grid(size_t N, size_t M, Complex & T, Model & model, const Point3d & bottom_left = Point3d(0, 0, 0), const Point3d & shift_right = Point3d(1, 0, 0), const Point3d & shift_up = Point3d(0, 1, 0)) [inline]`

Create a square grid with N lines and M columns.

Parameters

N Number of lines

M Number of columns

G Graph to initialize as a grid

m [Model](#) containing the initialization functions

bottom_left Position of the bottom-left corner of the bottom_left cell

shift_right Vector describing the width of a cell

shift_up Vector describing the height of a cell

Returns

True if successful

The model should contain this method:

```
void saveGridCellPosition(const cell&, Point2u pos, Complex&)
```

- Initialize a newly created vertex at line *i* and column *j*. This method is called only on centers, before the junctions exist.


```
Point2u gridCellPosition(const cell&, Complex&)
```

The neighbors are oriented in that order: bottom -> right -> top -> left

With the bottom-left corner being the one of coordinate (0,0).

The graph is cleared at the beginning of the process

Definition at line 116 of file complex_grid.h.

References `forall`, `IMPORT_COMPLEX_TYPES`, `algorithms::insertAfter()`, `factory::square_grid()`, `util::Vector< dim, T >::x()`, and `util::Vector< dim, T >::y()`.

```
120 {
121     IMPORT_COMPLEX_TYPES(Complex);
122     SquareFiller<Complex,Model> sf(bottom_left, shift_right, shift_up, T, model);

123     if(!factory::square_grid(N, M, T.C, sf))
124         return false;
125     // Store the bottom-left corner of each cell
126     std::vector<junction> corners((N+1)*(M+1), junction(0));
127
128     // Grid of cells
129     std::vector<std::vector<cell> > cell_grid(N, std::vector<cell>(M, cell(0)));
130     forall(const cell& c, T.C)
131     {
132         Point2u ij = model.gridCellPosition(c, T);
133         cell_grid[ij.x()][ij.y()] = c;
134     }
135
136     // Create the corners and connect them to the cells
137     size_t k = 0;
138     for(size_t i = 0 ; i < N+1 ; ++i)
139         for(size_t j = 0 ; j < M+1 ; ++j, ++k)
140         {
141             Point3d pos = bottom_left + (1.0*i)*shift_up + (1.0*j)*shift_right;
142             junction v;
143             T.S.insert(v);
144             T.W.insert(v);
145             model.setPosition(v, pos);
146             corners[k] = v;
147         }
148     // Connect the corners to each other
149     const size_t delta_right = 1;
150     const size_t delta_up = M+1;
151     k = 0;
152     for(size_t i = 0 ; i < N+1 ; ++i)
153         for(size_t j = 0 ; j < M+1 ; ++j, ++k)
154         {
155             const junction& v = corners[k];
156             junction pv(0);
157             // First, connect up
158             if(i < N)
159             {
160                 pv = corners[k+delta_up];
161                 T.W.insertEdge(v, pv);
162             }
163             // Second, connect left
164             if(j > 0)
165             {
```

```

166         pv = corners[k-delta_right];
167         T.W.insertEdge(v, pv);
168     }
169     // Third, connect down
170     if(i > 0)
171     {
172         const junction& nv = corners[k-delta_up];
173         insertAfter(v, pv, nv, T.W);
174         pv = nv;
175     }
176     // At last, connect right
177     if(j < M)
178     {
179         const junction& nv = corners[k+delta_right];
180         insertAfter(v, pv, nv, T.W);
181     }
182 }
183 // Next, connect the cells to the corners
184 for(size_t i = 0 ; i < N ; ++i)
185     for(size_t j = 0 ; j < M ; ++j, ++k)
186     {
187         const cell& c = cell_grid[i][j];
188         T.S.insert(c);
189         junction c1 = TF_CORNER(i, j);
190         junction c2 = TF_CORNER(i, j+1);
191         junction c3 = TF_CORNER(i+1, j+1);
192         junction c4 = TF_CORNER(i+1, j);
193         T.S.insertEdge(c, c1);
194         T.S.insertEdge(c, c2);
195         T.S.spliceAfter(c, c2, c3);
196         T.S.spliceAfter(c, c3, c4);
197
198         if(i == 0)
199         {
200             T.S.insertEdge(c1, c);
201             T.S.insertEdge(c2, c);
202         }
203         else
204         {
205             forall(const cell& cc, T.S.neighbors(c1))
206             {
207                 Point2u ij = model.gridCellPosition(cc, T);
208                 if(ij.x() == i-1 and ij.y() == j)
209                 {
210                     T.S.spliceAfter(c1, cc, c);
211                     T.S.spliceBefore(c2, cc, c);
212                     break;
213                 }
214             }
215         }
216
217         T.S.insertEdge(c3, c);
218         T.S.insertEdge(c4, c);
219
220         /*
221         T.S.spliceAfter(c1, c2, c);
222         T.S.spliceAfter(c2, c3, c);
223         T.S.spliceAfter(c3, c4, c);
224         T.S.spliceAfter(c4, c1, c);
225         */
226     }
227     return true;

```

```
228 }
```

16.4.2.7 `template<typename Complex , class Model >
SquareFiller<Complex,Model> complex_factory::squareFiller (const
Point3d & bottom_left, const Point3d & shift_right, const Point3d &
shift_up, Complex & T, Model & model) [inline]`

Create a square filler.

Definition at line 74 of file complex_grid.h.

```
79 {  
80     return SquareFiller<Complex,Model>(bottom_left, shift_right, shift_up, T, mod  
81     el);  
81 }
```

16.5 factory Namespace Reference

Contains predefined ways to initialize a graph.

Functions

- `template<typename Graph , class Model >`
`bool hex_grid (size_t N, size_t M, Graph &G, Model &model, bool torus=false)`

Build a hexagonal grid with N lines and M columns.

- `template<typename Graph , class Model >`
`bool square_grid (size_t N, size_t M, Graph &G, Model &model, bool torus=false, bool connect8=false)`

Create a square grid with N lines and M columns.

16.5.1 Detailed Description

Contains predefined ways to initialize a graph.

16.5.2 Function Documentation

16.5.2.1 `template<typename Graph , class Model > bool factory::hex_grid (size_t N, size_t M, Graph & G, Model & model, bool torus = false) [inline]`

Build a hexagonal grid with N lines and M columns.

Parameters

N Number of lines of the grid

M Number of columns of the grid

G Graph to fill in

model [Model](#) containing initialisation method

torus True to connect the top cells to the bottom ones and the left cells to the right ones. Requires an even number of lines.

Returns

True if successful

Definition at line 257 of file grid.h.

Referenced by `complex_factory::hex_grid()`.

```

258 {
259     if(torus and N%2)
260         return false;
261     typedef typename Graph::vertex_t vertex;
262     typedef typename Graph::edge_t edge;
263     // First, clear the graph
264     G.clear();
265     // Array of vertices by position
266     std::vector<vertex> vertices(M*N, vertex(0));
267     // Create the vertices and insert them
268     for(size_t k = 0 ; k < M*N ; ++k)
269     {
270         vertex v;
271         G.insert(v);
272         vertices[k] = v;
273     }
274     // Create the edges
275     size_t k = 0;
276     const size_t delta_right = 1;
277     const size_t delta_top_right_even = M;
278     const size_t delta_top_right_odd = M+1;
279     const size_t delta_top_left_even = M-1;
280     const size_t delta_top_left_odd = M;
281     const size_t delta_bottom_border_bottom_right = M*(N-1);
282     const size_t delta_bottom_border_bottom_left = M*(N-1)-1;
283     const size_t delta_right_border_top_right = 1;
284     const size_t delta_left_border_left = M-1;
285     const size_t delta_left_border_top_left = 2*M-1;
286     const size_t top_right_corner = M*N-1;
287     const size_t bottom_left_corner = 0;
288     for(size_t i = 0 ; i < N ; ++i)
289         for(size_t j = 0 ; j < M ; ++j, ++k)
290         {
291             vertex v = vertices[k];
292             // ** Connect to the right
293             if(j == M-1)
294             {
295                 if(torus)
296                     G.insertEdge(v, vertices[k-delta_left_border_left]);
297             }
298             else
299                 G.insertEdge(v, vertices[k+delta_right]);
300             // ** Connect to the left
301             if(j==0)
302             {
303                 if(torus)
304                     G.insertEdge(v, vertices[k+delta_left_border_left]);
305             }
306             else
307                 G.insertEdge(v, vertices[k-delta_right]);
308             // ** Connect to the top-right
309             if(k == top_right_corner)
310             {
311                 if(torus)
312                     G.spliceAfter(v, vertices[k-delta_left_border_left], vertices[bottom_
313 left_corner]);
314             }
315             else if(i == N-1)
316             {
317                 if(torus)
318                     G.spliceAfter(v, vertices[k+delta_right], vertices[k-delta_bottom_bor
319 der_bottom_left]);

```

```

318     }
319     else if(i%2)
320     {
321         if(j == M-1)
322         {
323             if(torus)
324                 G.spliceAfter(v, vertices[k-delta_left_border_left], vertices[k+del
ta_right_border_top_right]);
325         }
326         else
327             G.spliceAfter(v, vertices[k+delta_right], vertices[k+delta_top_right_
odd]);
328     }
329     else if(j == M-1)
330         G.spliceBefore(v, vertices[k-delta_right], vertices[k+delta_top_right_e
ven]);
331     else
332         G.spliceAfter(v, vertices[k+delta_right], vertices[k+delta_top_right_ev
en]);
333     // ** Connect to the top-left
334     if(k == top_right_corner)
335     {
336         if(torus)
337             G.spliceAfter(v, vertices[bottom_left_corner], vertices[k-delta_botto
m_border_bottom_right]);
338     }
339     else if(i == N-1)
340     {
341         if(torus)
342             G.spliceBefore(v, vertices[k-delta_right], vertices[k-delta_bottom_bo
rder_bottom_right]);
343     }
344     else if(i%2)
345     {
346         if(j == 0)
347             G.spliceAfter(v, vertices[k+delta_top_right_odd], vertices[k+delta_to
p_left_odd]);
348         else
349             G.spliceBefore(v, vertices[k-delta_right], vertices[k+delta_top_left_
odd]);
350     }
351     else
352     {
353         if(j == 0)
354         {
355             if(torus)
356                 G.spliceAfter(v, vertices[k+delta_top_right_even], vertices[k+delta_
_left_border_top_left]);
357         }
358         else
359             G.spliceBefore(v, vertices[k-delta_right], vertices[k+delta_top_left_
even]);
360     }
361     // ** Connect to the bottom-left
362     if(k == bottom_left_corner)
363     {
364         if(torus)
365             G.spliceBefore(v, vertices[k+delta_right], vertices[top_right_corner]
);
366     }
367     else if(i == 0)
368     {

```

```

369         if(torus)
370             G.spliceAfter(v, vertices[k-delta_right], vertices[k+delta_bottom_bor
der_bottom_left]);
371     }
372     else if(i%2)
373     {
374         if(j==0)
375             G.spliceBefore(v, vertices[k+delta_right], vertices[k-delta_top_right
_even]);
376     }
377     else
378         G.spliceAfter(v, vertices[k-delta_right], vertices[k-delta_top_right_
even]);
379     }
380     else if(j == 0)
381     {
382         if(torus)
383             G.spliceBefore(v, vertices[k+delta_right], vertices[k-delta_right_bor
der_top_right]);
384     }
385     else
386         G.spliceAfter(v, vertices[k-delta_right], vertices[k-delta_top_right_od
d]);
387     // ** Connect to the bottom-right
388     if(k == bottom_left_corner)
389     {
390         if(torus)
391             G.spliceAfter(v, vertices[top_right_corner], vertices[k+delta_bottom_
border_bottom_right]);
392     }
393     else if(i == 0)
394     {
395         if(torus)
396             G.spliceAfter(v, vertices[k+delta_bottom_border_bottom_left], vertice
s[k+delta_bottom_border_bottom_right]);
397     }
398     else if(i%2)
399     {
400         if(j==M-1)
401         {
402             if(torus)
403                 G.spliceBefore(v, vertices[k-delta_left_border_left], vertices[k-de
lta_left_border_top_left]);
404             }
405             else
406                 G.spliceBefore(v, vertices[k+delta_right], vertices[k-delta_top_left_
even]);
407         }
408         else if(j == M-1)
409             G.spliceAfter(v, vertices[k-delta_top_right_odd], vertices[k-delta_top_
left_odd]);
410         else
411             G.spliceBefore(v, vertices[k+delta_right], vertices[k-delta_top_left_od
d]);
412     }
413     }
414     k = 0;
415     for(size_t i = 0 ; i < N ; ++i)
416     for(size_t j = 0 ; j < M ; ++j, ++k)
417     {
418         model.initGridVertex(i, j, vertices[k], G);
419     }
420     return true;
421 }

```

16.5.2.2 `template<typename Graph , class Model > bool factory::square_grid (size_t N, size_t M, Graph & G, Model & model, bool torus = false, bool connect8 = false) [inline]`

Create a square grid with N lines and M columns.

Parameters

N Number of lines

M Number of columns

G Graph to initialize as a grid

m [Model](#) containing the initialization functions

torus Connect the top cells to the bottom ones and the left cells to the right ones if true.

connect8 Set to true to get a 8-connected grid, false to get a 4-connected grid.

Returns

True if successful

The model should contain this method:

```
void initGridVertex(size_t i, size_t j, const vertex&, Graph&)
```

Initialize a newly created vertex at line i and column j

The neighbors are oriented in that order: bottom -> right -> top -> left

With the bottom-left corner being the one of coordinate (0,0).

The graph is cleared at the beginning of the process

Definition at line 46 of file grid.h.

Referenced by `complex_factory::square_grid()`.

```
47  {
48      typedef typename Graph::vertex_t vertex;
49      typedef typename Graph::edge_t edge;
50      // First, clear the graph
51      G.clear();
52      // Array of vertices by position
53      std::vector<vertex> vertices(M*N, vertex(0));
54      // Create the vertices and insert them
55      for(size_t k = 0 ; k < M*N ; ++k)
56      {
57          vertex v;
58          G.insert(v);
59          vertices[k] = v;
60      }
61      // Create the edges
62      size_t k = 0;
63      const size_t delta_border_i = M*(N-1);
64      const size_t delta_border_j = M-1;
65      const size_t delta_i = M;
```



```

66     const size_t delta_j = 1;
67     for(size_t i = 0 ; i < N ; ++i)
68         for(size_t j = 0 ; j < M ; ++j, ++k)
69             {
70                 vertex v = vertices[k];
71                 // Connect to the right
72                 if(j == M-1)
73                     {
74                         if(torus)
75                             G.insertEdge(v, vertices[k-delta_border_j]);
76                     }
77                 else
78                     G.insertEdge(v, vertices[k+delta_j]);
79                 // Connect to the top
80                 if(i == N-1)
81                     {
82                         if(torus)
83                             G.insertEdge(v, vertices[k-delta_border_i]);
84                     }
85                 else
86                     G.insertEdge(v, vertices[k+delta_i]);
87                 // Connect to the left
88                 if(j == 0)
89                     {
90                         if(torus)
91                         {
92                             if(i==N-1)
93                                 G.spliceAfter(v, vertices[k-delta_border_i], vertices[k+delta_borde
94 r_j]);
95                         }
96                     }
97                 else
98                     {
99                         if(i==N-1)
100                         {
101                             if(j == M-1)
102                                 G.insertEdge(v, vertices[k-delta_j]);
103                             else
104                                 G.spliceAfter(v, vertices[k+delta_j], vertices[k-delta_j]);
105                         }
106                     }
107                 else
108                     G.spliceAfter(v, vertices[k+delta_i], vertices[k-delta_j]);
109             }
110             // Connect to the bottom
111             if(i == 0)
112                 {
113                     if(torus)
114                     {
115                         if(j==0)
116                             G.spliceAfter(v, vertices[k+delta_border_j], vertices[k+delta_borde
117 r_i]);
118                     }
119                 }
120             else
121                 {
122                     if(j==0)
123                         G.spliceBefore(v, vertices[k+delta_j], vertices[k-delta_i]);
124                     else

```

```

126         G.spliceAfter(v, vertices[k-delta_j], vertices[k-delta_i]);
127     }
128 }
129 if(connect8)
130 {
131     const size_t delta_ij = M+1;
132     const size_t delta_mij = M-1;
133     const size_t bottom_left_corner = 0;
134     const size_t bottom_right_corner = M-1;
135     const size_t top_left_corner = M*(N-1);
136     const size_t top_right_corner = M*N-1;
137     const size_t delta_border_bottom_ij = M*(N-1)-1;
138     const size_t delta_border_right_ij = 1;
139     const size_t delta_border_bottom_mij = M*(N-1)+1;
140     const size_t delta_border_left_mij = 2*M-1;
141     k = 0;
142     for(size_t i = 0 ; i < N ; ++i)
143         for(size_t j = 0 ; j < M ; ++j, ++k)
144         {
145             vertex v = vertices[k];
146             // ** Connect to the bottom-left
147             if(i == 0 && j == 0)
148             {
149                 // If in the corner
150                 if(torus)
151                     G.spliceAfter(v, vertices[bottom_right_corner], vertices[top_right_
corner]);
152             }
153             else if(i == 0)
154             {
155                 // If on the bottom line
156                 if(torus)
157                     G.spliceAfter(v, vertices[k-delta_j], vertices[k+delta_border_botto
m_ij]);
158             }
159             else if(j == 0)
160             {
161                 // If on the left border
162                 if(torus)
163                     G.spliceBefore(v, vertices[k-delta_i], vertices[k-delta_border_righ
t_ij]);
164             }
165             else
166                 G.spliceAfter(v, vertices[k-delta_j], vertices[k-delta_ij]);
167
168             // ** Connect the bottom-right
169             if(i == 0 && j == M-1)
170             {
171                 // If in the corner
172                 if(torus)
173                     G.spliceAfter(v, vertices[top_right_corner], vertices[top_left_corn
er]);
174             }
175             else if(i == 0)
176             {
177                 // If on the bottom line
178                 if(torus)
179                     G.spliceBefore(v, vertices[k+delta_j], vertices[k+delta_border_bott
om_mij]);
180             }
181             else if(j == M-1)
182             {

```

```

183         // If on the right line
184         if(torus)
185             G.spliceAfter(v, vertices[k-delta_i], vertices[k-delta_border_left_
mij]);
186     }
187     else
188         G.spliceAfter(v, vertices[k-delta_i], vertices[k-delta_mij]);
189
190     // ** Connect to the top-right
191     if(i == N-1 && j == M-1)
192     {
193         // If on the corner
194         if(torus)
195             G.spliceAfter(v, vertices[top_left_corner], vertices[bottom_left_co
rner]);
196     }
197     else if(i == N-1)
198     {
199         // If on the top line
200         if(torus)
201             G.spliceAfter(v, vertices[k+delta_j], vertices[k-delta_border_botto
m_mij]);
202     }
203     else if(j == M-1)
204     {
205         // If on the right line
206         if(torus)
207             G.spliceBefore(v, vertices[k+delta_i], vertices[k+delta_border_righ
t_mij]);
208     }
209     else
210         G.spliceAfter(v, vertices[k+delta_j], vertices[k+delta_ij]);
211
212     // ** Connect to the top-left
213     if(i == N-1 && j == 0)
214     {
215         // If on the bottom-left corner
216         if(torus)
217             G.spliceAfter(v, vertices[bottom_left_corner], vertices[bottom_righ
t_corner]);
218     }
219     else if(i == N-1)
220     {
221         // If on the top line
222         if(torus)
223             G.spliceBefore(v, vertices[k-delta_j], vertices[k-delta_border_bott
om_mij]);
224     }
225     else if(j == 0)
226     {
227         // If on the left line
228         if(torus)
229             G.spliceAfter(v, vertices[k+delta_i], vertices[k+delta_border_left_
mij]);
230     }
231     else
232         G.spliceBefore(v, vertices[k-delta_j], vertices[k+delta_mij]);
233     }
234 }
235 k = 0;
236 for(size_t i = 0 ; i < N ; ++i)
237     for(size_t j = 0 ; j < M ; ++j, ++k)

```

```
238     {  
239         model.initGridVertex(i, j, vertices[k], G);  
240     }  
241     return true;  
242 }
```

16.6 geometry Namespace Reference

Algorithmic geometry.

Classes

- class [Quaternion](#)
Implements the quaternion operations.

Typedefs

- typedef [util::Matrix](#)< 2, 2, double > [Matrix2d](#)
Type of 2x2 matrix.
- typedef [util::Matrix](#)< 3, 3, double > [Matrix3d](#)
Type of 3x3 matrix.
- typedef [util::Matrix](#)< 4, 4, double > [Matrix4d](#)
Type of a 4x4 matrix.
- typedef [util::Vector](#)< 2, double > [Point2d](#)
Type of a 2D point.
- typedef [util::Vector](#)< 3, double > [Point3d](#)
Type of a 3D point.
- typedef [util::Vector](#)< 4, double > [Point4d](#)
Type of a 4D point.

Functions

- [Point3d barycentricToCartesian](#) (const [Point3d](#) &barycentric, const [Point3d](#) &p1, const [Point3d](#) &p2, const [Point3d](#) &p3)
Convert barycentric coordinates to cartesian.
- [Matrix3d barycentricToCartesian_matrix](#) (const [Point3d](#) &p1, const [Point3d](#) &p2, const [Point3d](#) &p3)
Convert cartesian coordinates to barycentric coordinates in a triangle.
- [Point3d cartesianToBarycentric](#) (const [Point3d](#) &pos, const [Point3d](#) &p1, const [Point3d](#) &p2, const [Point3d](#) &p3)
Convert cartesian coordinates to barycentric coordinates in a triangle.

- [Matrix3d cartesianToBarycentric_matrix](#) (const [Point3d](#) &p1, const [Point3d](#) &p2, const [Point3d](#) &p3, int &index_one)
Convert cartesian coordinates to barycentric coordinates in a triangle.
- [Point2d centroid](#) (const std::vector< [Point2d](#) > &polygon)
Center of mass of a 2D non-overlapping polygon.
- bool [lineLineIntersection](#) ([Point2d](#) &u, const [Point2d](#) &p1, const [Point2d](#) &u1, const [Point2d](#) &p2, const [Point2d](#) &u2)
Find if the segment [p1,p2] intersect the line going through p of normal n.
- bool [lineSegmentIntersection](#) ([Point2d](#) &u, const [Point2d](#) &p1, const [Point2d](#) &p2, const [Point2d](#) &p, const [Point2d](#) &n)
Find if the segment [p1,p2] intersect the line going through p of normal n.
- bool [lineTriangleIntersection](#) ([Point3d](#) &u, double &s, const [Point3d](#) &p1, const [Point3d](#) &p2, const [Point3d](#) &tr1, const [Point3d](#) &tr2, const [Point3d](#) &tr3)
Line-triangle intersection.
- [Quaternion operator*](#) (const [Quaternion](#) &q, double s)
- [Quaternion operator*](#) (double s, const [Quaternion](#) &q)
- bool [planeLineIntersection](#) ([Point3d](#) &u, double &s, const [Point3d](#) &p, const [Point3d](#) &n, const [Point3d](#) &u1, const [Point3d](#) &u2)
Plane-Line Intersection.
- template<typename PointContainer >
bool [pointInPolygon](#) (const [Point2d](#) &p, const PointContainer &polygon)
- bool [pointInPolygon](#) (const [Point2d](#) &p, const std::vector< [Point2d](#) > &polygon)
Find if the point p is inside the polygon using the winding number.
- bool [pointInTriangle](#) ([Point3d](#) &u, double &s, const [Point3d](#) &p, const [Point3d](#) &p1, const [Point3d](#) &p2, const [Point3d](#) &p3)
Point in triangle test (3D).
- bool [pointInTriangle](#) (const [Point2d](#) &p, const [Point2d](#) &p1, const [Point2d](#) &p2, const [Point2d](#) &p3)
Point in triangle test (2D).
- template<size_t N>
double [polygonArea](#) (const std::vector< [util::Vector](#)< N, double > > &polygon)
Area of a planar polygon.
- template<size_t N>
[util::Vector](#)< N, double > [projectPointOnLine](#) (const [util::Vector](#)< N, double > &pt, const [util::Vector](#)< N, double > &u1, const [util::Vector](#)< N, double > &u2, bool strict=false)

Project a point on a line.

- [Point3d projectPointOnPlane](#) (const [Point3d](#) &pos, const [Point3d](#) &p, const [Point3d](#) &n)

Project a point into a plane defined by a point and a normal to the plane.

- [Point3d projectPointOnTriangle](#) (const [Point3d](#) &pt, const [Point3d](#) &p1, const [Point3d](#) &p2, const [Point3d](#) &p3, double &s, bool *in_triangle=0)

Project a point into a triangle.

- bool [segmentSegmentIntersection](#) ([Point2d](#) &u, const [Point2d](#) &p1, const [Point2d](#) &p2, const [Point2d](#) &q1, const [Point2d](#) &q2)

Find if two 2D segments intersect.

- [Quaternion slerp](#) (const [Quaternion](#) &q1, const [Quaternion](#) &q2, double t)
- template<size_t N>
double [triangleArea](#) (const [util::Vector](#)< N, double > &a, const [util::Vector](#)< N, double > &b, const [util::Vector](#)< N, double > &c)

Area of a triangle in 2D or 3D.

16.6.1 Detailed Description

Algorithmic geometry. This namespace contains different usefull geometry algorithms, like intersections, areas, ...

16.6.2 Typedef Documentation

16.6.2.1 typedef util::Matrix<2,2,double> geometry::Matrix2d

Type of 2x2 matrix.

Definition at line 41 of file geometry.h.

16.6.2.2 typedef util::Matrix<3,3,double> geometry::Matrix3d

Type of 3x3 matrix.

Definition at line 46 of file geometry.h.

16.6.2.3 typedef util::Matrix<4,4,double> geometry::Matrix4d

Type of a 4x4 matrix.

Definition at line 51 of file geometry.h.

16.6.2.4 typedef util::Vector<2,double> geometry::Point2d

Type of a 2D point.

Definition at line 36 of file geometry.h.

16.6.2.5 typedef util::Vector<3,double> geometry::Point3d

Type of a 3D point.

Definition at line 31 of file geometry.h.

16.6.2.6 typedef util::Vector<4,double> geometry::Point4d

Type of a 4D point.

Definition at line 26 of file geometry.h.

16.6.3 Function Documentation**16.6.3.1 Point3d geometry::barycentricToCartesian (const Point3d & *barycentric*, const Point3d & *p1*, const Point3d & *p2*, const Point3d & *p3*)**

Convert barycentric coordinates to cartesian.

The triangle is defined by *p1*, *p2*, *p3* and the result is such that: *result* = *barycentric*[0]**p1* + *barycentric*[1]**p2* + *barycentric*[2]**p3*

Referenced by algorithms::TriangleSurface::point3d(), and algorithms::TriangleSurface::smoothNormal().

16.6.3.2 Matrix3d geometry::barycentricToCartesian_matrix (const Point3d & *p1*, const Point3d & *p2*, const Point3d & *p3*)

Convert cartesian coordinates to barycentric coordinates in a triangle.

This version simply returns the matrix that convert from cartesian to barycentric.

16.6.3.3 Point3d geometry::cartesianToBarycentric (const Point3d & *pos*, const Point3d & *p1*, const Point3d & *p2*, const Point3d & *p3*)

Convert cartesian coordinates to barycentric coordinates in a triangle.

The triangle is defined by *p1*, *p2*, *p3* and the result is such that: *pos* = *result*[0]**p1* + *result*[1]**p2* + *result*[2]**p3*

Referenced by algorithms::TriangleSurface::position(), and algorithms::TriangleSurface::vector().

16.6.3.4 Matrix3d geometry::cartesianToBarycentric_matrix (const Point3d & *p1*, const Point3d & *p2*, const Point3d & *p3*, int & *index_one*)

Convert cartesian coordinates to barycentric coordinates in a triangle.

This version simply returns the matrix that convert from cartesian to barycentric. The resulting matrix has to be used like this:

```
Point3d pos(x,y,z);
int index_one;
Matrix3d mat = cartesianToBarycentric_matrix(p1, p2, p3, index_one);
pos[index_one] = 1;
Point3d bar = mat*pos;
```

Parameters

index_one Index of the position that has to be set to one.

16.6.3.5 Point2d geometry::centroid (const std::vector< Point2d > & *polygon*)

Center of mass of a 2D non-overlapping polygon.

16.6.3.6 bool geometry::lineLineIntersection (Point2d & *u*, const Point2d & *p1*, const Point2d & *u1*, const Point2d & *p2*, const Point2d & *u2*)

Find if the segment [*p1*,*p2*] intersect the line going through *p* of normal *n*.

Parameters

- *u* Intersection point if the function returns true, untouched otherwise
- ← *p1* Point on the first line
- ← *u1* Vector along the first line
- ← *p2* Point on the second line
- ← *u2* Vector along the second line

16.6.3.7 bool geometry::lineSegmentIntersection (Point2d & *u*, const Point2d & *p1*, const Point2d & *p2*, const Point2d & *p*, const Point2d & *n*)

Find if the segment [*p1*,*p2*] intersect the line going through *p* of normal *n*.

Parameters

- *u* Intersection point if the function returns true, untouched otherwise
- ← *p1* First point of the segment
- ← *p2* Second point of the segment
- ← *p* Point on the line
- ← *n* Normal vector to the line

16.6.3.8 `bool geometry::lineTriangleIntersection (Point3d & u, double & s, const Point3d & p1, const Point3d & p2, const Point3d & tr1, const Point3d & tr2, const Point3d & tr3)`

Line-triangle intersection.

Parameters

- *u* Intersection point if any
- *s* Position of the intersection on the line. Between 0 and 1, the intersection is on the segment. Greater than 0, the position is on the ray starting on *u1* and going through *u2*. If the line doesn't intersect the plane, *s* is valued to NaN.
- ← *p1* Starting point of the segment
- ← *p2* Ending point of the segment
- ← *tr1* First triangle point
- ← *tr2* Second triangle point
- ← *tr3* Third triangle point

16.6.3.9 `bool geometry::planeLineIntersection (Point3d & u, double & s, const Point3d & p, const Point3d & n, const Point3d & u1, const Point3d & u2)`

Plane-Line Intersection.

This function compute the intersection of the line [*u1*, *u2*] and the plane defined by the point *p* and the normal *n*.

Parameters

- *u* Intersection point if any
- *s* Position of the intersection on the line. Between 0 and 1, the intersection is on the segment. Greater than 0, the position is on the ray starting on *u1* and going through *u2*.
- ← *p* Point contained in the plane
- ← *n* Normal of the plane
- ← *u1* Starting point of the segment
- ← *u2* Ending point of the segment

Returns

True if there is an intersection.

Referenced by `tissue::findDivisionPoints()`, `cell_system::findDivisionPoints()`, and `vvcomplex::FindOppositeWall()`.

16.6.3.10 `bool geometry::pointInPolygon (const Point2d & p, const std::vector< Point2d > & polygon)`

Find if the point *p* is inside the polygon using the winding number.

16.6.3.11 `bool geometry::pointInTriangle (Point3d & u, double & s, const Point3d & p, const Point3d & p1, const Point3d & p2, const Point3d & p3)`

Point in triangle test (3D).

Parameters

u modified to the projection of the point in the triangle plane and

s the square distance from the point to the plane.

p Point to test

p1 First point of the triangle

p2 Second point of the triangle

p3 Third point of the triangle

Test if the point *p* is projected in the triangle (*p1*,*p2*,*p3*).

16.6.3.12 `bool geometry::pointInTriangle (const Point2d & p, const Point2d & p1, const Point2d & p2, const Point2d & p3)`

Point in triangle test (2D).

Test if the point *p* is in the triangle (*p1*,*p2*,*p3*)

Bug

Not working properly at all !

16.6.3.13 `template<size_t N> double geometry::polygonArea (const std::vector< util::Vector< N, double > > & polygon) [inline]`

Area of a planar polygon.

Definition at line 29 of file area.h.

```

30  {
31      size_t k = polygon.size(), h, l, i;
32      typename util::CrossProductType<N,double>::type surface = 0;
33      if(polygon.empty())
34          return 0;
35      h = (k-1)/2;
36      if(k%2)
37          l = 0;

```

```

38     else
39         l = k-1;
40     for(i = 1 ; i < h ; i++)
41         surface += (polygon[2*i] - polygon[0]) ^ (polygon[2*i+1] - polygon[2*i-1]);

42     surface += (polygon[2*h] - polygon[0]) ^ (polygon[1] - polygon[2*h-1]);
43     surface /= 2;
44     return util::norm(surface);
45 }
```

16.6.3.14 `template<size_t N> util::Vector<N,double>
 geometry::projectPointOnLine (const util::Vector< N, double > &pt,
 const util::Vector< N, double > &u1, const util::Vector< N, double
 > &u2, bool strict = false) [inline]`

Project a point on a line.

Note

Works whatever the dimension! Not restricted to 2d or 3d.

Parameters

pt Point to project

u1 Start of the line

u2 End of the line

strict If true, only the segment is considered, otherwise the whole line is

Returns

The coordinates of the projected point.

Definition at line 29 of file projection.h.

```

30 {
31     util::Vector<N,double> u = u2-u1;
32     util::Vector<N,double> pv = pt-u1;
33     double l = util::norm(u);
34     u /= l;
35     double proj_val = pv*u;
36     if(strict)
37     {
38         if(proj_val < 0)
39             proj_val = 0;
40         else if(proj_val > 1)
41             proj_val = 1;
42     }
43     return u1 + u*proj_val;
44 }
```

16.6.3.15 Point3d geometry::projectPointOnPlane (const Point3d & *pos*, const Point3d & *p*, const Point3d & *n*) [inline]

Project a point into a plane defined by a point and a normal to the plane.

Parameters

pos Point to project
p Point of the plane
n Normal to the plane

Definition at line 67 of file projection.h.

```
68 {  
69     Point3d res = pos - ((pos-p)*n)*n;  
70     return res;  
71 }
```

16.6.3.16 Point3d geometry::projectPointOnTriangle (const Point3d & *pt*, const Point3d & *p1*, const Point3d & *p2*, const Point3d & *p3*, double & *s*, bool * *in_triangle* = 0)

Project a point into a triangle.

Parameters

pt Point to project
p1 First point of the triangle
p2 Second point of the triangle
p3 Third point of the triangle
s Square of the distance from the triangle to the point
in_triangle If not null, set to true if the orthogonal projection is in the triangle, false otherwise.

Returns

The coordinate of the point of the triangle closest to *pt*.

Referenced by algorithms::TriangleSurface::position().

16.6.3.17 bool geometry::segmentSegmentIntersection (Point2d & *u*, const Point2d & *p1*, const Point2d & *p2*, const Point2d & *q1*, const Point2d & *q2*)

Find if two 2D segments intersect.

Based on "Faster line segment intersection." Graphics Gems III.

Parameters

u The actual intersection point, if any.

Warning

To be rewritten for numerical precision + check algo

16.6.3.18 `template<size_t N> double geometry::triangleArea (const
util::Vector< N, double > & a, const util::Vector< N, double > & b,
const util::Vector< N, double > & c) [inline]`

Area of a triangle in 2D or 3D.

Definition at line 20 of file area.h.

Referenced by `vvcomplex::findCenter()`, `tissue_model::TissueModel< RealModel, TissueClass >::updateCellsArea()`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::updateCellsArea()`, `cell_system::CellSystem< TissueClass, RealModel >::updateCellsArea()`, and `cell_system::volumeLeftCell()`.

```
21  {
22      return util::norm((a - b)^(a - c))/2.0;
23  }
```

16.7 graph Namespace Reference

Contains all the classes related to the graphs.

Classes

- class [_EmptyEdgeContent](#)
Empty class used as default for edge content.
- struct [Arc](#)
Type of a undirected edge (or arc).
- struct [CountedContent](#)
Type of the reference counted content.
- class [Edge](#)
Edge of a vv graph.
- class [Vertex](#)
Vertex of a vv graph.
- class [VVBigraph](#)
Class representing a VV graph.
- class [VVGraph](#)
Class representing a VV graph.
- class [WeakVertex](#)
Weak pointer on a vertex.

Typedefs

- typedef uintptr_t [edge_identity_t](#)
Type of the identifier of an edge.
- typedef uintptr_t [vertex_identity_t](#)
Type of the identifier of a vertex.

Functions

- template<typename T >
void [copy_symmetric](#) (T &e2, const T &e1)
Default copy of the arc uses the = operator.

- `template<typename BiGraph >`
`void create_bigraph_methods (BiGraph &G)`
- `template<typename Graph >`
`void create_graph_methods (Graph &G)`
- `template<typename T >`
`Edge< T > create_object (Edge< T > const &e)`
- `template<typename T >`
`Vertex< T > create_object (Vertex< T > const &v)`
- `intptr_t getNextGraphId ()`
- `template<TEMPLATE_VERTEX , typename charT >`
`std::basic_ostream< charT > & operator<< (std::basic_ostream< charT > &ss, const Vertex< VERTEX_ARGS > &v)`
- `template<typename V1 , typename V2 , typename E1 , typename E2 , bool compact>`
`bool serialization (storage::VVEStorage &store, VVBigraph< V1, V2, E1, E2, compact > &G)`
- `template<typename V , typename E , bool compact>`
`bool serialization (storage::VVEStorage &store, VVGraph< V, E, compact > &G)`
- `template<typename T >`
`bool serialization (storage::VVEStorage &store, Edge< T > &e)`
- `template<typename T >`
`bool serialization (storage::VVEStorage &store, Vertex< T > &v)`

Variables

- `size_t vertex_counter = 0`
Number used to enumerate the vertices of all types.

16.7.1 Detailed Description

Contains all the classes related to the graphs. For now, the only graph available is the [VVGraph](#).

16.7.2 Typedef Documentation

16.7.2.1 typedef uintptr_t graph::edge_identity_t

Type of the identifier of an edge.

Definition at line 22 of file edge.h.

16.7.2.2 typedef uintptr_t graph::vertex_identity_t

Type of the identifier of a vertex.

Definition at line 58 of file vertex.h.

16.7.3 Function Documentation

16.7.3.1 `template<typename T > void graph::copy_symmetric (T & e2, const T & e1) [inline]`

Default copy of the arc uses the = operator.

Overload this function for your own type to change this behavior.

Definition at line 256 of file edge.h.

Referenced by `graph::Arc< EdgeContent >::sync()`.

```
257     {  
258         e2 = e1;  
259     }
```

16.7.4 Variable Documentation

16.7.4.1 `size_t graph::vertex_counter = 0`

Number used to enumerate the vertices of all types.

This number exists only if `NO_NUMBER_VERTICES` is not defined at compile time.

Definition at line 6 of file vertex.cpp.

Referenced by `Model::Model()`.

16.8 parallel Namespace Reference

Define a thread pool for multi-threading computation.

Classes

- class [ThreadEval](#)
Base class for function evaluated in a thread.

Functions

- `template<class Arg1 , class Arg2 , class Arg3 , class Arg4 >`
`FreeFunction4< Arg1, Arg2, Arg3, Arg4 > freeFunction (void(*)(Arg1, Arg2, Arg3, Arg4))`
- `template<class Arg1 , class Arg2 , class Arg3 >`
`FreeFunction3< Arg1, Arg2, Arg3 > freeFunction (void(*)(Arg1, Arg2, Arg3))`
- `template<class Arg1 , class Arg2 >`
`FreeFunction2< Arg1, Arg2 > freeFunction (void(*)(Arg1, Arg2))`
- `template<class Arg1 >`
`FreeFunction1< Arg1 > freeFunction (void(*)(Arg1))`
- `template<class Class , typename Arg1 , typename Arg2 , typename Arg3 >`
`MemberFunction3< Class, Arg1, Arg2, Arg3 > memberFunction (void(Class::*m)(Arg1, Arg2, Arg3))`
- `template<class Class , typename Arg1 , typename Arg2 >`
`MemberFunction2< Class, Arg1, Arg2 > memberFunction (void(Class::*m)(Arg1, Arg2))`
- `template<class Class , typename Arg >`
`MemberFunction1< Class, Arg > memberFunction (void(Class::*m)(Arg))`
- `template<class Class >`
`MemberFunction0< Class > memberFunction (void(Class::*m)())`
- `template<typename Function >`
`void threadEval (const Function &fct, FIRST_ARG(Function) a1, SECOND_ARG(Function) a2, THIRD_ARG(Function) a3, FOURTH_ARG(Function) a4)`
- `template<typename Function >`
`void threadEval (const Function &fct, FIRST_ARG(Function) a1, SECOND_ARG(Function) a2, THIRD_ARG(Function) a3)`
- `template<typename Function >`
`void threadEval (const Function &fct, FIRST_ARG(Function) a1, SECOND_ARG(Function) a2)`
- `template<typename Function >`
`void threadEval (const Function &fct, ONLY_ARG(Function) a)`
- `template<typename Function >`
`void threadEval (const Function &fct)`

Evaluate a function without arguments using the thread pool.

- void **threadJoin** ()

16.8.1 Detailed Description

Define a thread pool for multi-threading computation.

16.8.2 Function Documentation

16.8.2.1 `template<typename Function > void parallel::threadEval (const
Function & fct) [inline]`

Evaluate a function without arguments using the thread pool.

16.9 shape Namespace Reference

Define a set of shape to be drawn using OpenGL.

Functions

- void [cube](#) (const Point3d ¢er, const Point3d &axis1, const Point3d &axis2, double size)

Draw a cube.

- void [cylinder](#) (const Point3d &base_center, const Point3d &axis, double radius_base, double radius_top, double length, int slices, int stacks, bool closed=true)

Draw a cylinder.

- void [disk](#) (const Point3d ¢er, const Point3d &normal, double inner_radius, double outer_radius, int slices, int loops)

Draw a disk.

- void [parallelepiped](#) (const Point3d &corner, const Point3d &edge1, const Point3d &edge2, const Point3d &edge3)

Draw a parallelepiped.

- void [sphere](#) (const Point3d ¢er, double radius, int slices, int stacks)

Draw a sphere.

16.9.1 Detailed Description

Define a set of shape to be drawn using OpenGL.

16.9.2 Function Documentation

16.9.2.1 void shape::cube (const Point3d & center, const Point3d & axis1, const Point3d & axis2, double size)

Draw a cube.

Parameters

center Center of the cube

axis1 Direction of a set of edges

axis2 The plane formed by (axis1,axis2) contains a set of faces

size Length of an edge

16.9.2.2 `void shape::cylinder (const Point3d & base_center, const Point3d & axis, double radius_base, double radius_top, double length, int slices, int stacks, bool closed = true)`

Draw a cylinder.

Parameters

base_center Position of the center of the base of the cylinder
axis Direction of the main axis of the cylinder
radius_base Radius of the base of the cylinder
radius_top Radius at the top of the cylinder
length Length of the cylinder
slices Number of slices used to draw the cylinder
stacks Number of stacks used to draw the cylinder
closed If true, two disks are drawn at the base and top of the cylinder

16.9.2.3 `void shape::disk (const Point3d & center, const Point3d & normal, double inner_radius, double outer_radius, int slices, int loops)`

Draw a disk.

Parameters

center Center of the disk
normal Vector normal to the surface of the disk
inner_radius Radius to start the disk, 0 for a full disk
outer_radius Radius of the disk
slices Number of slices used to draw the disk
loops Number of loops used to draw the disk

16.9.2.4 `void shape::parallelepiped (const Point3d & corner, const Point3d & edge1, const Point3d & edge2, const Point3d & edge3)`

Draw a parallelepiped.

Parameters

corner Position of one of the corner
edge1 Vector describing a set of edges (length and direction)
edge2 Vector describing a set of edges (length and direction)
edge3 Vector describing a set of edges (length and direction)

16.9.2.5 void shape::sphere (const Point3d & *center*, double *radius*, int *slices*, int *stacks*)

Draw a sphere.

Parameters

center Center of the sphere

radius Radius of the sphere

slices Number of slices used to create the sphere

stacks Number of stacks used to create the sphere

16.10 solver Namespace Reference

Implement a generic solver of ODE on a graph.

Classes

- class [Solver](#)

Implement a set of solvers for ODE on a graph.

Enumerations

- enum [SolvingMethod](#) {
[Euler](#), [AdaptiveEuler](#), [Fixedpoint](#), [Midpoint](#),
[RungeKutta](#), [AdaptiveRungeKutta](#), [AdaptiveCrankNicholson](#) }
Available solving methods.
- enum [ToleranceType](#) { [MAX_COMPONENT](#), [MEAN_COMPONENT](#) }
Type of the tolerances.

Functions

- `std::ostream & operator<< (std::ostream &s, const ToleranceType &m)`
- `std::ostream & operator<< (std::ostream &s, const SolvingMethod &m)`
- `QTextStream & operator<< (QTextStream &s, const SolvingMethod &m)`
- `std::istream & operator>> (std::istream &s, ToleranceType &m)`
- `std::istream & operator>> (std::istream &s, SolvingMethod &m)`
- `QTextStream & operator>> (QTextStream &s, SolvingMethod &m)`

16.10.1 Detailed Description

Implement a generic solver of ODE on a graph.

16.10.2 General information

This module implements a set of ODE solvers on a graph, from forward Euler to Cranck-Nicholson. It assumes a symmetric graph (i.e. for each vertex $v1$ and $v2$, if there is an edge from $v1$ to $v2$, there must be an edge from $v2$ to $v1$).

16.10.3 Module Usage

This module is used by constructing a `solver::Solver` object which holds the parameters for solving and the time stepping.

Most solvers requires an internal structure to be placed in the content of your vertexes and edge.

For example, to solve a 3-components system (i.e. 3 component on each vertex of the graph):

```
typedef solver::Solver<3> RDSolver;

struct VertexContent
{
    // ...
    RDSolver::VertexInternals interns;
};

struct EdgeContent
{
    // ...
    RDSolver::EdgeInternals interns;
};

class MyModel : public Model
{
public:
    RDSolver solve;
    typedef RDSolver::tag_t tag_t;
    vvgraph S;

    void step()
    {
        solve(S, *this);
    }
};
```

To use this module, the user needs to define some methods in its model class. Each class accept as argument a tag to differentiate between different solvers used in the same model. If you have different solvers with the same template arguments, then you can add an optional argument to differentiate them:

```
struct reaction_diffusion_solver {};
struct mass_spring_solver {};
typedef solver::Solver<2, reaction_diffusion_solver> RDSolver;
typedef solver::Solver<2, mass_spring_solver> MSSolver;

struct VertexContent
{
    // ...
    RDSolver::VertexInternals rd_interns;
    MSSolver::VertexInternals ms_interns;
};

struct EdgeContent
{
    // ...
    RDSolver::EdgeInternals rd_interns;
    MSSolver::EdgeInternals ms_interns;
};
```



```

class MyModel : public Model
{
    RDSolver rd_solve;
    typedef RDSolver::tag_t rd_tag_t;

    MSSolver ms_solve;
    typedef MSSolver::tag_t ms_tag_t;
};

```

Here are the methods to define in your model:

```

util::Vector<nb_chemicals,double>& values(const vertex& v, const tag_t& tag);

```

Returns a reference on the vector containing the values for the different components of the ODE on the vertex *v*.

```

util::Vector<nb_vars,double>& derivatives(const vertex& v, const tag_t& tag);

```

Returns a reference on the vector containing the derivatives for the different components of the ODE on the vertex *v*.

```

void updateDerivatives(const vertex& v, const tag_t& tag);

```

Update the time derivatives of the components on the vertex *v*. You must not change any data outside the vertex *v*. Plus, you should not assume any order in the evaluation of this function. You might even assume all the `updateDerivatives` may be evaluated in parallel (I hope it will eventually by the case).

```

Solver::VertexInternals& vertexInternals(const vertex& v, const tag_t& tag);

```

This method is not used for the forward Euler method. For all other you have to implement it. It means also that your vertex has to contain an instance `Solver::VertexInternals`. This method should return a reference on that structure.

```

Solver::EdgeInternals& edgeInternals(const vertex& src, const vertex& tgt, graph
    & S, const tag_t& tag);

```

This method is only used in the Cranck-Nicholson solver, so if you do not intent on using it, you don't need edge internals. Note however that if you let the user choose, you have to implement ALL the methods, even if you never use them. It should return a reference on the `edgeInternals` structure for the solver.

16.10.4 Enumeration Type Documentation

16.10.4.1 enum solver::SolvingMethod

Available solving methods.

There is stream operator defined to read and write the methods with the exact same name

Enumerator:

Euler Forward euler method.

AdaptiveEuler Adaptive forward euler method.

Fixedpoint Fixed point method (i.e. find the solution for the derivatives = 0).

Midpoint Mid-point method (i.e. order 2 Runge-Kutta).

RungeKutta Order 4 Runge-Kutta.

AdaptiveRungeKutta Adaptative order 4 Runge-Kutta.

AdaptiveCrankNicholson Adaptative Crank-Nicholson.

Definition at line 218 of file solver.h.

```

219  {
221      Euler,
223      AdaptiveEuler,
225      Fixedpoint,
227      Midpoint,
229      RungeKutta,
231      AdaptiveRungeKutta,
233      AdaptiveCrankNicholson
234  };

```

16.10.4.2 enum solver::ToleranceType

Type of the tolerances.

A stream operator define the read/write methods

Enumerator:

MAX_COMPONENT Extract the maximum value to compute the precision (name: MaxComponent).

MEAN_COMPONENT Extract the mean of the values to compute the precision (name: MeanComponent).

Definition at line 346 of file solver.h.

```

347  {
349      MAX_COMPONENT,
351      MEAN_COMPONENT
352  };

```

16.11 std Namespace Reference

STL namespace.

Functions

- `template<typename T >`
`const T & begin (const pair< T, T > &p)`
- `template<typename T >`
`const T & end (const pair< T, T > &p)`
- `Quaternion pow (const Quaternion &q, double p)`
- `template<typename T >`
`bool serialization (storage::VVEStorage &store, list< T > &vec)`
- `template<typename T, typename Sort >`
`bool serialization (storage::VVEStorage &store, set< T, Sort > &vec)`
- `template<typename T, typename Sort >`
`bool serialization (storage::VVEStorage &store, multiset< T, Sort > &vec)`
- `template<typename T >`
`bool serialization (storage::VVEStorage &store, vector< T > &vec)`
- `template<GRAPH_TEMPLATE >`
`void swap (graph::VVGraph< GRAPH_ARGS > &g1, graph::VVGraph< GRAPH_ARGS > &g2)`
Specialize the swap algorithm for two graphs.
- `template<BIGRAPH_TEMPLATE >`
`void swap (graph::VVBGraph< BIGRAPH_ARGS > &g1, graph::VVBGraph< BIGRAPH_ARGS > &g2)`

16.11.1 Detailed Description

STL namespace.

16.11.2 Function Documentation

16.11.2.1 `template<GRAPH_TEMPLATE > void std::swap` `(graph::VVGraph< GRAPH_ARGS > &g1, graph::VVGraph< GRAPH_ARGS > &g2) [inline]`

Specialize the swap algorithm for two graphs.

Definition at line 2910 of file `vvgraph.h`.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::swap()`.

```
2911     {
2912         g1.swap(g2);
2913     }
```

16.12 storage Namespace Reference

Namespace containing all functions and classes related to VVE persistence.

Classes

- struct [ConvertType](#)
Convert object of type from to object of type to.
- struct [Frame](#)
Data holding properties of a frame.
- struct [TypeTraits](#)
Defines main traits for types directly handled.
- class [VVEStorage](#)
Abstract base class defining the interactions with the persistence module.
- class [VVEStorage_BINReader](#)
Implementation reading an BIN file.
- class [VVEStorage_BINWriter](#)
Implementation writing an BIN file.
- class [VVEStorage_XMLReader](#)
Implementation reading an XML file.
- class [VVEStorage_XMLWriter](#)
Implementation writing an XML file.

Enumerations

- enum **BIN_TYPES** {
BT_BOOL = 0, **BT_WCHART**, **BT_CHAR**, **BT_SIGNED_CHAR**,
BT_UNSIGNED_CHAR, **BT_SHORT**, **BT_UNSIGNED_SHORT**, **BT_-**
INT,
BT_UNSIGNED_INT, **BT_LONG**, **BT_UNSIGNED_LONG**, **BT_LONG_-**
LONG,
BT_UNSIGNED_LONG_LONG, **BT_FLOAT**, **BT_DOUBLE**, **BT_-**
LONG_DOUBLE,
BT_STRING, **BT_START_REFERENCE**, **BT_START_COMPOUND** }

- enum **NUMBER_CLASS** {
SIGNED_INTEGER, **UNSIGNED_INTEGER**, **FLOATING_POINT**,
CHAR,
NOT_A_NUMBER }
- enum **Options** { **TypeChecking** }
Common options for reader/writer.
- enum **STORAGE_ERROR** {
NO_ERROR = 0, **IO_ERROR**, **BAD_CONTENT**, **TYPE_CHECK_ERROR**,
TYPE_CONVERSION_ERROR, **NO_FIELD_ERROR**, **REFERENCE_**-
ERROR, **USER_ERROR**,
UNKNOWN_ERROR }
Enumeration describing the basic errors.
- enum **TypeCheckingOption** {
TCO_Exact, **TCO_Strict**, **TCO_Permissive**, **TCO_PermissiveNoWarning**,
TCO_NoCheck }
Enumeration describing how the type of the elements should be handled.
- enum **TYPES** {
T_INVALID = -1, **T_BOOL** = 0, **T_WCHART**, **T_CHAR**,
T_SIGNED_CHAR, **T_UNSIGNED_CHAR**, **T_SHORT**, **T_UNSIGNED_**-
SHORT,
T_INT, **T_UNSIGNED_INT**, **T_LONG**, **T_UNSIGNED_LONG**,
T_LONG_LONG, **T_UNSIGNED_LONG_LONG**, **T_FLOAT**, **T_**-
DOUBLE,
T_LONG_DOUBLE, **T_STRING** }

Functions

- template<typename From , typename To >
bool **convert_type** (const From &from, To &to)
Function converting from to to.
- template<typename Ptr >
Ptr **create_object** (Ptr const &p)
- **TYPES** **find_type** (**NUMBER_CLASS** klass, size_t bytes)
- **FOR_ALL_TYPES_NOSTRING** (**CONVERT_FROM_STDSTRING**)
- **FOR_ALL_TYPES_NOSTRING** (**CONVERT_TO_STDSTRING**)
- bool **isStrictConversion** (**TYPES** from, **TYPES** to)
- template<typename Ptr >
ReferencePtr **make_reference** (Ptr &p)
- **TYPES** **read_type** (**QFile** *file)

- `template<typename T >`
`bool serialization (VVEStorage &storage, T &value)`
Function implementing the serialization.
- `template<typename T >`
`TYPES type_id (const T &=T())`
- `template<typename T >`
`const QString & type_name (const T &=T())`
- `const QString & typeid_to_name (TYPES id)`
- `TYPES typename_to_id (const QString &name)`
- `int versionNumber (const QString &file_version)`
Helper function to parse version number.

16.12.1 Detailed Description

Namespace containing all functions and classes related to VVE persistence.

16.12.2 Principle

The `storage` works using the `storage::VVEStorage` objects. This abstract class defines the generic behavior of a serialized `storage` for VVE. The implementations are created by the interpreter and given to the model. The `storage` implement an interface that is the same for reading and writing.

16.12.3 Defining the serialization

To allow for serialization the model should implement at least one method:

```
bool serialize(storage::VVEStorage& store);
```

This method actually implements the serialization of the model. It should returns true if the serialization worked, false otherwise. To test for the existence of this method, the interpreter will call it using a dummy storage object, simulating a storage writer always succeeding. If the result of this dummy call is true, then the object is recognised as being able to serialize data.

For every object, other than the model itself, you want to serialize, you can define its serialization in two ways:

1. via a method in the object:

```
bool serialize(storage::VVEStorage& store);
```

2. via an external function:

```
bool serialization(storage::VVEStorage& store, MyClass& object);
```

The first way is more intrusive but has the advantage to be inherited. The second way is non-intrusive but won't be inherited by sub-classes. In case the `serialization` method does not exist, the `serialize` method will be looked for.

16.12.3.1 File versionning

Optionnally, the model can define a version string. To do so, the class should define the method:

```
QString version() const;
```

The method will be called when saving a file.

Once the version is obtained (either from a file or by calling the `version` method), the `storage` object will call the method:

```
int versionNumber(const QString& version);
```

the version given is the model or file version prepended with the file format and a hyphen. For example, if the version of a model is 1.2.3 and the current file is an XML one, the version will be: XML-1.2.3

The number returned by the `versionNumber()` method is stored by the `storage` object and can be retrieved using the `VVEStorage::version()` method. As a helper, you can use the function `storage::versionNumber(const QString&)`.

16.12.4 Using the VVEStorage object

The `VVEStorage` object allows you to define your data structure as a hierarchical set of named fields. As the system is a serialization one, fields should always be read and written in the same order. The `storage` object is not a random access object but a serial one.

To help you structure your data, you can create three kind of elements:

1. **Fields** This is the basic type. If you want to store an element by value, you use a field. All the basic C++ types and strings are handled directly by the `storage` object. Any other type must have either a `serialization` function associated or a `serialize` method. If a field contains an object, a compound will actually be created. A field is created by the method:

```
template <typename T>
bool VVEStorage::field(const QString& name, T& value);
```

The name must not be more than 255 characters but can be empty. An empty field is valid only as sole member of a compound. In this case, the `storage` object can decide to simply create a field reusing the name of the compound (this is actually done both for the XML and BIN implementation). A typical exemple would be for a class with only one member:

```
struct MyStructure
{
    int value;
    bool serialize(storage::VVEStorage& store) {
        return store.field("", value);
    }
};
```

2. **References** References are similar to fields but are used for pointers. The element stored should be an actual pointer or a smart pointer. A reference is created with the method:

```
template <typename Ptr>
bool VVEStorage::reference(const QString& name, const QString& ref_type, Ptr& p
);
```

The point is using a reference is that:

- (a) if the same object is saved twice, it will be only once in the file
- (b) if two pointers share the same object when writing, they will be reconstructed as pointing to the same object.

A typical example in VVE is the vertices. If you store a vertex, the serialization function defined in `<storage/graph.h>` is defined as:

```
bool serialization(storage::VVEStorage& store, vertex& v) {
    return store.reference("", "Vertex", v);
}
```

The empty name tells the `storage` object that it can reuse the compound name (and actually not have a compound but just a reference). Then, the type is useful mostly for user-editable files where the reference ids are unique per type. Then, if you use the same vertex twice, only one instance will be stored and recreated later. And the two vertices will still share the same data. Once a reference is identified, a namesless field is created to hold the content of the reference.

3. Compounds Compounds are created using the two methods:

```
bool VVEStorage::startCompound(const QString& name);
bool VVEStorage::endCompound();
```

This will group all element defined in between the two. A compound should always be closed. As for references and fields, a compound can be nameless and can then be optimized. In the end, a compound is rarely useful unless you have a very complicated structure. It is used for example in the VVGraph serialization. And of course, a compound is implicitly created for every composed object.

16.12.5 Enumeration Type Documentation

16.12.5.1 enum storage::Options

Common options for reader/writer.

Enumerator:

TypeChecking Change the way type is checked while extracting a structure.

Definition at line 185 of file storage.h.

```
186 {
187     TypeChecking
188 };
```


16.12.5.2 enum storage::STORAGE_ERROR

Enumeration describing the basic errors.

Enumerator:

NO__ERROR No error occurred. The wierd name is due to the definition of NO_ERROR and NOERROR by windows already. Any complain should be sent to Microsoft.

IO_ERROR Error occuring during an IO access. Typically while reading or writing a file.

BAD_CONTENT Bad content error. This mean the format of the file is not correct. For example, an XML file does not respect the XML format.

TYPE_CHECK_ERROR Type error. It can occur either if the typing is strict and the type is different in the file and in the model or if the typing is permissive but the conversion does not exist.

TYPE_CONVERSION_ERROR Type conversion error. It can occur if the type conversion should be possible but the content is invalid. The typical case is the conversion from string to number if the string doesn't contain a number.

NO_FIELD_ERROR The field requested doesn't exist.

REFERENCE_ERROR Error while using a reference. Either the reference does not exist or the user try to access a non-reference as reference.

USER_ERROR Error of the user. The succession of commands of the user are invalid.

UNKNOWN_ERROR Error set if the serialize method fails without an error being set.

Definition at line 226 of file storage.h.

```
227 {  
233     NO__ERROR = 0,  
239     IO_ERROR,  
246     BAD_CONTENT,  
254     TYPE_CHECK_ERROR,  
262     TYPE_CONVERSION_ERROR,  
266     NO_FIELD_ERROR,  
273     REFERENCE_ERROR,  
279     USER_ERROR,  
283     UNKNOWN_ERROR  
284 };
```

16.12.5.3 enum storage::TypeCheckingOption

Enumeration describing how the type of the elements should be handled.

Enumerator:

TCO_Exact The type asked for must be the same as the one read.

TCO_Strict The conversion should be loss-less. For example, you'll be able to convert from short to int but not the other way around

TCO_Permissive The conversion can induce loss. However, any loss of precision conversion will lead to a warning in the standard output.

TCO_PermissiveNoWarning Same as Permissive, but without the warnings.

TCO_NoCheck No check is performed. An error will be raised only if the string cannot be interpreted as the type claimed.

Definition at line 194 of file storage.h.

```

195  {
197      TCO_Exact,
204      TCO_Strict,
211      TCO_Permissive,
215      TCO_PermissiveNoWarning,
220      TCO_NoCheck
221  };

```

16.12.6 Function Documentation

16.12.6.1 `template<typename From , typename To > bool
storage::convert_type (const From &from, To &to) [inline]`

Function converting from to to.

This allows custom (compile-time) conversion.

Definition at line 202 of file types.h.

```

203  {
204      static const ConvertType<From,To> converter = ConvertType<From,To>();
205      return converter(from, to);
206  }

```

16.12.6.2 `TYPES storage::find_type (NUMBER_CLASS klass, size_t bytes)`

Returns

a type corresponding to the number class and size.

If no such type exist, return T_INVALID

Referenced by storage::VVEStorage_BINReader::serialize(), and
storage::old::VVEStorage_BINReader::serialize().

16.12.6.3 `bool storage::isStrictConversion (TYPES from, TYPES to)`

Returns

True if you can convert from from to to without losing any information

16.12.6.4 `template<typename T > bool storage::serialization (VVEStorage & storage, T & value) [inline]`

Function implementing the serialization.

The default implementation expects a serialize method in the type:

```
bool T::serialize(VVEStorage&)
```

Definition at line 703 of file storage.h.

Referenced by storage::VVEStorage::field().

```
704 {
705     return value.serialize(storage);
706 }
```

16.12.6.5 `const QString & storage::typeid_to_name (TYPES id)`

Returns

the name associated to the type id

16.12.6.6 `TYPES storage::typename_to_id (const QString & name)`

Returns

the type id associated to the name

16.12.6.7 `int storage::versionNumber (const QString & file_version)`

Helper function to parse version number.

Returns

the version number of 0 if no number found.

This function will find any version on the form MAJOR[.MINOR[.MICRO]] and will return a unique integer on 32 bits supposing MAJOR and MINOR are less than 256 and micro is less than 65536. If the version numbers are too big, only the least significant bits are kept.

Any string other than the version number will be ignored.

Examples of valid versions:

File Version	Version	Number
VVE 1.2.20	1.2.20	0x01020014
XML 1.3	1.3	0x01030000
MyModel1.3.2a	1.3.2	0x01030002
OBJ 12.32.254	26.32.254	0x1a2000fe
FILE 12	12	0x0c000000

16.13 `template_utils` Namespace Reference

Namespace for meta-programming utils using templates.

16.13.1 Detailed Description

Namespace for meta-programming utils using templates.

16.14 test Namespace Reference

Namespace containing test facilities.

Functions

- bool **check** (bool tst, size_t line_nb, const char *tst_str)
- template<typename Complex >
bool **check_consistentComplex** (const Complex &T, size_t line_nb)
- template<typename T1 , typename T2 >
bool **check_equal** (const T1 &tst, const T2 &value, size_t line_nb, const char *tst_str, const char *value_str)
- template<typename T1 , typename T2 , typename T3 >
bool **check_float** (const T1 &tst, const T2 &value, const T3 &epsilon, size_t line_nb, const char *tst_str, const char *value_str)
- template<typename T1 , typename T2 , typename T3 >
bool **check_float_diff** (const T1 &tst, const T2 &value, const T3 &epsilon, size_t line_nb, const char *tst_str, const char *value_str)
- template<typename BiGraph >
bool **check_neighborsBiGraph** (const BiGraph &G, size_t line_nb)
- template<typename Graph >
bool **check_neighborsGraph** (const Graph &G, size_t line_nb)
- template<typename T1 , typename T2 >
bool **check_notequal** (const T1 &tst, const T2 &value, size_t line_nb, const char *tst_str, const char *value_str)
- template<typename BiGraph >
bool **check_symmetricBiGraph** (const BiGraph &G, size_t line_nb)
- template<typename Graph >
bool **check_symmetricGraph** (const Graph &G, size_t line_nb)
- template<typename T1 , typename T2 >
bool **check_zero** (const T1 &tst, const T2 &epsilon, size_t line_nb, const char *tst_str)
- void **resetResult** ()
- template<size_t N, typename T >
void **shuffle** (T array[])
- template<typename VertexContent >
QString **toStr** (const [graph::Vertex](#)< VertexContent > &v)

Variables

- QTextStream out (stdout)
- bool result = true

16.14.1 Detailed Description

Namespace containing test facilities.

16.15 tissue Namespace Reference

The tissue module handle cell division in a cell tissue.

Classes

- struct [CellPinchingParams](#)
Parameters of the cell pinching algorithm.
- struct [ClosestMidAlgoParams](#)
Parameters for the closest mid.
- struct [ClosestWallAlgoParams](#)
Parameters for the closest wall algorithm.
- struct [ShortWallAlgoParams](#)
Parameters for the shortest wall algorithm.
- class [Tissue](#)
Class handling the development and representation of a cell tissue.

Typedefs

- typedef [Tissue::cell_graph](#) [cell_graph](#)
Documentation alias for the type of the cell graph.
- typedef [Tissue::tissue_graph](#) [tissue_graph](#)
Documentation alias for the type of the tissue graph.
- typedef [Tissue::vertex](#) [vertex](#)
Documentation alias for the type of a graph vertex.

Enumerations

- enum [CELL_DIVISION_ALGORITHM](#) { [CLOSEST_MID](#), [SHORT_WALL](#), [CLOSEST_WALL](#) }
Enumeration of the provided division algorithms.

Functions

- `template<class Complex >
void cellPinching (const typename Complex::cell &c, Complex &T, DivisionData< typename Complex::junction_content > &data, const CellPinchingParams ¶ms)`
Pinching algorithm.
- `template<class Complex >
DivisionData< typename Complex::junction_content > findDivisionPoints (const typename Complex::cell &c, Complex &T, const ClosestWallAlgoParams ¶ms)`
Implementation of the closest wall algorithm.
- `template<class Complex >
DivisionData< typename Complex::junction_content > findDivisionPoints (const typename Complex::cell &c, Complex &T, const ClosestMidAlgoParams ¶ms)`
Implementation of the closest mid algorithm.
- `template<class Complex >
DivisionData< typename Complex::junction_content > findDivisionPoints (const typename Complex::cell &c, Complex &T, const ShortWallAlgoParams ¶ms)`
Implementation of the shortest wall algorithm.

16.15.1 Detailed Description

The tissue module handle cell division in a cell tissue. Compared to the 2d cell complex module, this module provides:

- Drawing function for the tissue
- Division algorithms

16.15.2 Typedef Documentation

16.15.2.1 `typedef Tissue::cell_graph tissue::cell_graph`

Documentation alias for the type of the cell graph.

Definition at line 63 of file tissue.h.

16.15.2.2 `typedef Tissue::tissue_graph tissue::tissue_graph`

Documentation alias for the type of the tissue graph.

Definition at line 67 of file tissue.h.

16.15.2.3 typedef Tissue::vertex tissue::vertex

Documentation alias for the type of a graph vertex.

Definition at line 59 of file tissue.h.

16.15.3 Enumeration Type Documentation

16.15.3.1 enum tissue::CELL_DIVISION_ALGORITHM

Enumeration of the provided division algorithms.

Definition at line 48 of file tissue.h.

```

49  {
50      CLOSEST_MID,
51      SHORT_WALL,
52      CLOSEST_WALL
53  };

```

16.15.4 Function Documentation

16.15.4.1 template<class Complex > void tissue::cellPinching (const typename Complex::cell & *c*, Complex & *T*, DivisionData< typename Complex::junction_content > & *data*, const CellPinchingParams & *params*) [inline]

Pinching algorithm.

This function displace the position of the newly inserted vertexes to account for the pinching of the cell. The pinching is always in the direction of the newly formed wall. A vertex on the side of the tissue won't be moved. Otherwise, the pinching is proportional to the distance to the closest extremity of the cut segment.

Parameters

- ← *c* Cell to be divided
- ← *S* Tissue graph
- ↔ *data* Parameters for the division to be updated
- ← *params* Parameters for the pinching
- ← *model* [Model](#) using the algorithm

Definition at line 117 of file tissue.h.

References [tissue::CellPinchingParams::cellMaxPinch](#), [tissue::CellPinchingParams::cellPinch](#), [vvcomplex::DivisionData< JunctionContent >::pu](#), [vvcomplex::DivisionData< JunctionContent >::pv](#), [vvcomplex::DivisionData< JunctionContent >::u1](#), and [vvcomplex::DivisionData< JunctionContent >::v1](#).

Referenced by [findDivisionPoints\(\)](#), and [cell_system::findDivisionPoints\(\)](#).


```
121 {
122     typedef typename Complex::cell cell;
123     typedef typename Complex::junction junction;
124     typename Complex::model_t& model = *T.model;
125     // Calculate cell pinch
126     double mind = 0.0;
127     double pinch = 0.0;
128
129     junction p_u1 = data.u1;
130     junction p_u2 = T.S.nextTo(c, p_u1);
131
132     junction p_v1 = data.v1;
133     junction p_v2 = T.S.nextTo(c, p_v1);
134
135     Point3d u1 = model.position(p_u1);
136     Point3d u2 = model.position(p_u2);
137
138     // If on edge, don't pinch
139     if(!T.border(p_u1, p_u2))
140     {
141         Point3d uc = (data.pv - data.pu);
142         double ucl = norm(uc);
143         uc /= ucl;
144         double uu1 = norm(data.pu-u1);
145         double uu2 = norm(data.pu-u2);
146         if(uu1 > uu2)
147             mind = uu2;
148         else
149             mind = uu1;
150         pinch = mind * ucl * params.cellPinch;
151         if(pinch > params.cellMaxPinch)
152             pinch = params.cellMaxPinch;
153         data.pu += uc * pinch;
154     }
155
156     Point3d v1 = model.position(p_v1);
157     Point3d v2 = model.position(p_v2);
158
159     if(!T.border(p_v1, p_v2))
160     {
161         Point3d vc = (data.pu - data.pv);
162         double vcl = norm(vc);
163         vc /= vcl;
164         double vv1 = norm(data.pv-v1);
165         double vv2 = norm(data.pv-v2);
166         if(vv1 > vv2)
167             mind = vv2;
168         else
169             mind = vv1;
170         pinch = mind * vcl * params.cellPinch;
171         if(pinch > params.cellMaxPinch)
172             pinch = params.cellMaxPinch;
173         data.pv += vc * pinch;
174     }
175 }
```

16.15.4.2 `template<class Complex > DivisionData<typename Complex::junction_content> tissue::findDivisionPoints (const typename Complex::cell & c, Complex & T, const ClosestWallAlgoParams & params) [inline]`

Implementation of the closest wall algorithm.

This algorithm defining the new wall as the one starting from the point on the walls the closest to the cell center and going through the cell center.

Parameters

c Cell to divide
S Tissue graph
params Parameters
model [Model](#) using the tissue

Returns

The parameters for the division

See also

[ClosestWallAlgoParams](#)

Definition at line 459 of file tissue.h.

References `cellPinching()`, `tissue::ClosestWallAlgoParams::cellWallMin`, `bspline_tissue_model::epsilon`, `vvcomplex::FindOppositeWall()`, `forall`, `IMPORT_COMPLEX_MODEL`, `IMPORT_COMPLEX_VERTICES`, `vvcomplex::DivisionData< JunctionContent >::pu`, `tissue::ClosestWallAlgoParams::strictCellWallMin`, `vvcomplex::testDivisionOnVertices()`, and `vvcomplex::DivisionData< JunctionContent >::u1`.

```

462 {
463     IMPORT_COMPLEX_VERTICES(Complex);
464     IMPORT_COMPLEX_MODEL(Complex, T);
465     DivisionData<typename Complex::junction_content> result;
466     // p_u1-u2, first edge, p_v1-v2 opposite edge
467     Point3d cpos = model.position(c);
468
469     // Find center (in case growth has changed it)
470     //FindCenter(c, S, model);
471
472     // Find closest point on wall
473     double mindis = 1000000;
474     double epsilon = 0;
475     forall( const junction& tu2, T.S.neighbors(c))
476     {
477         const junction& tu1 = T.S.prevTo(c, tu2);
478         Point3d u1 = model.position(tu1);
479         Point3d u2 = model.position(tu2);
480         Point3d ulu2 = u2 - u1;
481         double ulu2l = norm(ulu2);

```

```

482     ulu2 /= ulu2l;
483     epsilon = max(epsilon, VVE_REL_EPSILON*ulu2l);
484     Point3d u = u1 + ((cpos - u1) * ulu2) * ulu2;
485     if(FindWallMin(u, u1, u2, params.cellWallMin) or not params.strictCellWallM
in)
486     {
487         double dis = norm(u-cpos);
488         if(dis < mindis)
489         {
490             mindis = dis;
491             result.pu = u;
492             result.u1 = tu1;
493         }
494     }
495 }
496
497 FindOppositeWall(c, result, T, params.cellWallMin, params.strictCellWallMin);

498
499 if(params.cellWallMin == 0)
500     testDivisionOnVertices(c, result, T, epsilon);
501
502 cellPinching(c, T, result, params);
503 return result;
504 }
```

16.15.4.3 `template<class Complex > DivisionData<typename Complex::junction_content> tissue::findDivisionPoints(const typename Complex::cell & c, Complex & T, const ClosestMidAlgoParams & params) [inline]`

Implementation of the closest mid algorithm.

This algorithm defining the new wall as the one starting from the mid wall the closest to the cell center and going through the cell center.

Parameters

c Cell to divide
S Tissue graph
params Parameters
model [Model](#) using the tissue

Returns

The parameters for the division

See also

[ClosestMidAlgoParams](#)

Definition at line 406 of file tissue.h.

References `cellPinching()`, `tissue::ClosestMidAlgoParams::cellWallMin`, `vvcomplex::FindOppositeWall()`, `forall`, `IMPORT_COMPLEX_MODEL`, `IMPORT_COMPLEX_VERTICES`, `vvcomplex::DivisionData< JunctionContent >::pu`,

tissue::ClosestMidAlgoParams::strictCellWallMin, and vvcomplex::DivisionData< JunctionContent >::u1.

```

409 {
410     IMPORT_COMPLEX_VERTICES(Complex);
411     IMPORT_COMPLEX_MODEL(Complex, T);
412     DivisionData<typename Complex::junction_content> result;
413     // p_u1-u2, first edge, p_v1-v2 opposite edge
414     Point3d cpos = model.position(c);
415
416     // Find center (in case growth has changed it)
417     //FindCenter(c, S, model);
418
419     // Find closest midpoint on wall
420     double mindis = HUGE_VAL;
421     forall(const junction& tu2, T.S.neighbors(c))
422     {
423         // other end of wall
424         const junction& tu1 = T.S.prevTo(c, tu2);
425         Point3d u1 = model.position(tu1);
426         Point3d u2 = model.position(tu2);
427         Point3d u = (u1 + u2)/2.0;
428         double dis = norm(u-cpos);
429         if(dis < mindis)
430         {
431             mindis = dis;
432             result.pu = u;
433             result.u1 = tu1;
434         }
435     }
436
437     FindOppositeWall(c, result, T, params.cellWallMin, params.strictCellWallMin);
438
439     cellPinching(c, T, result, params);
440     return result;
441 }
```

16.15.4.4 `template<class Complex > DivisionData<typename Complex::junction_content> tissue::findDivisionPoints(const typename Complex::cell & c, Complex & T, const ShortWallAlgoParams & params) [inline]`

Implementation of the shortest wall algorithm.

This algorithm search for the shortest segment traversing the cell and going through the cell center.

Parameters

- c* Cell to divide
- S* Tissue graph
- params* Parameters
- model* [Model](#) using the tissue

Returns

The parameters for the division

See also

[ShortWallAlgoParams](#)

Definition at line 307 of file tissue.h.

References `cellPinching()`, `tissue::ShortWallAlgoParams::cellWallMin`, `tissue::ShortWallAlgoParams::cellWallSample`, `tissue::ShortWallAlgoParams::dx`, `bspline_tissue_model::epsilon`, `forall`, `IMPORT_COMPLEX_MODEL`, `IMPORT_COMPLEX_VERTICES`, `geometry::planeLineIntersection()`, `vvcomplex::DivisionData< JunctionContent >::pu`, `vvcomplex::DivisionData< JunctionContent >::pv`, `tissue::ShortWallAlgoParams::strictCellWallMin`, `vvcomplex::testDivisionOnVertices()`, `vvcomplex::DivisionData< JunctionContent >::u1`, and `vvcomplex::DivisionData< JunctionContent >::v1`.

```

310 {
311     IMPORT_COMPLEX_VERTICES(Complex);
312     IMPORT_COMPLEX_MODEL(Complex, T);
313     DivisionData<typename Complex::junction_content> result;
314     // p_u1-u2, first edge, p_v1-v2 opposite edge
315     Point3d minu, minv;
316
317     // Find center (in case growth has changed it)
318     //FindCenter(c, S, model);
319
320     Point3d cpos = model.position(c);
321
322     // Find shortest line through center
323     double mindis = HUGE_VAL;
324     Point3d n;
325     n = model.normal(c);
326     double epsilon = 0;
327     forall(const junction& tu1, T.S.neighbors(c))
328     {
329         const junction& tu2 = T.S.nextTo(c, tu1);
330         Point3d u1 = model.position(tu1);
331         Point3d u2 = model.position(tu2);
332         Point3d ulu2 = u2 - u1;
333         double ulu2l = norm(ulu2);
334         ulu2 /= ulu2l;
335         epsilon = max(epsilon, VVE_REL_EPSILON*ulu2l);
336         forall(const junction& tv1, T.S.neighbors(c))
337         {
338             if(tv1 == tu1)
339             {
340                 continue;
341             }
342             const junction& tv2 = T.S.nextTo(c, tv1);
343             const Point3d& v1 = model.position(tv1);
344             const Point3d& v2 = model.position(tv2);
345
346             double ule = ulu2l - params.cellWallMin;
347             for(double ul = params.cellWallMin; ul < ule; ul += params.cellWallSample
348         )
349         {
350             Point3d u = u1 + ul * ulu2;

```

```

350         Point3d uc = cpos - u;
351         Point3d nu = n^uc;
352         Point3d v;
353         double s;
354         if(geometry::planeLineIntersection(v, s, u, nu, v1, v2))
355         {
356             if(s >= -params.dx && s <= (1.0 + params.dx))
357             {
358                 if(FindWallMin(v, v1, v2, params.cellWallMin) or not params.strictC
ellWallMin)
359                 {
360                     double dis = norm(u-v);
361                     if(dis < mindis and (u-cpos)*(v-cpos) < 0)
362                     {
363                         mindis = dis;
364                         minu = u;
365                         minv = v;
366                         result.ul = tul;
367                         result.vl = tvl;
368                     }
369                 }
370             }
371         }
372     } // Samples
373 } // v1v2
374 } // ulu2
375 if(mindis == HUGE_VAL)
376 {
377     util::out << "findDivisionPoints::Error:Failed to find divide line" << endl
;
378     return DivisionData<typename Complex::junction_content>();
379 }
380 result.pu = minu;
381 result.pv = minv;
382
383 if(params.cellWallMin == 0)
384     testDivisionOnVertices(c, result, T, epsilon);
385
386 cellPinching(c, T, result, params);
387 return result;
388 }
```

16.16 tissue_model Namespace Reference

Namespace containing the base class for the [tissue](#) model helper.

Classes

- struct [TissueModel](#)
Base class for the [tissue_model](#) helper.

Variables

- const double [epsilon](#) = 0.0001
Relative constant for geometrical error.

16.16.1 Detailed Description

Namespace containing the base class for the [tissue](#) model helper.

16.16.2 Variable Documentation

16.16.2.1 const double tissue_model::epsilon = 0.0001

Relative constant for geometrical error.

Definition at line 174 of file tissue_model.h.

Referenced by `tissue_model::TissueModel< RealModel, TissueClass >::updateCellsArea()`.

16.17 util Namespace Reference

Various utility classes for generic programming.

Namespaces

- namespace [Shapes](#)
This namespace contains various shapes.

Classes

- class [Buffer](#)
A ranged checked array.
- class [CircIterator](#)
Creates a circular iterator from a range of forward iterators.
- class [Color](#)
A utility class to encapsulate color data.
- class [Contour](#)
[Contour](#) utility class.
- struct [CrossProductType< 2, T >](#)
For 2d -> a scalar.
- struct [CrossProductType< 3, T >](#)
For 3d -> a vector.
- class [FileObject](#)
This class is the base class for any object that might be watched by the [WatchDog](#).
- class [Function](#)
A utility class for functions.
- class [GraphInset](#)
Class used to draw a graph of time stamped data in the model window.
- class [KeyFramer](#)
Class describing the evolution of a bspline surface through time.
- class [Materials](#)
A utility class for materials.

- class [Matrix](#)
Class representing a fixed-size matrix.
- class [Palette](#)
A utility class for palettes.
- class [Parms](#)
A utility class to parse L-Studio like parameter files.
- class [Point](#)
A 3D point/vector class.
- class [range](#)
Class representing a range of values from two iterators.
- class [refpair](#)
Class used to hold references for the [util::tie\(\)](#) function.
- class [SelectMemberIterator](#)
Iterate over a container of structure, dereferencing only a member of it.
- struct [set_vector](#)
Define a class using (small) vectors as unordered sets.
- class [Tensor](#)
A growth tensor class.
- class [Texture1D](#)
A utility class for one-dimensional textures.
- class [Texture2D](#)
A utility class for two-dimensional textures.
- class [Vector](#)
[Vector](#) class supporting all classic classic vector operations.
- class [WatchDog](#)
Watch the modifications of file objects for you.

Typedefs

- typedef [Color](#)< double > **Colord**
- typedef [Color](#)< float > **Colorf**

Functions

- void **__assert_fail** (const **QString** &assertion, const char *file, unsigned int line, const char *function)
- void **__assert_fail_txt** (const **QString** &assertion, const char *file, unsigned int line, const char *function)
- std::string **absoluteDir** (std::string filename, std::string current_dir=std::string())

Returns the absolute path of filename.

- bool **approx** (double v1, double v2, double abs_epsilon=VVE_EPSILON, double rel_epsilon=VVE_REL_EPSILON)

Test if v1 == v2, with floating point error.

- template<class T >
T **clamp** (const T &val, const T &min, const T &max)

A function to clamp a value to a range.

- void **convertFromQColor** (**Color**< long long > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< long > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< int > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< short > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< char > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< unsigned long long > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< unsigned long > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< unsigned int > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< unsigned short > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< unsigned char > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< long double > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< double > &c, const **QColor** &col)
- void **convertFromQColor** (**Color**< float > &c, const **QColor** &col)
- template<class T >
Color< T > **convertHSVtoRGB** (T h, T s, T v)

Return a color based on HSV values.

- **QColor** **convertToQColor** (const **Color**< long long > &c)
- **QColor** **convertToQColor** (const **Color**< long > &c)
- **QColor** **convertToQColor** (const **Color**< int > &c)
- **QColor** **convertToQColor** (const **Color**< short > &c)
- **QColor** **convertToQColor** (const **Color**< char > &c)
- **QColor** **convertToQColor** (const **Color**< unsigned long long > &c)
- **QColor** **convertToQColor** (const **Color**< unsigned long > &c)
- **QColor** **convertToQColor** (const **Color**< unsigned int > &c)
- **QColor** **convertToQColor** (const **Color**< unsigned short > &c)
- **QColor** **convertToQColor** (const **Color**< unsigned char > &c)
- **QColor** **convertToQColor** (const **Color**< long double > &c)

- **QColor convertToQColor** (const [Color](#)< double > &c)
- **QColor convertToQColor** (const [Color](#)< float > &c)
- std::string [demangle](#) (std::string s)
Demangle all the words in the string.
- template<size_t dim>
[Vector](#)< dim, double > [gaussRan](#) (const [Vector](#)< dim, double > &mean, const [Vector](#)< dim, double > &sigma)
Generate a random vector with gaussian distribution.
- double [gaussRan](#) (double mean, double sigma)
Generate a random number with gaussian distribution.
- bool [greater](#) (double v1, double v2, double abs_epsilon=VVE_EPSILON, double rel_epsilon=VVE_REL_EPSILON)
Test if v1 > v2, with floating point error.
- bool [less](#) (double v1, double v2, double abs_epsilon=VVE_EPSILON, double rel_epsilon=VVE_REL_EPSILON)
Test if v1 < v2, with floating point error.
- template<typename Iterator >
[range](#)< Iterator > [make_range](#) (const std::pair< Iterator, Iterator > &p)
Function to create a range from a pair of iterators.
- template<typename Iterator >
[range](#)< Iterator > [make_range](#) (const Iterator &first, const Iterator &last)
Function to create a range from two iterators.
- template<size_t dim, typename T, typename T1 >
[Vector](#)< dim, T > [map](#) (T(*fct)(const T1 &), const [Vector](#)< dim, T1 > &v)
- template<size_t dim, typename T, typename T1 >
[Vector](#)< dim, T > [map](#) (const T &(*fct)(const T1 &), const [Vector](#)< dim, T1 > &v)
- template<size_t dim, typename T >
[Vector](#)< dim, T > [map](#) (T(*fct)(T), const [Vector](#)< dim, T > &v)
- template<size_t dim, typename T >
[Vector](#)< dim, T > [map](#) (T(*fct)(const T &), const [Vector](#)< dim, T > &v)
- template<size_t dim, typename T >
[Vector](#)< dim, T > [map](#) (const T &(*fct)(const T &), const [Vector](#)< dim, T > &v)
- template<size_t nRows, typename T >
T [matrix_minor](#) (const [Matrix](#)< nRows, nRows, T > &mat, size_t i, size_t j)
- template<typename T >
T [maximum](#) (const T &a, const T &b)
Return the maximum of two values.

- `template<typename T >`
`T minimum (const T &a, const T &b)`
Return the minimum of two values.
- `bool notequal (double v1, double v2, double abs_epsilon=VVE_EPSILON, double rel_epsilon=VVE_REL_EPSILON)`
Test if $v1 \neq v2$, with floating point error.
- `template<class T >`
`Point< T > operator* (const T &s, const Point< T > &p)`
Scalar multiplication operator.
- `template<class T >`
`Point< T > operator* (const Point< T > &p, const T &s)`
Scalar multiplication operator.
- `template<class T >`
`std::ostream & operator<< (std::ostream &os, Tensor< T > &tensor)`
- `template<class T, unsigned int size>`
`std::ostream & operator<< (std::ostream &os, Buffer< T, size > &b)`
Stream output to the array.
- `template<class T >`
`std::istream & operator>> (std::istream &is, Tensor< T > &tensor)`
- `QTextStream & operator>> (QTextStream &ss, bool b)`
- `template<class T, unsigned int size>`
`std::istream & operator>> (std::istream &is, Buffer< T, size > &b)`
Stream input to the array.
- `QString qdemangle (std::string s)`
Demangle all the words in the string.
- `template<size_t dim>`
`Vector< dim, double > ran (const Vector< dim, double > &V)`
Generate a random vector uniformly distributed between [Vector](#)(0) and V.
- `double ran (double M)`
Generate a random number uniformly distributed between 0 and M.
- `template<size_t dim>`
`Vector< dim, long int > random (const Vector< dim, long int > &n)`
Returns a vector of random numbers.
- `long int random (long int n)`
Returns a random number between 0 and n (excluded), for $n \leq RAND_MAX$.
- `long int random ()`

Returns a random number between 0 and RAND_MAX.

- `template<typename T >`
`bool range_c (const T &min, const T &max, const T &val)`
Check if a value is in a C-style range (closed on lower bound and open on the upper).
- `template<typename T >`
`bool range_closed (const T &min, const T &max, const T &val)`
Check if a value is in an closed range.
- `template<typename T >`
`bool range_open (const T &min, const T &max, const T &val)`
Check if a value is in an open range.
- `template<size_t dim, class T >`
`bool serialization (storage::VVEStorage &store, Vector< dim, T > &vec)`
- `template<size_t nRows, size_t nCols, typename T >`
`bool serialization (storage::VVEStorage &store, Matrix< nRows, nCols, T > &mat)`
- `void sran (unsigned int seed)`
Initialize the random number generator.
- `unsigned int sran_time ()`
Initialize the random number with the current time of the day (in microsecond).
- `template<typename T, typename U >`
`refpair< T, U > tie (T &x, U &y)`
Tie two variables to the values of a pair.

Variables

- `QTextStream err` (stderr)
- `QTextStream out` (stdout)

16.17.1 Detailed Description

Various utility classes for generic programming. Namespace containing all the utility classes.

16.17.2 Function Documentation

16.17.2.1 `std::string util::absoluteDir (std::string filename, std::string current_dir = std::string())`

Returns the absolute path of `filename`.

If `filename` is not already absolute, it is considered relative to `current_dir`.

16.17.2.2 `bool util::approx (double v1, double v2, double abs_epsilon = VVE_EPSILON, double rel_epsilon = VVE_REL_EPSILON) [inline]`

Test if `v1 == v2`, with floating point error.

Definition at line 45 of file `floats.h`.

Referenced by `notequal()`.

```

46  {
47      double diff = fabs(v1-v2);
48      if(diff < abs_epsilon) return true;
49      double tot = fabs(v1)+fabs(v2);
50      return (diff/tot) < rel_epsilon;
51  }
```

16.17.2.3 `template<class T> T util::clamp (const T & val, const T & min, const T & max) [inline]`

A function to clamp a value to a range.

Parameters

val The start value.

min The minimum value of the range.

max The maximum value of the range.

If `min` is more than `max`, the function returns `max`.

Definition at line 19 of file `clamp.h`.

```

19                                     {
20      if (min >= max) return max;
21      else if (val < min) return min;
22      else if (val > max) return max;
23      else return val;
24  }
```

16.17.2.4 `template<class T> Color<T> util::convertHSVtoRGB (T h, T s, T v) [inline]`

Return a color based on HSV values.

Definition at line 194 of file `color.h`.

References `util::Color< T >::a()`, `util::Color< T >::b()`, `util::Color< T >::g()`, and `util::Color< T >::r()`.

```

194                                     {
195     // based on Jo's code in medit
196
197     Color<T> rgb;
198     rgb.a(1.0);
199
200     while (h > 360.0) h -= 360.0;
201     while (h < 0.0) h += 360.0;
202
203     h /= 60.0;
204
205     int i = int(h);
206
207     double f = h - i;
208     double p = v * (1 - s);
209     double q = v * (1 - (s * f));
210     double t = v * (1 - (s * (1 - f)));
211
212     switch (i) {
213     case 0: rgb.r(v); rgb.g(t); rgb.b(p); break;
214     case 1: rgb.r(q); rgb.g(v); rgb.b(p); break;
215     case 2: rgb.r(p); rgb.g(v); rgb.b(t); break;
216     case 3: rgb.r(p); rgb.g(q); rgb.b(v); break;
217     case 4: rgb.r(t); rgb.g(p); rgb.b(v); break;
218     case 5: rgb.r(v); rgb.g(p); rgb.b(q); break;
219     }
220
221     return rgb;
222 }

```

16.17.2.5 std::string util::demangle (std::string s)

Demangle all the words in the string.

Returns

The string with all symbols demangled.

16.17.2.6 template<size_t dim> Vector<dim,double> util::gaussRan (const Vector< dim, double > & mean, const Vector< dim, double > & sigma) [inline]

Generate a random vector with gaussian distribution.

Parameters

mean [Vector](#) with mean of the gaussian distribution for each dimension

sigma [Vector](#) with standard deviation of the gaussian distribution for each dimension

Definition at line 61 of file random.h.

References [gaussRan\(\)](#).

```

62  {
63      Vector<dim,double> result;
64      for(size_t i = 0 ; i < dim ; ++i)
65          result[i] = gaussRan(mean[i], sigma[i]);
66      return result;
67  }

```

16.17.2.7 double util::gaussRan (double *mean*, double *sigma*)

Generate a random number with gaussian distribution.

Parameters

mean Mean of the gaussian distribution

sigma Standard deviation of the gaussian distribution

Referenced by gaussRan().

16.17.2.8 bool util::greater (double *v1*, double *v2*, double *abs_epsilon* = VVE_EPSILON, double *rel_epsilon* = VVE_REL_EPSILON) [inline]

Test if $v1 > v2$, with floating point error.

Definition at line 29 of file floats.h.

Referenced by less().

```

30  {
31      return (v1*(1+rel_epsilon/2) + abs_epsilon/2) > (v2*(1-rel_epsilon/2) - abs_epsilon/2);
32  }

```

16.17.2.9 bool util::less (double *v1*, double *v2*, double *abs_epsilon* = VVE_EPSILON, double *rel_epsilon* = VVE_REL_EPSILON) [inline]

Test if $v1 < v2$, with floating point error.

Definition at line 37 of file floats.h.

References greater().

```

38  {
39      return greater(v2, v1, abs_epsilon, rel_epsilon);
40  }

```


16.17.2.10 `template<typename Iterator > range<Iterator> util::make_range (const std::pair< Iterator, Iterator > & p) [inline]`

[Function](#) to create a range from a pair of iterators.

Definition at line 179 of file range.h.

```
180 {
181     return range<Iterator>(p);
182 }
```

16.17.2.11 `template<typename Iterator > range<Iterator> util::make_range (const Iterator & first, const Iterator & last) [inline]`

[Function](#) to create a range from two iterators.

Definition at line 170 of file range.h.

Referenced by `graph::VVGrahp< VertexContent, EdgeContent, compact >::iNeighbors()`, `graph::VVBIGrahp< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iNeighbors()`, `graph::VVGrahp< VertexContent, EdgeContent, compact >::neighbors()`, `graph::VVBIGrahp< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors()`, `graph::VVBIGrahp< typename cell::content_t, typename junction::content_t >::vertices1()`, and `graph::VVBIGrahp< typename cell::content_t, typename junction::content_t >::vertices2()`.

```
171 {
172     return range<Iterator>(first, last);
173 }
```

16.17.2.12 `template<typename T > T util::maximum (const T & a, const T & b) [inline]`

Return the maximum of two values.

Parameters

a The first value.

b The second value.

Definition at line 13 of file minmax.h.

```
13 {
14     return (a > b) ? a : b;
15 }
```

16.17.2.13 `template<typename T > T util::minimum (const T & a, const T & b) [inline]`

Return the minimum of two values.

Parameters

- a* The first value.
- b* The second value.

Definition at line 22 of file minmax.h.

```

22                                     {
23     return (a < b) ? a : b;
24 }
```

16.17.2.14 `bool util::notequal (double v1, double v2, double abs_epsilon = VVE_EPSILON, double rel_epsilon = VVE_REL_EPSILON) [inline]`

Test if $v1 \neq v2$, with floating point error.

Definition at line 56 of file floats.h.

References `approx()`.

```

57 {
58     return !approx(v1, v2, abs_epsilon, rel_epsilon);
59 }
```

16.17.2.15 `template<class T > Point<T> util::operator* (const T & s, const Point< T > & p) [inline]`

Scalar multiplication operator.

Parameters

- p* The point to scale.
- s* The scaling value.

Definition at line 73 of file point.h.

```

73                                     {
74     return p * s;
75 }
```

16.17.2.16 `template<class T > Point<T> util::operator* (const Point< T > & p, const T & s) [inline]`

Scalar multiplication operator.

Parameters

- p* The point to scale.
- s* The scaling value.

Definition at line 62 of file point.h.

References `util::Vector< 3, T >::Vector()`.

```
62                                     {
63     Point<T> r( p.Vector<3,T>::operator*( s ) );
64     return r;
65 }
```

16.17.2.17 `template<class T , unsigned int size> std::ostream& util::operator<< (std::ostream & os, Buffer< T, size > & b) [inline]`

Stream output to the array.

Parameters

- os* The output stream.
- b* The [Buffer](#) object to read from.

The following stream output writes 'size' elements, assuming that elements are separated by a single space. For this to work, T must have an implemented output stream operator.

Definition at line 58 of file buffer.h.

```
58                                     {
59     for (unsigned int i = 0; i < size; ++i) {
60         if (i) os << " ";
61         os << b[i];
62     }
63     return os;
64 }
```

16.17.2.18 `template<class T , unsigned int size> std::istream& util::operator>> (std::istream & is, Buffer< T, size > & b) [inline]`

Stream input to the array.

Parameters

- is* The input stream.
- b* The [Buffer](#) object to write into.

The following stream input reads 'size' elements, assuming that elements are separated by whitespace as defined by `std::ws`. For this to work, T must have an implemented input stream operator.

Definition at line 42 of file `buffer.h`.

```

42                                     {
43     for (unsigned int i = 0; i < size; ++i)
44         is >> std::ws >> b[i];
45     return is;
46 }
```

16.17.2.19 QString util::qdemangle (std::string s)

Demangle all the words in the string.

Returns

The string with all symbols demangled.

16.17.2.20 template<size_t dim> Vector<dim,double> util::ran (const Vector< dim, double > & V) [inline]

Generate a random vector uniformly distributed between Vector(0) and V.

Definition at line 36 of file `random.h`.

References `ran()`.

```

37 {
38     Vector<dim,double> result;
39     for(size_t i = 0 ; i < dim ; ++i)
40         result[i] = ran(V[i]);
41     return result;
42 }
```

16.17.2.21 double util::ran (double M)

Generate a random number uniformly distributed between 0 and M.

Referenced by `ran()`.

16.17.2.22 `template<size_t dim> Vector<dim,long int> util::random (const Vector< dim, long int > &n) [inline]`

Returns a vector of random numbers.

Parameters

n [Vector](#) with the maximum number for each dimension

Definition at line 96 of file random.h.

References [random\(\)](#).

```

97  {
98      Vector<dim,double> result;
99      for(size_t i = 0 ; i < dim ; ++i)
100          result[i] = random(n[i]);
101      return result;
102  }
```

16.17.2.23 `long int util::random (long int n)`

Returns a random number between 0 and n (excluded), for n <= RAND_MAX.

16.17.2.24 `Vector< dim, long int > util::random () [inline]`

Returns a random number between 0 and RAND_MAX.

Returns a vector of random numbers between 0 and RAND_MAX.

Definition at line 78 of file random.h.

Referenced by [random\(\)](#).

```

79  {
80      Vector<dim,double> result;
81      for(size_t i = 0 ; i < dim ; ++i)
82          result[i] = random();
83      return result;
84  }
```

16.17.2.25 `template<typename T > bool util::range_c (const T &min, const T &max, const T &val) [inline]`

Check if a value is in a C-style range (closed on lower bound and open on the upper).

Parameters

min The minimum range value.

max The maximum range value.

val The value to check.

Definition at line 211 of file range.h.

```
211
212     return (val >= min && val < max);
213 }
```

{

16.17.2.26 `template<typename T> bool util::range_closed (const T & min, const T & max, const T & val) [inline]`

Check if a value is in an closed range.

Parameters

min The minimum range value.

max The maximum range value.

val The value to check.

Definition at line 190 of file range.h.

```
190
191     {
192     return (val >= min && val <= max);
192 }
```

16.17.2.27 `template<typename T> bool util::range_open (const T & min, const T & max, const T & val) [inline]`

Check if a value is in an open range.

Parameters

min The minimum range value.

max The maximum range value.

val The value to check.

Definition at line 200 of file range.h.

```
200
201     {
202     return (val > min && val < max);
202 }
```

16.17.2.28 `void util::sran (unsigned int seed)`

Initialize the random number generator.

16.17.2.29 unsigned int util::srand_time ()

Initialize the random number with the current time of the day (in microsecond).

Returns

the seed used.

16.17.2.30 template<typename T , typename U > refpair<T, U> util::tie (T & x, U & y) [inline]

Tie two variables to the values of a pair.

Example:

```
std::pair<int,double> p(1,2.5);
int a;
double b;
util::tie(a,b) = p;
```

At the end, a is 1 and b is 2.5

Definition at line 73 of file tie.h.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact >::insert(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insert(), graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdge(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertEdge(), graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdges(), graph::VVGraph< VertexContent, EdgeContent, compact >::replace(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::replace(), graph::VVGraph< VertexContent, EdgeContent, compact >::spliceAfter(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter(), graph::VVGraph< VertexContent, EdgeContent, compact >::spliceBefore(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceBefore().

```
73 { return refpair<T, U>(x, y); }
```

16.18 util::Shapes Namespace Reference

This namespace contains various shapes.

Classes

- class [BSplineSurface](#)
Class describing a bspline surface.

16.18.1 Detailed Description

This namespace contains various shapes. For now, only [BSplineSurface](#) is included.

16.19 vvcomplex Namespace Reference

Definition of classes and functions for 2d cell complexes.

Classes

- class [DivisionData](#)
Class holding the data needed to actually divide a cell.
- class [InModelDivisionParam](#)
Class used to define the division parameters from within the model class.
- class [VVComplex](#)
Class handling the representation and development of 2D cell complex.
- class [VVComplexGraph](#)
Definition of the bipartite graph with specialized methods for the complex graph.

Typedefs

Types defined by `EXPORT_COMPLEX_TYPES(VVComplex)`

- typedef [VVComplex::cell](#) `cell`
Type of a cell.
- typedef [VVComplex::cell_edge](#) `cell_edge`
Type of an edge in the cell graph.
- typedef [VVComplex::cell_graph](#) `cell_graph`
Type of the cell graph.
- typedef [VVComplex::cell_junction_edge](#) `cell_junction_edge`
Type of an edge from a cell to a junction.
- typedef [VVComplex::complex_graph](#) `complex_graph`
Type of the complex graph (i.e. linking cells to junctions and vice-versa).
- typedef [VVComplex::const_cell_edge](#) `const_cell_edge`
Type of an edge in the cell graph.
- typedef [VVComplex::const_cell_junction_edge](#) `const_cell_junction_edge`
Type of an edge from a cell to a junction.
- typedef [VVComplex::const_junction_cell_edge](#) `const_junction_cell_edge`
Type of an edge from a junction to a cell.
- typedef [VVComplex::const_wall](#) `const_wall`

Type of a wall (i.e. an edge in the wall graph).

- typedef [VVComplex::junction](#) **junction**
Type of a junction.
- typedef [VVComplex::junction_cell_edge](#) **junction_cell_edge**
Type of an edge from a junction to a cell.
- typedef [VVComplex::wall](#) **wall**
Type of a wall (i.e. an edge in the wall graph).
- typedef [VVComplex::wall_graph](#) **wall_graph**
Type of the wall graph.

Functions

- template<typename Complex >
std::vector< typename Complex::junction > **contour** (const Complex &T)
- template<typename CplxGraph >
void **create_cplxgraph_methods** (CplxGraph &G)
- template<class VVComplex >
void **FindCenter** (const typename [VVComplex::cell](#) &c, [VVComplex](#) &T)
Compute the position of the center of a cell.
- template<class VVComplex >
Point3d **findCenter** (const typename [VVComplex::cell](#) &c, [VVComplex](#) &T)
Return the position of the center of a cell.
- template<class VVComplex >
[DivisionData](#)< typename [VVComplex::junction_content](#) > **findDivisionPoints**
(const typename [VVComplex::cell](#) &c, [VVComplex](#) &T, const [InModelDivisionParam](#) ¶m)
Implementation of the in model division scheme.
- template<typename VVComplex >
void **FindOppositeWall** (const typename [VVComplex::cell](#) &c, [DivisionData](#)<
typename [VVComplex::junction_content](#) > &result, [VVComplex](#) &T, double
cellWallMin, bool strictCellWallMin)
Find the wall opposite to the point already selected.
- bool **FindWallMin** (Point3d &v, const Point3d &v1, const Point3d &v2, double
mw, double *displacement=0)
Move point v to avoid segment borders.
- template<typename V1 , typename V2 , typename E1 , typename E2 , bool compact>
bool **serialization** ([storage::VVEStorage](#) &store, [VVComplexGraph](#)< V1, V2,
E1, E2, compact > &G)

- `template<typename VVComplex >`
`void testDivisionOnVertices (const typename VVComplex::cell &c, Division-`
`Data< typename VVComplex::junction_content > &result, VVComplex &T,`
`double epsilon)`

Test if the division point in result is close enough to one of the extremum.

16.19.1 Detailed Description

Definition of classes and functions for 2d cell complexes.

16.19.2 General Information

This namespace mainly provides a `VVComplex` class template handling the creation and evolution of cell complexes.

A 2D cell complex is defined by a set of regions of the space such that two regions neighbor regions share a continuous set of 1D cells (i.e. line segments). We represent here the regions as a bipartite graph where the two group of vertices are the cells and the junctions. The junctions neighbors of a cell are ordered such that two junctions next to each other are linked by a wall. The cells neighbors of a junction are ordered such that two cells sharing a wall will be next to each other. In addition, two useful graphs are maintained: the cell graph, linking cells sharing a wall; and the wall graph, linking junctions parts of the same wall.

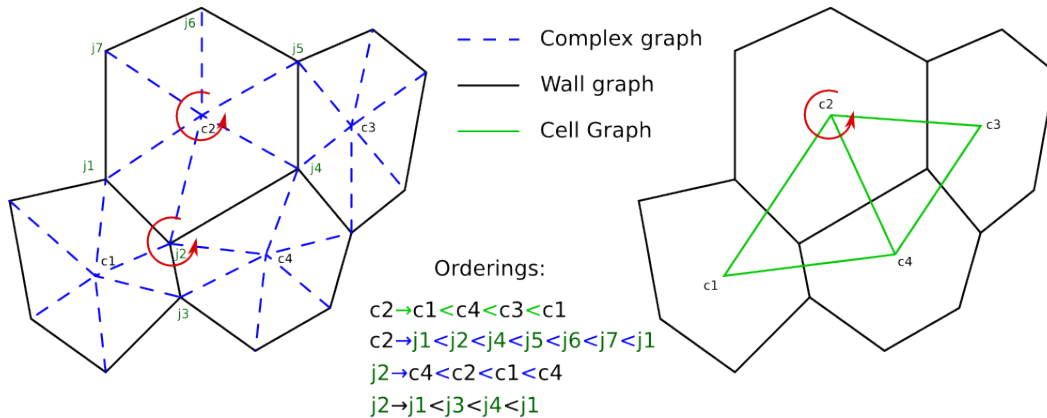


Figure 16.1: Graphs and ordering in 2D cell complexes.

By default, all graphs are ordered counter-clockwise, but any consistent ordering is possible.

Note

The complex graph is the only one defining the cell complex. The cell graph can be defined as the graph linking two cells sharing two vertices next to each other

in their neighborhood. The wall graph can be defined as the graph linking two vertices if there exist a cell in which the two vertices are next to each other.

16.19.3 2D Cell Complex Usage

16.19.3.1 Definition of a 2D Cell Complex

A 2D cell complex needs 6 types and the type of the model. The 6 types are:

1. The content of a cell
2. The content of a junction
3. The content of a wall (i.e. an edge in the wall graph)
4. The content of an edge between two cells
5. The content of an edge from a cell to a junction
6. The content of an edge from a junction to a cell

For the rest of this page, these types will be called:

1. CellContent
2. JunctionContent
3. WallContent
4. CellEdgeContent
5. CellJunctionEdgeContent
6. JunctionCellEdgeContent

Then, if the type of the model is `MyModel`, the cell complex is defined as:

```
typedef
    vvcomplex::VVComplex<MyModel, CellContent, JunctionContent, WallContent, CellEdgeContent, Ce
    VVComplex;
```

All edge content types are optional and defaults to an empty class. Once created, a macro is provided to help exporting all the types:

```
EXPORT_COMPLEX_TYPES (VVComplex) ;
```

is equivalent to:

```
typedef VVComplex::cell cell;
typedef VVComplex::junction junction;

typedef VVComplex::wall wall;
typedef VVComplex::cell_edge cell_edge;
typedef VVComplex::cell_junction_edge cell_junction_edge;
```

```

typedef VVComplex::junction_cell_edge junction_cell_edge;

typedef VVComplex::const_wall const_wall;
typedef VVComplex::const_cell_edge const_cell_edge;
typedef VVComplex::const_cell_junction_edge const_cell_junction_edge;
typedef VVComplex::const_junction_cell_edge const_junction_cell_edge;

typedef VVComplex::complex_graph complex_graph;
typedef VVComplex::cell_graph cell_graph;
typedef VVComplex::wall_graph wall_graph;

```

Other macro, for partial export or for import in template function or class, are defined in `complex.h`

16.19.3.2 Creation and Edition of a 2D Cell Complex

2D cell complexes are created by instantiating the type defined in the previous section, specifying at construction time the pointer on the current model. In a function, the declaration will look like:

```
VVComplex T(&model);
```

As in the model class, the declaration will look like:

```

VVComplex T;
MyModel(QObject* parent)
    : Model(parent)
    , T(this)

```

Once created, the first cell can be created by creating first junctions and a cell, then using the method [VVComplex::addCell](#). For example to create a square cell (supposing the junctions have a `pos` attributes holding a 3D vector):

```

junction j1, j2, j3, j4;
cell c;
j1->pos = Point3d(0,0,0);
j2->pos = Point3d(1,0,0);
j3->pos = Point3d(1,1,0);
j4->pos = Point3d(0,1,0);
T.addCell(c, j1, j2, j3, j4);

```

Another way is to use one of the factory provided in the [complex_factory](#) namespace.

You can then continue to edit the cell complex using the [VVComplex::addCell](#), [VVComplex::divideCell](#) or [VVComplex::deleteCell](#) methods. For finer edition, you can use [VVComplex::splitWall](#) or [VVComplex::deleteJunction](#).

For example, given a division algorithm using parameters `DivisionParams`, you can divide a cell with:

```
T.divideCell(c, DivisionParams(), c);
```

During the division, the `updateFromOld` method of the model object will be called. During that called, the members [VVComplex::OldS](#), [VVComplex::OldC](#) and [VVComplex::OldW](#) will be available with the part of the graph saved (in the example, only the cell `c` and its junctions are saved). For example, if you want to increase a generation counter, both for the junctions and the cells:

```

void updateFromOld(const cell& cl, const cell& cr, const cell& c,
                  const VVComplex::division_data& ddata, VVComplex& T)
{
    cl->generation = cr->generation = c->generation+1;
    forall( const junction& j , T.S.neighbors(cl))
    {
        if(!T.OldW.contains(j))
            j->generation = cl->generation;
    }
}

```

Here, the generation of a junction is defined as the generation of the cells for which it was created.

At last, it can be useful to know the two ways to iterate over cells and junctions. These notations are pretty much equivalent:

```

forall(const cell& c,T.C) { ... }
forall_named(const cell& c, T.S, cells) { ... }

forall(const junction& j,T.W) { ... }
forall_named(const junction& j, T.S, junctions) { ... }

```

The difference is in performance. It is always better to iterate over the graph used in the loop itself.

16.19.3.3 Requirements on the Model using 2D Cell Complexes

The following methods must exist in the model class to be able to use the 2d cell complexes.

```

Point3d position(const cell& c);
Point3d position(const junction& j);

```

Returns the position of a cell or a junction.

```

void setPosition(const cell& c, const Point3d& pos);
void setPosition(const junction& j, const Point3d& pos);

```

Set the position of a cell or a junction.

```

void setPositionHint(const junction& j, const junction& j1, const junction& j2,
                    double ratio);

```

Used to preset the position of a newly formed junction during cell division.

```

Point3d normal(const cell& c);

```

Returns the normal to a cell.

```

void updateFromOld(const cell& cl, const cell& cr, const cell& c, const
                  VVComplex::division_data& ddata, VVComplex& T);

```

Method called after the division occurred. `cl` and `cr` are the two newly created cells. `c` is the previous cell. `ddata` is the data used for the division. The complex has the old graphs setup with whatever was required to be saved during division.

16.19.3.4 Implementing a Cell Division Algorithm

Implementing a cell division algorithm consists in implementing the function:

```
template <class VVComplex>
vvcomplex::DivisionData<typename VVComplex::junction_content>
    findDivisionPoints(const typename VVComplex::cell& c,
                      VVComplex& T,
                      const MyDivisionParam& param);
```

with `MyDivisionParams` being a type containing the parameters needed for the division and, most importantly, unique for the division as it will identify which division algorithm to use.

This function must not change the graph! Within the function, it might be convenient to use the macros `IMPORT_COMPLEX_TYPES()` and `IMPORT_COMPLEX_MODEL()` to access easily the structure of the cell complex. A good example is the very simple the division algorithm using `vvcomplex::InModelDivisionParam` as parameters: `vvcomplex::findDivisionPoints()`.

16.19.4 Extending the Cell Complex Class

Extending the cell complex can simply consist in adding methods (like the `tissue::Tissue` class does) or it can also consist in maintaining extra structures, like extra graphs. To allow extra graphs to be maintained without user input, virtual functions can be redefined:

- `VVComplex::addCellExtra(const cell&)`
- `VVComplex::reconnectGraphs()`
- `VVComplex::saveSubgraphs()`
- `VVComplex::clearOldGraphs()`
- `VVComplex::deleteCellInGraphs()`
- `VVComplex::splitWallExtra()`
- `VVComplex::deleteJunctionExtra()`

Each of these method **must** call its parent's method to ensure all graphs are properly maintained.

Also, it is very important for the complex to know the instantiated class. For that purpose, the last argument of the `VVComplex` class is the leaf class instantiated. If you want to provide a class someone else may inherit from, it is recommended to look at the `Tissue` class declaration and achieve something similar.

16.19.5 Example

Here is a commented example of a model using the cell complex.

First, we start by defining the data structure for the cell and the vertices. As we don't label the edges, we ignore them for now.

```
#include <vve.h>

#include <geometry/geometry.h>
#include <algorithms/cellsystem.h>
#include <geometry/area.h>

using geometry::Point3d;
struct CellContent
{
    Point3d position;
    int id;
};

struct JunctionContent
{
    Point3d position;
};
```

Now, we declare the complex data structure. As it needs to know about the model class used, we need to place a forward declaration of the model class. At last, we export the types defined in the complex class (i.e. the cell, junction, cell_edge, ... types).

```
class MyModel;
typedef vvcplex::VVCplex<MyModel, CellContent, JunctionContent> MyComplex;
EXPORT_COMPLEX_TYPES(MyComplex);
```

Now, the model should contain an instance of MyComplex and the methods required by the complex class.

```
struct MyModel : public Model
{
    MyComplex T;
    Point3d position(const cell& c) { return c->position; }
    Point3d position(const junction& j) { return j->position; }
    Point3d setPosition(const cell& c, const Point3d& pos) { c->position = pos; }
    Point3d setPosition(const junction& j, const Point3d& pos) { j->position = pos; }
    ; }
    void setPositionHint(const junction&, const junction&, const junction&, double ) {}
    Point3d normal(const cell& ) { return Point3d(0,0,1); }
    Point3d normal(const junction& ) { return Point3d(0,0,1); }

    void updateFromOld(const cell& cl, const cell& cr, const cell& c, const MyComplex::division_data&, MyComplex& )
    {
        cr->id = 2*c->id;
        cl->id = 2*c->id+1;
    }
};
```

Now, to use the structure, we need to initialise it in the constructor, and to add its first cell.

```
MyModel(QObject *parent)
```



```

: Model(parent)
, T(this)
{
    // Add a square cell
    junction j1, j2, j3, j4;
    j1->position = Point3d(0,0,0);
    j2->position = Point3d(1,0,0);
    j3->position = Point3d(1,1,0);
    j4->position = Point3d(0,1,0);

    cell c;
    c->id = 1;
    c->position = Point3d(.5, .5, 0);
    T.addCell(c, j1, j2, j3, j4);
}

```

For the step, we'll use the cell system division mechanism alternating the normal from horizontal to vertical.

```

void step()
{
    static bool horizontal = false;
    horizontal = !horizontal;
    cell_system::CellSystemDivisionParams params(Point3d(1,0,0));
    if(horizontal)
        params.angle = M_PI/2;
    std::vector<cell> cells(T.C.begin(), T.C.end()); // Copy the cells
    forall(const cell& c, cells)
    {
        T.divideCell(c, params);
    }
}

// This method is needed for the cell system division
double area(const cell& c)
{
    std::vector<Point3d> poss;
    forall( const junction& j , T.S.neighbors(c))
    {
        poss.push_back(j->position);
    }
    return geometry::polygonArea(poss);
}

```

At last, we draw the result simply with lines for the walls.

```

void draw(Viewer* viewer)
{
    Point3d pmin(HUGE_VAL, HUGE_VAL, HUGE_VAL);
    Point3d pmax = -pmin;
    glBegin(GL_LINES);
    glColor3f(1,1,1);
    forall(const junction& j, T.W)
    {
        const Point3d& p = j->position;
        if(p.x() < pmin.x())
            pmin.x() = p.x();
        if(p.y() < pmin.y())
            pmin.y() = p.y();
        if(p.z() < pmin.z())

```

```

        pmin.z() = p.z();
        if(p.x() > pmax.x())
            pmax.x() = p.x();
        if(p.y() > pmax.y())
            pmax.y() = p.y();
        if(p.z() > pmax.z())
            pmax.z() = p.z();
        forall( const junction& n , T.W.neighbors(j))
        {
            glVertex3dv(p.c_data());
            glVertex3dv(n->position.c_data());
        }
    }
    glEnd();
    viewer->setSceneBoundingBox(Vec(pmin), Vec(pmax));
}

};

DEFINE_MODEL(MyModel);

```

16.19.6 Typedef Documentation

16.19.6.1 typedef VVComplex::cell vvcomplex::cell

Type of a cell.

Definition at line 693 of file complex.h.

16.19.6.2 typedef VVComplex::cell_edge vvcomplex::cell_edge

Type of an edge in the cell graph.

Definition at line 699 of file complex.h.

16.19.6.3 typedef VVComplex::cell_graph vvcomplex::cell_graph

Type of the cell graph.

Definition at line 726 of file complex.h.

16.19.6.4 typedef VVComplex::cell_junction_edge vvcomplex::cell_junction_edge

Type of an edge from a cell to a junction.

Definition at line 711 of file complex.h.

16.19.6.5 typedef VVComplex::complex_graph vvcomplex::complex_graph

Type of the complex graph (i.e. linking cells to junctions and vice-versa).

Definition at line 723 of file complex.h.

16.19.6.6 typedef VVComplex::const_cell_edge vvcomplex::const_cell_edge

Type of an edge in the cell graph.

Definition at line 702 of file complex.h.

**16.19.6.7 typedef VVComplex::const_cell_junction_edge
vvcomplex::const_cell_junction_edge**

Type of an edge from a cell to a junction.

Definition at line 714 of file complex.h.

**16.19.6.8 typedef VVComplex::const_junction_cell_edge
vvcomplex::const_junction_cell_edge**

Type of an edge from a junction to a cell.

Definition at line 720 of file complex.h.

16.19.6.9 typedef VVComplex::const_wall vvcomplex::const_wall

Type of a wall (i.e. an edge in the wall graph).

Definition at line 708 of file complex.h.

16.19.6.10 typedef VVComplex::junction vvcomplex::junction

Type of a junction.

Definition at line 696 of file complex.h.

**16.19.6.11 typedef VVComplex::junction_cell_edge
vvcomplex::junction_cell_edge**

Type of an edge from a junction to a cell.

Definition at line 717 of file complex.h.

16.19.6.12 typedef VVComplex::wall vvcomplex::wall

Type of a wall (i.e. an edge in the wall graph).

Definition at line 705 of file complex.h.

16.19.6.13 typedef VVComplex::wall_graph vvcomplex::wall_graph

Type of the wall graph.

Definition at line 729 of file complex.h.

16.19.7 Function Documentation

16.19.7.1 `template<class VVComplex > void vvcomplex::FindCenter (const
typename VVComplex::cell & c, VVComplex & T) [inline]`

Compute the position of the center of a cell.

This function compute the barycenter of a cell using the position of its junctions.

Note

The position of the center is only updated, thus there is no call to [setPositionHint](#)

Deprecated

You should use [vvcomplex::findCenter](#) instead and update yourself the position of the cell.

Definition at line 1914 of file complex.h.

References [findCenter\(\)](#), and `IMPORT_COMPLEX_MODEL`.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell()`.

```
1915     {
1916         IMPORT_COMPLEX_MODEL(VVComplex, T);
1917         const Point3d& cpos = findCenter(c, T);
1918         model.setPosition(c, cpos);
1919     }
```

16.19.7.2 `template<class VVComplex > Point3d vvcomplex::findCenter (const
typename VVComplex::cell & c, VVComplex & T) [inline]`

Return the position of the center of a cell.

This function compute the barycenter of a cell using the position of its junctions.

Definition at line 1872 of file complex.h.

References `forall`, `IMPORT_COMPLEX_MODEL`, `IMPORT_COMPLEX_VERTICES`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::prevTo()`, `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::S`, `geometry::triangleArea()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::valence()`.

Referenced by `FindCenter()`.

```

1873     {
1874         IMPORT_COMPLEX_VERTICES(VVComplex);
1875         IMPORT_COMPLEX_MODEL(VVComplex, T);
1876         // First get average of points as estimate to center
1877         Point3d cest;
1878         forall( const junction& n , T.S.neighbors(c))
1879         {
1880             cest += model.position(n);
1881         }
1882         cest /= T.S.valence(c);
1883         // Average Area * midpoint of each triangle
1884         Point3d cpos;
1885         double sum = 0;
1886         forall( const junction& m , T.S.neighbors(c))
1887         {
1888             const junction& n = T.S.prevTo(c, m);
1889             const Point3d& mpos = model.position(m);
1890             const Point3d& npos = model.position(n);
1891             double ta = geometry::triangleArea(mpos, npos, cest);
1892             cpos += ta * (cest + ((mpos - cest) + (npos - cest))/3.0);
1893             sum += ta;
1894         }
1895         cpos /= sum;
1896
1897         return cpos;
1898     }

```

16.19.7.3 `template<class VVComplex > DivisionData<typename VVComplex::junction_content> vvcomplex::findDivisionPoints(const typename VVComplex::cell & c, VVComplex & T, const InModelDivisionParam & param) [inline]`

Implementation of the in model division scheme.

Implementation:

```

template <class VVComplex>
DivisionData<typename VVComplex::junction_content> findDivisionPoints(const ty
    pename VVComplex::cell& c,
    VVComplex& T,
    const InModelDivisionParam& param)
{
    IMPORT_COMPLEX_MODEL(VVComplex, T);
    return model.divisionParameters(c, T, param);
}

```

Definition at line 1745 of file complex.h.

References `IMPORT_COMPLEX_MODEL`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell()`.

```

1748     {
1749         IMPORT_COMPLEX_MODEL(VVComplex, T);
1750         return model.divisionParameters(c, T, param);
1751     }

```

16.19.7.4 `template<typename VVComplex > void
vvcomplex::FindOppositeWall (const typename VVComplex::cell &
c, DivisionData< typename VVComplex::junction_content > &
result, VVComplex & T, double cellWallMin, bool strictCellWallMin)
[inline]`

Find the wall opposite to the point already selected.

This function suppose the first division point was found. The other wall is the one intersecting the line starting from the point `result.pu` and going through the cell center. Then, the point is displaced to endure the minimum size of a wall.

Parameters

- ← *c* Cell to divide
- ↔ *result* Structure to update with the opposite wall
- ← *S* VV graph
- ← *cellWallMin* minimum size of a wall
- ← *model* [Model](#) using the algorithm

Definition at line 1786 of file `complex.h`.

References `FindWallMin()`, `forall`, `IMPORT_COMPLEX_MODEL`, `IMPORT_COMPLEX_VERTICES`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo()`, `geometry::planeLineIntersection()`, `vvcomplex::DivisionData< JunctionContent >::pu`, `vvcomplex::DivisionData< JunctionContent >::pv`, `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::S`, `vvcomplex::DivisionData< JunctionContent >::u1`, and `vvcomplex::DivisionData< JunctionContent >::v1`.

Referenced by `tissue::findDivisionPoints()`.

```

1790     {
1791         IMPORT_COMPLEX_VERTICES(VVComplex);
1792         IMPORT_COMPLEX_MODEL(VVComplex, T);
1793         double mindis = HUGE_VAL;
1794         const Point3d& cnml = model.normal(c);
1795         Point3d minu = result.pu;
1796         const Point3d& cpos = model.position(c);
1797         junction p_u1 = result.u1;
1798         junction p_u2 = T.S.nextTo(c, p_u1);
1799         forall( const junction& tv1 , T.S.neighbors(c))
1800         {
1801             const junction& tv2 = T.S.nextTo(c, tv1);
1802             const Point3d& v1 = model.position(tv1);
1803             const Point3d& v2 = model.position(tv2);
1804             if(tv2 == p_u2)
1805             {
1806                 continue;
1807             }
1808             // Keep uc close to same plane as v1v2
1809             Point3d uc = cpos - minu;

```

```

1810         uc -= (uc * cnml) * cnml;
1811         Point3d n = cnml^uc;
1812         uc += minu;
1813         double s;
1814         Point3d v;
1815         bool found = geometry::planeLineIntersection(v, s, uc, n, v1, v2);
1816         if(found)
1817         {
1818             double dis;
1819             if(FindWallMin(v, v1, v2, cellWallMin, &dis) or not strictCellWallMin)
1820             {
1821                 if(dis < mindis)
1822                 {
1823                     mindis = dis;
1824                     result.pv = v;
1825                     result.v1 = tv1;
1826                 }
1827                 if(dis == 0) break;
1828             }
1829         }
1830     }
1831 }

```

16.19.7.5 bool vvcomplex::FindWallMin (Point3d & v, const Point3d & v1, const Point3d & v2, double mw, double * displacement = 0)

Move point v to avoid segment borders.

Find out how to move the point v on the segment [v1, v2] to be at least at a distance mw from the v1 and v2.

Parameters

- ↔ **v** Point to displace if needed
- ← **v1** Extremity of the segment
- ← **v2** Extremity of the segment
- ← **mw** Minimum distance from the extremities
- **displacement** If non-null, it is set to be the displacement of the point.

Returns

True if it was possible, false otherwise. In case it wasn't possible, it displaces the point exactly between v1 and v2.

Referenced by FindOppositeWall().

16.19.7.6 template<typename VVComplex > void vvcomplex::testDivisionOnVertices (const typename VVComplex::cell & c, DivisionData< typename VVComplex::junction_content > & result, VVComplex & T, double epsilon) [inline]

Test if the division point in result is close enough to one of the extremum.

If so, setup the division to go through the vertex.

Definition at line 1838 of file complex.h.

References `vvcomplex::DivisionData< JunctionContent >::divide_at_u1`, `vvcomplex::DivisionData< JunctionContent >::divide_at_v1`, `IMPORT_COMPLEX_MODEL`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo()`, `vvcomplex::DivisionData< JunctionContent >::pu`, `vvcomplex::DivisionData< JunctionContent >::pv`, `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::S`, `vvcomplex::DivisionData< JunctionContent >::u1`, and `vvcomplex::DivisionData< JunctionContent >::v1`.

Referenced by `tissue::findDivisionPoints()`, and `cell_system::findDivisionPoints()`.

```

1842     {
1843         IMPORT_COMPLEX_MODEL(VVComplex, T);
1844         double eps_sq = epsilon*epsilon;
1845         if(util::normsq(result.pu-model.position(result.u1)) < eps_sq)
1846         {
1847             result.divide_at_u1 = true;
1848         }
1849         else if(util::normsq(result.pu-model.position(T.S.nextTo(c,result.u1))) < e
ps_sq)
1850         {
1851             result.u1 = T.S.nextTo(c,result.u1);
1852             result.divide_at_u1 = true;
1853         }
1854         if(util::normsq(result.pv-model.position(result.v1)) < eps_sq)
1855         {
1856             result.divide_at_v1 = true;
1857         }
1858         else if(util::normsq(result.pv-model.position(T.S.nextTo(c,result.v1))) < e
ps_sq)
1859         {
1860             result.v1 = T.S.nextTo(c,result.v1);
1861             result.divide_at_v1 = true;
1862         }
1863     }

```


Chapter 17

Class Documentation

17.1 `graph::_EmptyEdgeContent` Class Reference

Empty class used as default for edge content.

```
#include <graph/vvgraph.h>
```

Public Member Functions

- `_EmptyEdgeContent` & `operator=` (const `_EmptyEdgeContent` &)
- bool `serialize` (`storage::VVEStorage` &)

17.1.1 Detailed Description

Empty class used as default for edge content.

Definition at line 296 of file `vvgraph.h`.

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

- `vvelib/graph/vvgraph.h`

17.2 graph::Arc< EdgeContent > Struct Template Reference

Type of a undirected edge (or arc).

```
#include <edge.h>
```

Public Types

- typedef EdgeContent **content_t**
- typedef [Edge](#)< EdgeContent > **edge_t**
- typedef [edge_identity_t](#) **identity_t**

Type of the identity of a vertex.

Public Member Functions

- [Arc](#) ([identity_t](#) src, [identity_t](#) tgt, EdgeContent *c1, EdgeContent *c2)
Full constructor -> should never be called by the user.
- [Arc](#) (const [Arc](#) ©)
Copy constructor.
- [Arc](#) ()
Construct an empty (null) arc.
- [Arc](#) inv () const
Returns the opposite arc.
- bool isNull () const
Test if the content is null.
- operator bool () const
AN arc evaluates to true if it contains some data.
- operator [edge_t](#) () const
Convert the current arc into the corresponding edge.
- EdgeContent & operator* () const
Reference the content of the arc.
- [Arc](#) operator- () const
Unary '-' operator synchronize edges and returns opposite arc.
- EdgeContent * operator-> () const

Reference the content of the arc as a pointer.

- `identity_t source () const`

Returns the identifier of the source of the edge.

- `void sync () const`

Synchronize both sides of the arc.

- `identity_t target () const`

Returns the identifier of the target of the edge.

- `~Arc ()`

Destroy and copy the content of the arcs.

Protected Attributes

- `identity_t _source`
- `identity_t _target`
- `EdgeContent * main`
- `EdgeContent * other`

17.2.1 Detailed Description

template<typename EdgeContent> struct graph::Arc< EdgeContent >

Type of a undirected edge (or arc). When an arc is destroyed, the content of the main edge is copied in the other one.

Note that you should never keep an arc! Is is meant for temporary usage. This is why there is no copy constructor or operator=.

Definition at line 271 of file edge.h.

17.2.2 Member Typedef Documentation

17.2.2.1 template<typename EdgeContent > typedef edge_identity_t graph::Arc< EdgeContent >::identity_t

Type of the identity of a vertex.

Definition at line 276 of file edge.h.

17.2.3 Constructor & Destructor Documentation

17.2.3.1 `template<typename EdgeContent > graph::Arc< EdgeContent >::Arc () [inline]`

Construct an empty (null) arc.

Definition at line 371 of file `edge.h`.

Referenced by `graph::Arc< EdgeContent >::inv()`.

```

372     : _source(0)
373     , _target(0)
374     , main(0)
375     , other(0)
376     {}

```

17.2.3.2 `template<typename EdgeContent > graph::Arc< EdgeContent >::Arc (const Arc< EdgeContent > & copy) [inline]`

Copy constructor.

Definition at line 387 of file `edge.h`.

```

388     : _source(copy._source)
389     , _target(copy._target)
390     , main(copy.main)
391     , other(copy.other)
392     {}

```

17.2.3.3 `template<typename EdgeContent > graph::Arc< EdgeContent >::Arc (identity_t src, identity_t tgt, EdgeContent * c1, EdgeContent * c2) [inline]`

Full constructor -> should never be called by the user.

Definition at line 379 of file `edge.h`.

```

380     : _source(src)
381     , _target(tgt)
382     , main(c1)
383     , other(c2)
384     {}

```

17.2.3.4 `template<typename EdgeContent > graph::Arc< EdgeContent >::~~Arc () [inline]`

Destroy and copy the content of the arcs.

Definition at line 324 of file `edge.h`.

References `graph::Arc< EdgeContent >::sync()`.

```
324 { sync(); }
```

17.2.4 Member Function Documentation

17.2.4.1 `template<typename EdgeContent > Arc< EdgeContent > graph::Arc< EdgeContent >::inv () const [inline]`

Returns the opposite arc.

Careful, it is not synchronized!

Definition at line 396 of file edge.h.

References graph::Arc< EdgeContent >::Arc().

Referenced by graph::Arc< EdgeContent >::operator-().

```
397 {  
398     return Arc(_target, _source, other, main);  
399 }
```

17.2.4.2 `template<typename EdgeContent > bool graph::Arc< EdgeContent >::isNull () const [inline]`

Test if the content is null.

Definition at line 329 of file edge.h.

```
329 { return main == 0; }
```

17.2.4.3 `template<typename EdgeContent > graph::Arc< EdgeContent >::operator bool () const [inline]`

AN arc evaluates to true if it contains some data.

Definition at line 334 of file edge.h.

```
334 { return main != 0; }
```

17.2.4.4 `template<typename EdgeContent > graph::Arc< EdgeContent >::operator edge_t () const [inline]`

Convert the current arc into the corresponding edge.

Definition at line 304 of file edge.h.

```
305 {  
306     return edge_t(_source, _target, main);  
307 }
```

17.2.4.5 `template<typename EdgeContent > EdgeContent& graph::Arc< EdgeContent >::operator* () const [inline]`

Reference the content of the arc.

Definition at line 339 of file edge.h.

```
339 { return *main; }
```

17.2.4.6 `template<typename EdgeContent > Arc graph::Arc< EdgeContent >::operator- () const [inline]`

Unary '-' operator synchronize edges and returns opposite arc.

Definition at line 319 of file edge.h.

References `graph::Arc< EdgeContent >::inv()`.

```
319 { return inv(); }
```

17.2.4.7 `template<typename EdgeContent > EdgeContent* graph::Arc< EdgeContent >::operator-> () const [inline]`

Reference the content of the arc as a pointer.

Definition at line 343 of file edge.h.

```
343 { return main; }
```

17.2.4.8 `template<typename EdgeContent > identity_t graph::Arc< EdgeContent >::source () const [inline]`

Returns the identifier of the source of the edge.

Note

You should rather use the [VVGraph::source\(\)](#) method that returns the source vertex.

Definition at line 351 of file edge.h.

```
351 { return _source; }
```

17.2.4.9 template<typename EdgeContent > void graph::Arc< EdgeContent >::sync () const [inline]

Synchronize both sides of the arc.

Definition at line 363 of file edge.h.

References graph::copy_symmetric().

Referenced by graph::Arc< EdgeContent >::~~Arc().

```
363 { if(main) copy_symmetric(*other, *main); }
```

17.2.4.10 template<typename EdgeContent > identity_t graph::Arc< EdgeContent >::target () const [inline]

Returns the identifier of the target of the edge.

Note

You should rather use the [VVGraph::target\(\)](#) method that returns the target vertex.

Definition at line 358 of file edge.h.

```
358 { return _target; }
```

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

- [vvelib/graph/edge.h](#)

17.3 util::Shapes::BSplineSurface Class Reference

Class describing a bspline surface.

```
#include <util/bsurface.h>
```

Public Member Functions

- **BSplineSurface** (const [BSplineSurface](#) &B)
- void **CalcFootprint** (unsigned int i, unsigned int j, unsigned int level)
- void **CalcPreImage** (int i, int j, unsigned int level)
- Point3d **ContourEval** (double u) const
- Point2d **ContourInverse** (double u) const
- vector< vector< vector< Point3d > > > * **ControlPoints** ()
- double **ConvertU** (double u) const
- double **ConvertV** (double v) const
- int **DimU** ()
- int **DimV** ()
- void **DisplayControlPoints** (Matrix3d A, Point3d t)
- void **DisplaySurface** ()
- void **DoubleU** ()
- void **DoubleV** ()
- void **Draw** (int uDiv, int vDiv)
Render the object in the current OpenGL context.
- void **DrawCircle** (double x, double y, double z, double w)
- bool **EditLevel** (unsigned int i)
- Point3d **Eval** (double u, double v) const
- Point3d **EvalN** (double u, double v) const
- Point3d **EvalU** (double u, int m) const
- void **GetControlMesh** (int i, int j, double &x, double &y, double &z)
- void **HalveU** ()
- void **HalveV** ()
- float **HasContracted** ([BSplineSurface](#) *b, float u, float v)
- bool **InsideContour** (vector< Point3d > shape, Point3d to_check)
- [BSplineSurface](#) **Interpolate** (const [BSplineSurface](#) &BS1, const [BSplineSurface](#) &BS2, double alpha, int num_rows, int num_cols) const
- void **IsShaded** (bool shaded)
- bool **LoadPatch** (const char *fname)
- [BSplineSurface](#) **LowestRes** ()
- unsigned int **LowestResDim** (int i)
- void **MovePoint** (Point3d m, Matrix3d A, Point3d t)
- void **MoveRegion** (Point3d dis)
- Point3d **NormalEval** (double u, double v) const
- Point3d **NormalEvalN** (double u, double v) const
- [BSplineSurface](#) **operator*** (const double c)

- [BSplineSurface](#) **operator+** (const [BSplineSurface](#) &B)
- [BSplineSurface](#) **operator-** (const [BSplineSurface](#) &B)
- [BSplineSurface](#) & **operator=** (const [BSplineSurface](#) &B)
- Point3d **PartialU** (double u, double v) const
- Point3d **PartialUN** (double u, double v) const
- Point3d **PartialV** (double u, double v) const
- Point3d **PartialVN** (double u, double v) const
- bool **PointInversion** (Point2d &surf_p, Point2d p0, Point3d q) const
- bool **PointInversionN** (Point2d &surf_p, Point2d p0, Point3d q) const
- void [Recompute](#) ()

Force recomputation of the surface.

- void **RenderFootprint** ()
- void **RenderLine** (int i1, int j1, int level1, int i2, int j2, int level2)
- void **RenderSupport** (unsigned int i, unsigned int j, unsigned int level)
- void **Reset** (int u, int v)
- void **Resize** (unsigned int num_rows, unsigned int num_cols)
- bool **SavePatch** (const char *fname)
- vector< int * > **SelectedIndicies** ()
- int **SelectedLevel** ()
- void **SelectPoint** (Point3d n, Matrix3d A, Point3d t)
- void **SelectRegion** (vector< Point3d > s_reg, Matrix3d A, Point3d t)
- void **SetColors** (double c1[3], double c2[3], double c3[3])
- void **SetControlMesh** (int i, int j, double x, double y, double z)
- void **SetPrecision** (double u, double v)
- void **SetPrevious** ([BSplineSurface](#) *b)
- void **ToggleControlMesh** ()
- void **ToggleNormals** ()
- void **TogglePolygonMesh** ()
- void **ToggleShadedMesh** ()
- void **UpdateFootprint** ()
- void **VisualizeContraction** (bool c)

Interface to LPFG

- void [BoundingBox](#) (double &x1, double &y1, double &z1, double &x2, double &y2, double &z2)
Compute the bounding box of the surface.
- bool [IsTextured](#) () const
True if a texture is attached to the surface.
- void [Load](#) (const char *fname, double scale, int TextureId, int divU, int divV)
Load a file describing the [BSplineSurface](#).
- void [Reread](#) ()
Reread the last loaded file (if modified).

- `int TextureId () const`
Get the texture id attached to the object (0 if none).
- `void Transform (Matrix3d M, double x, double y, double z)`
Transform the surface using a 3D linear transformation and a translation.

17.3.1 Detailed Description

Class describing a bspline surface.

Definition at line 37 of file `bsurface.h`.

17.3.2 Member Function Documentation

17.3.2.1 `void util::Shapes::BSplineSurface::BoundingBox (double & x1, double & y1, double & z1, double & x2, double & y2, double & z2)`

Compute the bounding box of the surface.

Definition at line 65 of file `bsurface.cpp`.

```

66 {
67     x1=x2=Control_points[0][0][0][0];
68     y1=y2=Control_points[0][0][0][1];
69     z1=z2=Control_points[0][0][0][2];
70
71     for (unsigned int i=0; i<Control_points[0].size(); i++)
72         for (unsigned int j=0; j<Control_points[0][0].size(); j++) {
73
74             if (Control_points[0][i][j][0]<x1)
75                 x1=Control_points[0][i][j][0];
76
77             if (Control_points[0][i][j][1]<y1)
78                 y1=Control_points[0][i][j][1];
79
80             if (Control_points[0][i][j][2]<z1)
81                 z1=Control_points[0][i][j][2];
82
83             if (Control_points[0][i][j][0]>x2)
84                 x2=Control_points[0][i][j][0];
85
86             if (Control_points[0][i][j][1]>y2)
87                 y2=Control_points[0][i][j][1];
88
89             if (Control_points[0][i][j][2]>z2)
90                 z2=Control_points[0][i][j][2];
91         }
92 }
```

17.3.2.2 `void util::Shapes::BSplineSurface::Draw (int uDiv, int vDiv)`

Render the object in the current OpenGL context.

Definition at line 112 of file bsurface.cpp.

```

113 {
114     if(!precomputed)
115         Precompute(uDiv,vDiv);
116     //const double du = 1.0/uDiv*(n_u-degree);
117     //const double dv = 1.0/vDiv*(n_v-degree);
118
119
120     for (int i=0; i<_divU; ++i) {
121         const double u1 = 1.0*i/uDiv;
122         const double u2 = 1.0*(i+1)/uDiv;
123
124         if(!s_mesh)
125             glBegin(GL_TRIANGLES);
126
127         for (int j=0; j<_divV; ++j) {
128             bool flip=false;
129             int id1 = PtId(i, j);
130             int id2 = PtId(i+1,j);
131             int id3 = PtId(i+1,j+1);
132             int id4 = PtId(i,j+1);
133             const double v1 = 1.0*j/vDiv;
134             const double v2 = 1.0*(j+1)/vDiv;
135
136
137
138
139             if(s_mesh)
140                 glBegin(GL_LINES);
141
142             if(contraction){
143                 if( psurf && _ctract[id4]<0)
144                     glColor3f(0.8*(-_ctract[id4]/(-c_min))+0.2,0.2,0.2);
145                 else
146                     glColor3f(0.2,0.8*(_ctract[id4]/(c_max))+0.2,0.2);
147             }
148
149             glTexCoord2d(u1, v2);
150             glNormal3d(_nrml[id4][0],_nrml[id4][1],_nrml[id4][2]);
151             glVertex3d(_vrtx[id4][0],_vrtx[id4][1],_vrtx[id4][2]);
152
153
154             if(contraction){
155                 if( psurf && _ctract[id1]<0)
156                     glColor3f(0.8*(-_ctract[id1]/(-c_min))+0.2,0.2,0.2);
157                 else
158                     glColor3f(0.2,0.8*(_ctract[id1]/(c_max))+0.2,0.2);
159             }
160
161
162             glTexCoord2d(u1, v1);
163             glNormal3d(_nrml[id1][0],_nrml[id1][1],_nrml[id1][2]);
164             glVertex3d(_vrtx[id1][0],_vrtx[id1][1],_vrtx[id1][2]);
165
166
167             if(_nrml[id4]*_nrml[id2]>_nrml[id3]*_nrml[id1]) {
168
169                 if(contraction){
170                     if( psurf && _ctract[id2]<0)
171                         glColor3f(0.8*(-_ctract[id2]/(-c_min))+0.2,0.2,0.2);
172                     else

```

```

173         glColor3f(0.2,0.8*(_ctract[id2]/(c_max))+0.2,0.2);
174     }
175
176
177     flip=true;
178     glTexCoord2d(u2, v1);
179     glNormal3d(_nrml[id2][0],_nrml[id2][1],_nrml[id2][2]);
180     glVertex3d(_vrtx[id2][0],_vrtx[id2][1],_vrtx[id2][2]);
181
182     if(s_mesh) {
183
184
185         if(contraction){
186             if( psurf && _ctract[id4]<0)
187                 glColor3f(0.8*(-_ctract[id4]/(-c_min))+0.2,0.2,0.2);
188             else
189                 glColor3f(0.2,0.8*(_ctract[id4]/(c_max))+0.2,0.2);
190         }
191
192
193         glTexCoord2d(u1, v2);
194         glNormal3d(_nrml[id4][0],_nrml[id4][1],_nrml[id4][2]);
195         glVertex3d(_vrtx[id4][0],_vrtx[id4][1],_vrtx[id4][2]);
196     }
197
198
199
200     if(contraction){
201         if( psurf && _ctract[id4]<0)
202             glColor3f(0.8*(-_ctract[id4]/(-c_min))+0.2,0.2,0.2);
203         else
204             glColor3f(0.2,0.8*(_ctract[id4]/(c_max))+0.2,0.2);
205     }
206
207
208     glTexCoord2d(u1, v2);
209     glNormal3d(_nrml[id4][0],_nrml[id4][1],_nrml[id4][2]);
210     glVertex3d(_vrtx[id4][0],_vrtx[id4][1],_vrtx[id4][2]);
211
212 } else {
213
214
215     if(contraction){
216         if( psurf && _ctract[id3]<0)
217             glColor3f(0.8*(-_ctract[id3]/(-c_min))+0.2,0.2,0.2);
218         else
219             glColor3f(0.2,0.8*(_ctract[id3]/(c_max))+0.2,0.2);
220     }
221
222
223     glTexCoord2d(u2, v2);
224     glNormal3d(_nrml[id3][0],_nrml[id3][1],_nrml[id3][2]);
225     glVertex3d(_vrtx[id3][0],_vrtx[id3][1],_vrtx[id3][2]);
226
227     if(s_mesh) {
228
229         if(contraction){
230             if( psurf && _ctract[id4]<0)
231                 glColor3f(0.8*(-_ctract[id4]/(-c_min))+0.2,0.2,0.2);
232             else
233                 glColor3f(0.2,0.8*(_ctract[id4]/(c_max))+0.2,0.2);
234         }

```

```
235
236
237     glTexCoord2d(u1, v2);
238     glNormal3d(_nrml[id4][0],_nrml[id4][1],_nrml[id4][2]);
239     glVertex3d(_vrtx[id4][0],_vrtx[id4][1],_vrtx[id4][2]);
240 }
241
242
243     if(contraction){
244         if( psurf && _ctract[id1]<0)
245             glColor3f(0.8*(-_ctract[id1]/(-c_min))+0.2,0.2,0.2);
246         else
247             glColor3f(0.2,0.8*(_ctract[id1]/(c_max))+0.2,0.2);
248     }
249
250
251     glTexCoord2d(u1, v1);
252     glNormal3d(_nrml[id1][0],_nrml[id1][1],_nrml[id1][2]);
253     glVertex3d(_vrtx[id1][0],_vrtx[id1][1],_vrtx[id1][2]);
254 }
255
256
257     if(contraction){
258         if( psurf && _ctract[id2]<0)
259             glColor3f(0.8*(-_ctract[id2]/(-c_min))+0.2,0.2,0.2);
260         else
261             glColor3f(0.2,0.8*(_ctract[id2]/(c_max))+0.2,0.2);
262     }
263
264
265     glTexCoord2d(u2, v1);
266     glNormal3d(_nrml[id2][0],_nrml[id2][1],_nrml[id2][2]);
267     glVertex3d(_vrtx[id2][0],_vrtx[id2][1],_vrtx[id2][2]);
268
269
270     if(contraction){
271         if( psurf && _ctract[id3]<0)
272             glColor3f(0.8*(-_ctract[id3]/(-c_min))+0.2,0.2,0.2);
273         else
274             glColor3f(0.2,0.8*(_ctract[id3]/(c_max))+0.2,0.2);
275     }
276
277
278
279     glTexCoord2d(u2, v2);
280     glNormal3d(_nrml[id3][0],_nrml[id3][1],_nrml[id3][2]);
281     glVertex3d(_vrtx[id3][0],_vrtx[id3][1],_vrtx[id3][2]);
282
283     if(s_mesh) {
284
285         if(flip) {
286
287             if(contraction){
288                 if( psurf && _ctract[id4]<0)
289                     glColor3f(0.8*(-_ctract[id4]/(-c_min))+0.2,0.2,0.2);
290                 else
291                     glColor3f(0.2,0.8*(_ctract[id4]/(c_max))+0.2,0.2);
292             }
293
294
295             glTexCoord2d(u1, v2);
296             glNormal3d(_nrml[id4][0],_nrml[id4][1],_nrml[id4][2]);
```

```

297         glVertex3d(_vrtx[id4][0],_vrtx[id4][1],_vrtx[id4][2]);
298     } else {
299
300
301         if(contraction){
302             if( psurf && _ctract[id1]<0)
303                 glColor3f(0.8*(-_ctract[id1]/(-c_min))+0.2,0.2,0.2);
304             else
305                 glColor3f(0.2,0.8*(_ctract[id1]/(c_max))+0.2,0.2);
306         }
307
308
309         glTexCoord2d(u1, v1);
310         glNormal3d(_nrml[id1][0],_nrml[id1][1],_nrml[id1][2]);
311         glVertex3d(_vrtx[id1][0],_vrtx[id1][1],_vrtx[id1][2]);
312     }
313     glEnd();
314 }
315 }
316 glEnd();
317 }
318
319
320
321
322
323 /*
324 // draw normals
325 {
326     GLprimitive lns(GL_LINES);
327     for (int i=0; i<_vrtx.size(); ++i)
328     {
329         Point3d v = _vrtx[i];
330         glVertex3d(v[0],v[1],v[2]);
331         v = v+_nrml[i]*0.2;
332         glVertex3d(v[0],v[1],v[2]);
333     }
334 }
335 */
336 }

```

17.3.2.3 bool util::Shapes::BSplineSurface::IsTextured() const [inline]

True if a texture is attached to the surface.

Definition at line 59 of file bsurface.h.

```

60     { return _TextureId >= 0; }

```

17.3.2.4 void util::Shapes::BSplineSurface::Load (const char **fname*, double *scale*, int *TextureId*, int *divU*, int *divV*)

Load a file describing the [BSplineSurface](#).

Parameters

fname File name

scale Scale modification

TextureId texture used for rendering

divU number of division in U to use in the description

divV number of division in V to use in the description

Definition at line 48 of file bsurface.cpp.

```

49 {
50     _TextureId=TextureId;
51     _scale=scale;
52     _divU=divU;
53     _divV=divV;
54     LoadPatch(fnm);
55     precomputed=false;
56 }
```

17.3.2.5 void util::Shapes::BSplineSurface::Recompute() [inline]

Force recomputation of the surface.

Definition at line 71 of file bsurface.h.

```

72     {precomputed=false;}
```

17.3.2.6 void util::Shapes::BSplineSurface::Reread()

Reread the last loaded file (if modified).

Definition at line 58 of file bsurface.cpp.

```

59 {
60     _divU = _divV = -1;
61     LoadPatch(_fname.c_str());
62     precomputed=false;
63 }
```

17.3.2.7 int util::Shapes::BSplineSurface::TextureId() const [inline]

Get the texture id attached to the object (0 if none).

Definition at line 63 of file bsurface.h.

```

64     { return _TextureId; }
```

17.3.2.8 void util::Shapes::BSplineSurface::Transform (Matrix3d *M*, double *x*, double *y*, double *z*)

Transform the surface using a 3D linear transformation and a translation.

Definition at line 94 of file bsurface.cpp.

```
95 {
96     Point3d D;
97     D[0]=x;
98     D[1]=y;
99     D[2]=z;
100
101     for (unsigned int i=0; i<Control_points[0].size(); i++)
102         for (unsigned int j=0; j<Control_points[0][0].size(); j++) {
103
104
105             Control_points[0][i][j]=M*(Control_points[0][i][j]);
106             Control_points[0][i][j]=Control_points[0][i][j]+D;
107         }
108
109     precomputed=false;
110 }
```

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

- vvelib/util/bsurface.h
- vvelib/util/bsurface.cpp

17.4 util::Buffer< T, size > Class Template Reference

A ranged checked array.

```
#include <util/buffer.h>
```

Public Member Functions

- [Buffer](#) (const T &val=T())
Constructor.
- T * [c_data](#) ()
Return the data as a C-style array.
- T & [operator\[\]](#) (unsigned int i) throw (std::out_of_range)
Index operator.
- virtual [~Buffer](#) ()
Destructor.

17.4.1 Detailed Description

```
template<class T, unsigned int size> class util::Buffer< T, size >
```

A ranged checked array.

Parameters

T The type to store.

size The size of the array.

Definition at line 20 of file buffer.h.

17.4.2 Constructor & Destructor Documentation

17.4.2.1 `template<class T , unsigned int size> util::Buffer< T, size >::Buffer
(const T & val = T ()) [inline]`

Constructor.

Parameters

val The initial value for the elements of [Buffer](#).

Definition at line 72 of file buffer.h.

```

72     {
73     for (unsigned int i = 0; i < size; i++) data[i] = val;
74 }

```

17.4.2.2 `template<class T, unsigned int size> util::Buffer< T, size >::~Buffer()` `[inline, virtual]`

Destructor.

Definition at line 78 of file buffer.h.

```

78 {}

```

17.4.3 Member Function Documentation

17.4.3.1 `template<class T, unsigned int size> T * util::Buffer< T, size >::c_data()` `[inline]`

Return the data as a C-style array.

This function allows the data in [Buffer](#) to be passed as a pointer to a C-style array. Like all C-style arrays, it should be used with caution as access is not checked.

Definition at line 101 of file buffer.h.

```

101     {
102     return data;
103 }

```

17.4.3.2 `template<class T, unsigned int size> T & util::Buffer< T, size >::operator[] (unsigned int i) throw (std::out_of_range)` `[inline]`

Index operator.

Parameters

i The index to reference.

The index operator is range checked. If *i* is out of range, `std::out_of_range` is thrown.

Definition at line 88 of file buffer.h.

```

88     {
89     if (i < size) return data[i];
90     else throw std::out_of_range("Buffer index is out of range.");
91 }

```

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

- [vvelib/util/buffer.h](#)

17.5 algorithms::TriangleSurface::Cell Struct Reference

Type of a cell, i.e.

```
#include <triangle_growth.h>
```

Public Attributes

- int [id](#)
Id of the cell.
- Point3d [normal](#)
Normal to the cell.

17.5.1 Detailed Description

Type of a cell, i.e. a triangle

Definition at line 37 of file triangle_growth.h.

17.5.2 Member Data Documentation

17.5.2.1 int algorithms::TriangleSurface::Cell::id

Id of the cell.

Definition at line 42 of file triangle_growth.h.

17.5.2.2 Point3d algorithms::TriangleSurface::Cell::normal

Normal to the cell.

Definition at line 40 of file triangle_growth.h.

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

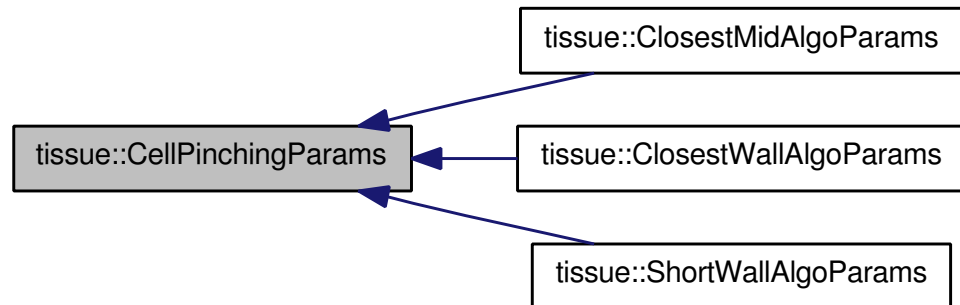
- vvelib/algorithms/[triangle_growth.h](#)

17.6 tissue::CellPinchingParams Struct Reference

Parameters of the cell pinching algorithm.

```
#include <tissue.h>
```

Inheritance diagram for tissue::CellPinchingParams:



Public Member Functions

- `CellPinchingParams` (const `CellPinchingParams` ©)

Copy constructor.

- `CellPinchingParams` (double cp=0, double cmp=0)

Default and initialization constructor.

Public Attributes

- double `cellMaxPinch`

Maximum displacement due to the pinching.

- double `cellPinch`

Pinching ratio.

17.6.1 Detailed Description

Parameters of the cell pinching algorithm.

Definition at line 73 of file `tissue.h`.

17.6.2 Constructor & Destructor Documentation

17.6.2.1 tissue::CellPinchingParams::CellPinchingParams (double *cp* = 0, double *cmp* = 0) [inline]

Default and initialization constructor.

Definition at line 78 of file tissue.h.

```
79         : cellPinch(cp)
80         , cellMaxPinch(cmp)
81     { }
```

17.6.2.2 tissue::CellPinchingParams::CellPinchingParams (const CellPinchingParams & *copy*) [inline]

Copy constructor.

Definition at line 86 of file tissue.h.

```
87         : cellPinch(copy.cellPinch)
88         , cellMaxPinch(copy.cellMaxPinch)
89     { }
```

17.6.3 Member Data Documentation

17.6.3.1 double tissue::CellPinchingParams::cellMaxPinch

Maximum displacement due to the pinching.

Definition at line 98 of file tissue.h.

Referenced by tissue::cellPinching(), and tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::setCellPinchingParams().

17.6.3.2 double tissue::CellPinchingParams::cellPinch

Pinching ratio.

Definition at line 94 of file tissue.h.

Referenced by tissue::cellPinching(), and tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::setCellPinchingParams().

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

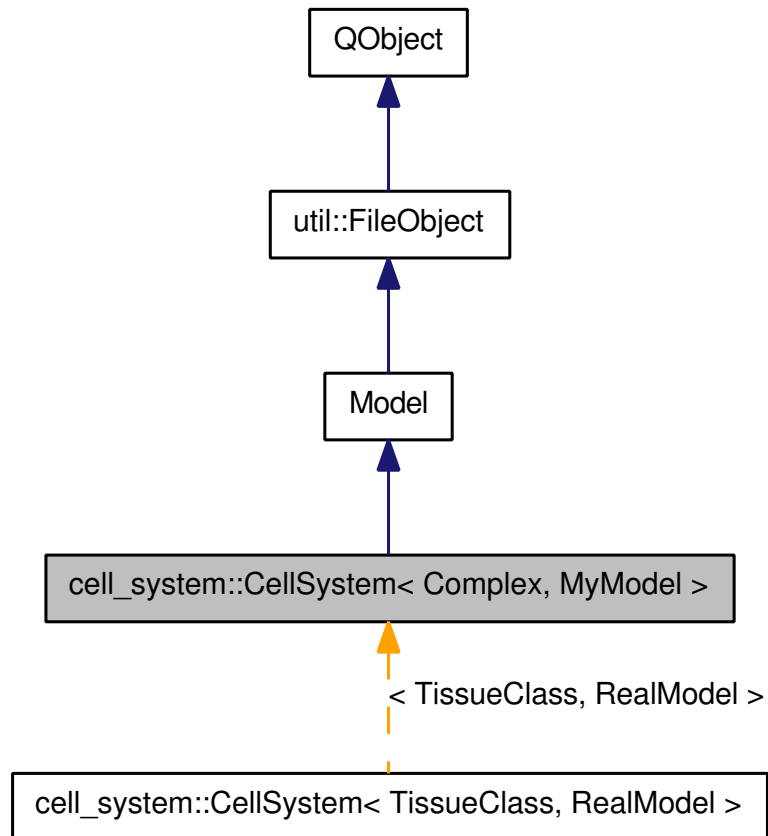
- vvelib/algorithms/[tissue.h](#)

17.7 cell_system::CellSystem< Complex, MyModel > Class Template Reference

Full cell-system model.

```
#include <algorithms/cellsystem.h>
```

Inheritance diagram for cell_system::CellSystem< Complex, MyModel >:



Public Types

- `typedef std::map< label_t, DivisionParams > division_rules_t`

Type of rules of type $A \rightarrow B$ $|(a,r)$ C .

- `typedef std::map< label_t, label_t > label_change_rules_t`

Type of rules of type $A \rightarrow B$.

Public Member Functions

- double **area** (const [cell](#) &c) const
- [CellSystem](#) ([QObject](#) *parent)
Constructor .
- **IMPORT_COMPLEX_EDGES** (Complex)
- **IMPORT_COMPLEX_GRAPHS** (Complex)
- **IMPORT_COMPLEX_VERTICES** (Complex)
- [label_t](#) **label** ([QString](#) name)
Returns the label number of a symbol.
- void **modifiedFiles** (const std::set< [QString](#) > &filenames)
- [Point3d](#) **normal** (const [cell](#) &) const
- [Point3d](#) **normal** (const [junction](#) &) const
- [Point3d](#) **position** (const [junction](#) &v) const
- [Point3d](#) **position** (const [cell](#) &v) const
- void **readMechanicParam** ([util::Params](#) &parms, const [QString](#) §ion)
Read the mechanical parameters.
- virtual void **readParam** ()
Read the parameters.
- void **readRules** ([util::Params](#) &parms, const [QString](#) §ion)
Read a prepare the cell-system rules.
- void **readSymbols** ([util::Params](#) &parms, const [QString](#) §ion)
Read the list of symbols.
- [label_t](#) **registerSymbol** ([QString](#) name)
Register a new symbol to be used in the cell system.
- bool **serialize** ([storage::VVEStorage](#) &store)
Reimplement this method to load/save the current state of the model.
- void **setPosition** (const [junction](#) &v, const [Point3d](#) &pos)
- void **setPosition** (const [cell](#) &v, const [Point3d](#) &pos)
- void **setPositionHint** (const [junction](#) &, const [junction](#) &, const [junction](#) &, double)
- void **step** ()
Default step method, just applies the rules and do the mechanics.
- bool **step_cellsystem** ()
Cell system step.
- void **step_cellsystem_division** ()

Divide the cells according to their current geometry and the rules in the parameter file.

- bool `step_cellsystem_meca` ()
Compute the mecanic stability of the system for a time `showInterval`.
- void `updateCellsArea` ()
Update the area and the center of all the cells.
- void `updateFromOld` (const `cell` &cl, const `cell` &cr, const `cell` &c, const type-name `Complex::division_data` &, `Complex` &T)
- `QString` `version` () const
- int `versionNumber` (const `QString` &version)

Public Attributes

- `DivisionParams` `current_div`
Current division rule.
- double `damping`
Damping parameter.
- `division_rules_t` `division_rules`
Rules of type $A \dashrightarrow B \mid (a,r) C$.
- double `dt`
Time step for mechanical simulation.
- double `ke`
Stiffness of an external spring.
- double `ki`
Stiffness of an internal spring.
- `label_change_rules_t` `label_change_rules`
Rules of type $A \dashrightarrow B$.
- `std::vector< QString >` `label_names`
Mapping from the label numbers to the label strings.
- `std::map< QString, label_t >` `label_numbers`
Mapping from the label names to the label numbers.
- double `mass`
Mass of the junctions.

17.7 `cell_system::CellSystem< Complex, MyModel >` Class Template Reference

- double `pressure`
Pressure coefficient in a cell.
- double `restLength`
Restlength of the springs.
- double `showInterval`
Time step between two images.
- double `showTime`
Time since the last image.
- double `stability`
Stability criteria.
- Complex `T`
Cell complex.
- double `time`
Current time.
- bool `viewRulesApplication`
Set to true to display the rules when applied.
- bool `viewRulesDefinitions`
Set to true to display the rules when analyzed.

17.7.1 Detailed Description

`template<class Complex, class MyModel> class cell_system::CellSystem< Complex, MyModel >`

Full cell-system model. Inherit this model to use cell-system with rules defined in the parameter file.

Parameters

Complex Cell complex class to be used (allows for sub-type of `vvcomplex::VVComplex` as `tissue::Tissue`)

Model Subclass used. Useful to initialise the cell complex class

Definition at line 418 of file `cellsystem.h`.

17.7.2 Member Typedef Documentation

17.7.2.1 `template<class Complex, class MyModel> typedef std::map<label_t, DivisionParams> cell_system::CellSystem< Complex, MyModel >::division_rules_t`

Type of rules of type A --> B |(a,r) C.

Definition at line 480 of file cellsystem.h.

17.7.2.2 `template<class Complex, class MyModel> typedef std::map<label_t, label_t> cell_system::CellSystem< Complex, MyModel >::label_change_rules_t`

Type of rules of type A --> B.

Definition at line 476 of file cellsystem.h.

17.7.3 Constructor & Destructor Documentation

17.7.3.1 `template<class Complex, class MyModel> cell_system::CellSystem< Complex, MyModel >::CellSystem (QObject *parent) [inline]`

Constructor .

..

Definition at line 778 of file cellsystem.h.

```

779      : Model( parent )
780      , T( (MyModel*)this)
781      , showTime(0)
782      , time(0)
783      , viewRulesDefinitions(false)
784      , viewRulesApplication(false)
785      {
786      }
```

17.7.4 Member Function Documentation

17.7.4.1 `template<class Complex, class MyModel> label_t cell_system::CellSystem< Complex, MyModel >::label (QString name) [inline]`

Returns the label number of a symbol.

Parameters

name Name of the symbol

If the symbol do not exist, it is created

17.7 cell_system::CellSystem< Complex, MyModel > Class Template Reference

Definition at line 582 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readRules(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division().

```
583     {
584         std::map<QString, label_t>::const_iterator found = label_numbers.find(name);

585         if(found != label_numbers.end())
586         {
587             return found->second;
588         }
589         cout << "Error, symbol '" << name.toStdString() << "' used but not defined"
590         << endl;
591         return -1;
592     }
```

17.7.4.2 template<class Complex, class MyModel> void cell_system::CellSystem< Complex, MyModel >::readMechanicParam (util::Parms & parms, const QString & section) [inline]

Read the mechanical parameters.

The mechanical parameters include: * dt: Time step for mechanical simulation * ki: Stiffness of internal springs * ke: Stability of external springs * ShowInterval: Time step between two images * Pressure: Pressure coefficient * Damping: Damping coefficient * RestLength: Rest length of the springs * Stability: Max force allowed on a junction to consider the system at equilibrium * Mass: Mass of a junction

Parameters

parms Parameter object to read from

section Section containing the mechanical parameters

Definition at line 560 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readParam().

```
561     {
562         parms(section, "dt", dt);
563         parms(section, "ki", ki);
564         parms(section, "ke", ke);
565         parms(section, "ShowInterval", showInterval);
566         if(showInterval < dt)
567             showInterval = dt;
568         parms(section, "Pressure", pressure);
569         parms(section, "Damping", damping);
570         parms(section, "RestLength", restLength);
571         parms(section, "Stability", stability);
572         parms(section, "Mass", mass);
573     }
```

17.7.4.3 `template<class Complex, class MyModel> virtual void cell_system::CellSystem< Complex, MyModel >::readParam () [inline, virtual]`

Read the parameters.

Usefull if no other parameters are needed (i.e. very unlikely)

Definition at line 761 of file cellsystem.h.

```

762     {
763         util::Parms parms("view.v");
764         readMechanicParam(parms, "Mechanic");
765         readSymbols(parms, "Symbols");
766         readRules(parms, "Rules");
767     }
```

17.7.4.4 `template<class Complex, class MyModel> void cell_system::CellSystem< Complex, MyModel >::readRules (util::Parms & parms, const QString & section) [inline]`

Read a prepare the cell-system rules.

Parameters

parms Parameter object to read from

section Section containing the rules (and only them)

Warning

[readSymbols\(\)](#) has to be called **before** readRules

The rules can take three form: # A: B, correspond to a relabelling $A \rightarrow B$ # A: B |(a,r) C, correspond to a cell division. a is the angle between the reference vector and the normal to the new wall, r is the volume ration V_B/V_A . B and C and the labels of the new cells. If r is 0 then the wall goes through the center of the cell. # A: B |(a) C, is the same as the above rule but with $r = 0.5$.

There can be only one rule per symbol (i.e. no two rule can share the same left hand side symbol).

Definition at line 656 of file cellsystem.h.

Referenced by `cell_system::CellSystem< TissueClass, RealModel >::readParam()`.

```

657     {
658         typedef std::map<QString, std::vector<QString> > rmap;
659         rmap rules;
660         parms.all(section, rules);
661
662         for(rmap::iterator it = rules.begin() ; it != rules.end() ; ++it)
663         {
664             QString name = it->first;
```

17.7 cell_system::CellSystem< Complex, MyModel > Class Template Reference

```
665         label(name);
666     }
667
668     label_change_rules.clear();
669     division_rules.clear();
670
671     for(rmap::iterator it = rules.begin() ; it != rules.end() ; ++it)
672     {
673         label_t lhs = label_numbers[it->first];
674         if(it->second.size() > 1)
675         {
676             cout << "Error, more than one rule for '" << lhs << "' ... ignoring" <<
endl;
677             continue;
678         }
679         QString rhs = it->second[0];
680         int pos = rhs.indexOf('|');
681         if(pos == -1)
682         {
683             label_t new_label = label(rhs);
684             label_change_rules[lhs] = new_label;
685             if(viewRulesDefinitions)
686                 cout << "Rule: " << label_names[lhs].toString() << " --> " <<
label_names[new_label].toString() << endl;
687         }
688         else
689         {
690             if(pos == -1)
691             {
692                 cout << "Error, division rule without '|': " << lhs << " --> " << rhs.
toString() << endl;
693                 continue;
694             }
695             QString left_name = rhs.mid(0, pos).trimmed();
696             label_t left_label = label(left_name);
697             int pos_left_paren, pos_right_paren, pos_comma;
698             pos_left_paren = rhs.indexOf('(', pos);
699             pos_comma = rhs.indexOf(',', pos_left_paren);
700             pos_right_paren = rhs.indexOf(')', pos_left_paren);
701             if(pos_left_paren == -1)
702             {
703                 cout << "Error, division rule without '(': " << lhs << " --> " << rhs.
toString() << endl;
704                 continue;
705             }
706             if(pos_right_paren == -1)
707             {
708                 cout << "Error, division rule without ')': " << lhs << " --> " << rhs.
toString() << endl;
709                 continue;
710             }
711             QString right_name = rhs.mid(pos_right_paren+1).trimmed();
712             label_t right_label = label(right_name);
713             if(pos_comma != -1)
714             {
715                 QString angle_str = rhs.mid(pos_left_paren+1, pos_comma-pos_left_pare
n-1);
716                 QString ratio_str = rhs.mid(pos_comma+1, pos_right_paren-pos_comma-1)
;
717                 bool ok;
718                 double angle = angle_str.toDouble(&ok);
719                 if(!ok)
```

```

720         {
721             cout << "Error, cannot read angle (" << angle_str.toStdString() <<
") in " << lhs << " --> " << rhs.toStdString() << endl;
722             continue;
723         }
724         angle *= M_PI/180;
725         double ratio = ratio_str.toDouble(&ok);
726         if(!ok)
727         {
728             cout << "Error, cannot read ratio (" << ratio_str.toStdString() <<
") in " << lhs << " --> " << rhs.toStdString() << endl;
729             continue;
730         }
731         division_rules[lhs] = DivisionParams(left_label, right_label, angle,
ratio);
732     }
733     else
734     {
735         QString angle_str = rhs.mid(pos_left_paren+1, pos_right_paren-pos_lef
t_paren-1);
736         bool ok;
737         double angle = angle_str.toDouble(&ok);
738         if(!ok)
739         {
740             cout << "Error, cannot read angle (" << angle_str.toStdString() <<
") in " << lhs << " --> " << rhs.toStdString() << endl;
741             continue;
742         }
743         angle *= M_PI/180;
744         division_rules[lhs] = DivisionParams(left_label, right_label, angle);

745     }
746     const DivisionParams& dp = division_rules[lhs];
747     if(viewRulesDefinitions)
748         cout << "Rule: " << label_names[lhs].toStdString() << " --> " <<
label_names[dp.left_label].toStdString() << " | ("
749             << (dp.angle*180/M_PI) << ", "
750             << dp.ratio << ") " << label_names[dp.right_label].toStdString() <<
endl;
751     }
752 }
753 cout << endl;
754 }

```

17.7.4.5 `template<class Complex, class MyModel> void cell_system::CellSystem< Complex, MyModel >::readSymbols (util::Parms & parms, const QString & section) [inline]`

Read the list of symbols.

The symbols are the keys and may have values (but it is not necessary)

Definition at line 599 of file cellsystem.h.

Referenced by `cell_system::CellSystem< TissueClass, RealModel >::readParam()`.

```

600     {
601         typedef std::map<QString, std::vector<QString> > rmap;
602         rmap symbols;

```

17.7 `cell_system::CellSystem< Complex, MyModel >` Class Template Reference

```
603     parms.all(section, symbols);
604     if(viewRulesDefinitions)
605         util::out << "Reading " << symbols.size() << " symbols from section [" <<
        section << "]" << endl;
606     for(rmap::iterator it = symbols.begin() ; it != symbols.end() ; ++it)
607     {
608         QString name = it->first;
609         registerSymbol(name);
610     }
611 }
```

17.7.4.6 `template<class Complex, class MyModel> label_t cell_system::CellSystem< Complex, MyModel >::registerSymbol (QString name) [inline]`

Register a new symbol to be used in the cell system.

If the symbol already exists, return the id of the existing one.

name Name of the new symbol

Returns

The id of the symbol

Definition at line 622 of file `cellsystem.h`.

Referenced by `cell_system::CellSystem< TissueClass, RealModel >::readSymbols()`.

```
623     {
624         std::map<QString, label_t>::iterator found = label_numbers.find(name);
625         if(found == label_numbers.end())
626         {
627             label_t new_label = label_names.size();
628             label_names.push_back(name);
629             label_numbers[name] = new_label;
630             if(viewRulesDefinitions)
631                 cout << "Defining symbol '" << name.toStdString() << "' as value " << n
        ew_label << endl;
632             return new_label;
633         }
634         return found->second;
635     }
```

17.7.4.7 `template<class Complex, class MyModel> bool cell_system::CellSystem< Complex, MyModel >::serialize (storage::VVEStorage &) [inline, virtual]`

Reimplement this method to load/save the current state of the model.

Returns

true if serialization was successful, false otherwise.

Parameters

store Object to store the state in the model in.

Reimplemented from [Model](#).

Definition at line 997 of file cellsystem.h.

```

998     {
999         if(!store.field("T", T))
1000         {
1001             return false;
1002         }
1003         if(!store.field("Time", time))
1004         {
1005             std::cerr << "Warning, cell system saved without time. Setting time to 0"
<< endl;
1006             time = 0;
1007         }
1008         return true;
1009     }

```

17.7.4.8 **template<class Complex, class MyModel> void cell_system::CellSystem< Complex, MyModel >::step () [inline, virtual]**

Default step method, just applies the rules and do the mechanics.

Implements [Model](#).

Definition at line 791 of file cellsystem.h.

```

792     {
793         step_cellsystem();
794     }

```

17.7.4.9 **template<class Complex, class MyModel> bool cell_system::CellSystem< Complex, MyModel >::step_cellsystem () [inline]**

Cell system step.

Returns true if the rules were applied, false if just the mechanic was done (i.e. the system is not stabilised)

Definition at line 903 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::step().

```

904     {
905         if(!this->step_cellsystem_meca())
906             return false;
907
908         step_cellsystem_division();
909         return true;
910     }

```


17.7 cell_system::CellSystem< Complex, MyModel > Class Template Reference

17.7.4.10 template<class Complex, class MyModel> void cell_system::CellSystem< Complex, MyModel >::step_cellsystem_division () [inline]

Divide the cells according to their current geometry and the rules in the parameter file.

Definition at line 864 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem().

```
865     {
866         updateCellsArea();
867
868         // Second, division
869         std::vector<cell> cells(T.C.begin(), T.C.end());
870         forall(const cell& c, cells)
871         {
872             label_t label = c->label;
873             label_change_rules_t::const_iterator label_found = label_change_rules.find
874             d(label);
875             division_rules_t::const_iterator division_found = division_rules.find(label);
876             if (label_found != label_change_rules.end())
877             {
878                 if (viewRulesApplication)
879                     cout << "Applying rule: " << c->label << " --> " << label_found->second
880                     << endl;
881                 c->label = label_found->second;
882             }
883             else if (division_found != division_rules.end())
884             {
885                 const DivisionParams& div = division_found->second;
886                 if (viewRulesApplication)
887                     cout << "Applying rule: " << c->label << " --> "
888                     << div.left_label << " | (" << (div.angle*180/M_PI) << ", " << div.ratio
889                     << " ) " << div.right_label << endl;
890                 current_div = div;
891                 Point3d ref = ((MyModel*)this)->referenceVector(c);
892                 ref.normalize();
893                 CellSystemDivisionParams parms(ref, div.angle, div.ratio, EPSILON);
894                 T.divideCell(c, parms, c);
895             }
896         }
897     }
```

17.7.4.11 template<class Complex, class MyModel> bool cell_system::CellSystem< Complex, MyModel >::step_cellsystem_meca () [inline]

Compute the mechanic stability of the system for a time showInterval.

Return true if the system is stable, false otherwise.

Definition at line 801 of file cellsystem.h.

Referenced by `cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem()`.

```

802     {
803         time += dt;
804         bool stable;
805         do
806         {
807             showTime += dt;
808             // First, mechanical model
809             updateCellsArea();
810             forall(const junction& v, T.W)
811             {
812                 v->force = 0;
813             }
814             forall_named(const cell& c, T.S, cells)
815             {
816                 double area= c->area;
817                 // First, compute the forces due to the pressure
818                 forall(const junction& j, T.S.neighbors(c))
819                 {
820                     const junction& k = T.S.nextTo(c, j);
821                     Point3d u = k->pos - j->pos;
822                     double l = util::norm(u);
823                     u /= l;
824                     Point3d n(u.y(), -u.x(), 0);
825                     n *= pressure*l/area/2;
826                     j->force += n;
827                     k->force += n;
828                 }
829             }
830             // Second, compute the forces due to the springs
831             forall(const junction& j, T.W)
832             {
833                 const Point3d& jpos = j->pos;
834                 forall(const junction& k, T.W.neighbors(j))
835                 {
836                     double ks = ki;
837                     if(T.border(j, k))
838                         ks = ke;
839                     Point3d u = k->pos - jpos;
840                     double l = util::norm(u);
841                     u *= ks*(1 - restLength/l);
842                     j->force += u;
843                 }
844             }
845             // Now, integrate and check for stability
846             stable = true;
847             forall(const junction& j, T.W)
848             {
849                 if(util::norm(j->force) > stability) stable = false;
850                 j->velocity += (j->force - damping*j->velocity)/mass*dt;
851                 j->pos += j->velocity*dt;
852             }
853         } while(showTime < showInterval);
854         showTime -= showInterval;
855
856         return stable;
857     }
858 }
```

17.7 cell_system::CellSystem< Complex, MyModel > Class Template Reference 221

17.7.4.12 template<class Complex, class MyModel> void cell_system::CellSystem< Complex, MyModel >::updateCellsArea () [inline]

Update the area and the center of all the cells.

Definition at line 517 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca().

```
518     {
519         forall_named( const cell& c, T.S, cells)
520         {
521             Point3d center;
522             double np = 0;
523             c->area = 0.0;
524             forall(const junction& n, T.S.neighbors(c))
525             {
526                 // Find area
527                 const junction& m = T.S.nextTo(c, n);
528                 c->area += geometry::triangleArea(c->pos, m->pos, n->pos);
529                 center += n->pos;
530                 np++;
531             }
532             center /= np;
533             c->pos = center;
534             if(c->area <= EPSILON ) // Avoid inf
535             {
536                 cout << "Bad area " << c->area << endl;
537                 c->area = 0.1;
538             }
539         }
540     }
```

17.7.4.13 template<class Complex, class MyModel> QString cell_system::CellSystem< Complex, MyModel >::version () const [inline, virtual]

Returns

the current version of the class.

The default implementation returns an empty string

Reimplemented from [Model](#).

Definition at line 992 of file cellsystem.h.

```
993     {
994         return "CellSystemModel 1.0";
995     }
```

17.7.4.14 `template<class Complex, class MyModel> int
cell_system::CellSystem< Complex, MyModel >::versionNumber
(const QString &) [inline, virtual]`

Returns

an integer representing the version string for the model.

The default implementation always return 0.

The number has no meaning outside the model.

Reimplemented from [Model](#).

Definition at line 987 of file cellsystem.h.

Referenced by `cell_system::CellSystem< TissueClass, RealModel >::versionNumber()`.

```
988     {
989         return storage::versionNumber(version);
990     }
```

17.7.5 Member Data Documentation

17.7.5.1 `template<class Complex, class MyModel> DivisionParams
cell_system::CellSystem< Complex, MyModel >::current_div`

Current division rule.

Used to setup the cell labels in `updateFromOld`

Definition at line 503 of file cellsystem.h.

Referenced by `cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division()`.

17.7.5.2 `template<class Complex, class MyModel> double
cell_system::CellSystem< Complex, MyModel >::damping`

Damping parameter.

Definition at line 455 of file cellsystem.h.

Referenced by `cell_system::CellSystem< TissueClass, RealModel >::readMechanicParam()`, and `cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca()`.

17.7.5.3 `template<class Complex, class MyModel> division_rules_t
cell_system::CellSystem< Complex, MyModel >::division_rules`

Rules of type $A \rightarrow B \mid (a,r) C$.

Definition at line 488 of file cellsystem.h.

17.7 cell_system::CellSystem< Complex, MyModel > Class Template Reference

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readRules(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division().

17.7.5.4 template<class Complex, class MyModel> double cell_system::CellSystem< Complex, MyModel >::dt

Time step for mechanical simulation.

Definition at line 431 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readMechanicParam(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca().

17.7.5.5 template<class Complex, class MyModel> double cell_system::CellSystem< Complex, MyModel >::ke

Stiffness of an external spring.

Definition at line 443 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readMechanicParam(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca().

17.7.5.6 template<class Complex, class MyModel> double cell_system::CellSystem< Complex, MyModel >::ki

Stiffness of an internal spring.

Definition at line 447 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readMechanicParam(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca().

17.7.5.7 template<class Complex, class MyModel> label_change_rules_t cell_system::CellSystem< Complex, MyModel >::label_change_rules

Rules of type A --> B.

Definition at line 484 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readRules(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division().

17.7.5.8 **template<class Complex, class MyModel> std::vector<QString> cell_system::CellSystem< Complex, MyModel >::label_names**

Mapping from the label numbers to the label strings.

Definition at line 496 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readRules(), and cell_system::CellSystem< TissueClass, RealModel >::registerSymbol().

17.7.5.9 **template<class Complex, class MyModel> std::map<QString, label_t> cell_system::CellSystem< Complex, MyModel >::label_numbers**

Mapping from the label names to the label numbers.

Definition at line 492 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::label(), cell_system::CellSystem< TissueClass, RealModel >::readRules(), and cell_system::CellSystem< TissueClass, RealModel >::registerSymbol().

17.7.5.10 **template<class Complex, class MyModel> double cell_system::CellSystem< Complex, MyModel >::mass**

Mass of the junctions.

Definition at line 467 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readMechanicParam(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca().

17.7.5.11 **template<class Complex, class MyModel> double cell_system::CellSystem< Complex, MyModel >::pressure**

Pressure coefficient in a cell.

Definition at line 451 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readMechanicParam(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca().

17.7.5.12 **template<class Complex, class MyModel> double cell_system::CellSystem< Complex, MyModel >::restLength**

Restlength of the springs.

Definition at line 459 of file cellsystem.h.

17.7 cell_system::CellSystem< Complex, MyModel > Class Template Reference

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readMechanicParam(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca().

17.7.5.13 template<class Complex, class MyModel> double cell_system::CellSystem< Complex, MyModel >::showInterval

Time step between two images.

Definition at line 435 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readMechanicParam(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca().

17.7.5.14 template<class Complex, class MyModel> double cell_system::CellSystem< Complex, MyModel >::showTime

Time since the last image.

Definition at line 439 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca().

17.7.5.15 template<class Complex, class MyModel> double cell_system::CellSystem< Complex, MyModel >::stability

Stability criteria.

Definition at line 463 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readMechanicParam(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca().

17.7.5.16 template<class Complex, class MyModel> Complex cell_system::CellSystem< Complex, MyModel >::T

Cell complex.

Definition at line 427 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division(), cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca(), and cell_system::CellSystem< TissueClass, RealModel >::updateCellsArea().

17.7.5.17 `template<class Complex, class MyModel> double
cell_system::CellSystem< Complex, MyModel >::time`

Current time.

Definition at line 471 of file `cellsystem.h`.

Referenced by `cell_system::CellSystem< TissueClass, RealModel >::serialize()`, and `cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca()`.

17.7.5.18 `template<class Complex, class MyModel> bool
cell_system::CellSystem< Complex, MyModel
>::viewRulesApplication`

Set to true to display the rules when applied.

Definition at line 512 of file `cellsystem.h`.

Referenced by `cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division()`.

17.7.5.19 `template<class Complex, class MyModel> bool
cell_system::CellSystem< Complex, MyModel
>::viewRulesDefinitions`

Set to true to display the rules when analyzed.

Definition at line 508 of file `cellsystem.h`.

Referenced by `cell_system::CellSystem< TissueClass, RealModel >::readRules()`, `cell_system::CellSystem< TissueClass, RealModel >::readSymbols()`, and `cell_system::CellSystem< TissueClass, RealModel >::registerSymbol()`.

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

- `vvelib/algorithms/cellsystem.h`

17.8 cell_system::CellSystemCell Class Reference

Content of a vertex for the 2D cell system.

```
#include <algorithms/cellsystem.h>
```

Public Member Functions

- bool [serialize](#) (storage::VVEStorage &)
Serialization method.

Public Attributes

- double [area](#)
Area of the cell.
- [label_t](#) [label](#)
Label of the cell.
- Point3d [pos](#)
Position of the vertex.

17.8.1 Detailed Description

Content of a vertex for the 2D cell system.

Definition at line 311 of file cellsystem.h.

17.8.2 Member Function Documentation

17.8.2.1 bool cell_system::CellSystemCell::serialize (storage::VVEStorage & store)

Serialization method.

Definition at line 11 of file cellsystem.cpp.

References [area](#), [storage::VVEStorage::field\(\)](#), [label](#), and [pos](#).

```
12 {  
13     if(!store.field("Area", area))  
14     {  
15         area = 0;  
16         cerr << "Warning, reading cell with invalid area" << endl;  
17     }  
18     if(!store.field("Position", pos))  
19         return false;
```

```
20     if(!store.field("Label", label))
21         return false;
22     return true;
23 }
```

17.8.3 Member Data Documentation

17.8.3.1 `double cell_system::CellSystemCell::area`

Area of the cell.

Definition at line 316 of file `cellsystem.h`.

Referenced by `serialize()`.

17.8.3.2 `label_t cell_system::CellSystemCell::label`

Label of the cell.

Definition at line 324 of file `cellsystem.h`.

Referenced by `serialize()`.

17.8.3.3 `Point3d cell_system::CellSystemCell::pos`

Position of the vertex.

Definition at line 320 of file `cellsystem.h`.

Referenced by `serialize()`.

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

- `vvelib/algorithms/cellsystem.h`
- `vvelib/algorithms/cellsystem.cpp`

17.9 cell_system::CellSystemDivisionParams Class Reference

Parameters for cell-system division.

```
#include <algorithms/cellsystem.h>
```

Public Member Functions

- [CellSystemDivisionParams](#) (const [CellSystemDivisionParams](#) ©)
Copy constructor.
- [CellSystemDivisionParams](#) (const [Point3d](#) &dir, double a=0, double r=0.5, double eps=0.000001, bool pinch=false, [tissue::CellPinchingParams](#) pp=[tissue::CellPinchingParams\(\)](#))
Constructor.

Public Attributes

- double [angle](#)
Angle between the direction and the normal to the wall.
- [Point3d](#) [direction](#)
Reference direction to find the normal of the new wall.
- double [epsilon](#)
Relative epsilon for floating point comparison.
- double [minCellWall](#)
Minimum size of a cell wall.
- bool [pinching](#)
Ask for pinching after the cell wall was found.
- [tissue::CellPinchingParams](#) [pinchingParam](#)
Parameters for cell pinching.
- double [ratio](#)
Ratio of the area from the daughter cell opposite from the normal and the area of the mother cell.

17.9.1 Detailed Description

Parameters for cell-system division.

Definition at line 46 of file cellsystem.h.

17.9.2 Constructor & Destructor Documentation

17.9.2.1 `cell_system::CellSystemDivisionParams::CellSystemDivisionParams (const Point3d & dir, double a = 0, double r = 0.5, double eps = 0.000001, bool pinch = false, tissue::CellPinchingParams pp = tissue::CellPinchingParams()) [inline]`

Constructor.

Only the direction has to be specified.

Definition at line 51 of file cellsystem.h.

```

53         : angle(a)
54         , ratio(r)
55         , direction(dir)
56         , epsilon(eps)
57         , minCellWall(0)
58         , pinching(pinch)
59         , pinchingParam(pp)
60     { }
```

17.9.2.2 `cell_system::CellSystemDivisionParams::CellSystemDivisionParams (const CellSystemDivisionParams & copy) [inline]`

Copy constructor.

Definition at line 65 of file cellsystem.h.

```

66         : angle(copy.angle)
67         , ratio(copy.ratio)
68         , direction(copy.direction)
69         , epsilon(copy.epsilon)
70         , minCellWall(0)
71         , pinching(copy.pinching)
72         , pinchingParam(copy.pinchingParam)
73     { }
```

17.9.3 Member Data Documentation

17.9.3.1 `double cell_system::CellSystemDivisionParams::angle`

Angle between the direction and the normal to the wall.

Definition at line 78 of file cellsystem.h.

Referenced by `cell_system::findDivisionPoints()`.

17.9.3.2 Point3d cell_system::CellSystemDivisionParams::direction

Reference direction to find the normal of the new wall.

Definition at line 91 of file cellsystem.h.

Referenced by cell_system::findDivisionPoints().

17.9.3.3 double cell_system::CellSystemDivisionParams::epsilon

Relative epsilon for floating point comparison.

Definition at line 96 of file cellsystem.h.

Referenced by cell_system::findDivisionPoints().

17.9.3.4 double cell_system::CellSystemDivisionParams::minCellWall

Minimum size of a cell wall.

Definition at line 101 of file cellsystem.h.

Referenced by cell_system::findDivisionPoints().

17.9.3.5 bool cell_system::CellSystemDivisionParams::pinching

Ask for pinching after the cell wall was found.

Definition at line 106 of file cellsystem.h.

Referenced by cell_system::findDivisionPoints().

17.9.3.6 tissue::CellPinchingParams cell_system::CellSystemDivisionParams::pinchingParam

Parameters for cell pinching.

Definition at line 111 of file cellsystem.h.

Referenced by cell_system::findDivisionPoints().

17.9.3.7 double cell_system::CellSystemDivisionParams::ratio

Ratio of the area from the daughter cell opposite from the normal and the area of the mother cell.

If ratio is 0, then just keep the new wall going through the center.

Definition at line 86 of file cellsystem.h.

Referenced by cell_system::findDivisionPoints().

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

- [vvelib/algorithms/cellsystem.h](#)

17.10 cell_system::CellSystemJunction Class Reference

Content of a vertex for the 2D cell system.

```
#include <algorithms/cellsystem.h>
```

Public Member Functions

- bool [serialize](#) (storage::VVEStorage &)
Serialization method.

Public Attributes

- Point3d [force](#)
Force of the junction.
- Point3d [pos](#)
Position of the vertex.
- Point3d [velocity](#)
Velocity of the junction.

17.10.1 Detailed Description

Content of a vertex for the 2D cell system.

Definition at line 337 of file cellsystem.h.

17.10.2 Member Function Documentation

17.10.2.1 bool cell_system::CellSystemJunction::serialize (storage::VVEStorage & store)

Serialization method.

Definition at line 25 of file cellsystem.cpp.

References storage::VVEStorage::field(), force, pos, and velocity.

```
26 {  
27     if(!store.field("Position", pos))  
28         return false;  
29     if(!store.field("Force", force))  
30     {  
31         cerr << "Warning, cell system junction without any force. Starting with zer
```

```
        o force." << endl;
32     force = Point3d();
33     }
34     if(!store.field("Velocity", velocity))
35     {
36         cerr << "Warning, cell system junction without any velocity. Starting with
zero velocity." << endl;
37         velocity = Point3d();
38     }
39     return true;
40 }
```

17.10.3 Member Data Documentation

17.10.3.1 Point3d cell_system::CellSystemJunction::force

Force of the junction.

Definition at line 346 of file cellsystem.h.

Referenced by `serialize()`.

17.10.3.2 Point3d cell_system::CellSystemJunction::pos

Position of the vertex.

Definition at line 342 of file cellsystem.h.

Referenced by `serialize()`.

17.10.3.3 Point3d cell_system::CellSystemJunction::velocity

Velocity of the junction.

Definition at line 350 of file cellsystem.h.

Referenced by `serialize()`.

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

- `vvelib/algorithms/cellsystem.h`
- `vvelib/algorithms/cellsystem.cpp`

17.11 util::CircIterator< ForwardIterator > Class Template Reference

Creates a circular iterator from a range of forward iterators.

```
#include <util/circ_iterator.h>
```

Public Types

- typedef ForwardIterator **base_type**
- typedef std::iterator_traits< ForwardIterator >::difference_type **difference_type**
- typedef std::forward_iterator_tag **iterator_category**
- typedef std::iterator_traits< ForwardIterator >::pointer **pointer**
- typedef std::iterator_traits< ForwardIterator >::reference **reference**
- typedef std::iterator_traits< ForwardIterator >::value_type **value_type**

Public Member Functions

- base_type **base** () const
- **CircIterator** (const [CircIterator](#) ©)
- [CircIterator](#) (const [CircIterator](#) ©, const ForwardIterator &new_cur)
Copy the iterator, but change the current position to somewhere else.
- [CircIterator](#) (const ForwardIterator &f, const ForwardIterator &l)
Creates an 'end' iterator.
- [CircIterator](#) (const ForwardIterator &f, const ForwardIterator &l, const ForwardIterator &c)
Create an iterator within a range, starting at c.
- [CircIterator](#) **end** () const
Create the end iterator from a current one.
- bool [isInitIterator](#) (const base_type &cmp) const
Check if the base iterator given as argument correspond to the position this circular iterator starts from.
- bool **operator!=** (const [CircIterator](#) &other) const
- bool **operator!=** (const ForwardIterator &other) const
- reference **operator*** ()
- [CircIterator](#) **operator++** (int)
- [CircIterator](#) & **operator++** ()
- pointer **operator->** ()
- bool **operator==** (const [CircIterator](#) &other) const
- bool **operator==** (const ForwardIterator &other) const

Protected Attributes

- ForwardIterator **cur**
- ForwardIterator **first**
- ForwardIterator **init**
- ForwardIterator **last**

17.11.1 Detailed Description

template<typename ForwardIterator> class util::CircIterator< ForwardIterator >

Creates a circular iterator from a range of forward iterators.

Definition at line 15 of file circ_iterator.h.

17.11.2 Constructor & Destructor Documentation

17.11.2.1 template<typename ForwardIterator > util::CircIterator< ForwardIterator >::CircIterator (const ForwardIterator &f, const ForwardIterator &l, const ForwardIterator &c) [inline]

Create an iterator within a range, starting at c.

Definition at line 30 of file circ_iterator.h.

```

31      : first(f)
32      , last(l)
33      , init(c)
34      , cur(c)
35      {
36      }
```

17.11.2.2 template<typename ForwardIterator > util::CircIterator< ForwardIterator >::CircIterator (const ForwardIterator &f, const ForwardIterator &l) [inline]

Creates an 'end' iterator.

Definition at line 41 of file circ_iterator.h.

```

42      : first(f)
43      , last(l)
44      , init(l)
45      , cur(l)
46      {
47      }
```

17.11.2.3 `template<typename ForwardIterator > util::CircIterator< ForwardIterator >::CircIterator (const CircIterator< ForwardIterator > & copy, const ForwardIterator & new_cur) [inline]`

Copy the iterator, but change the current position to somewhere else.

Definition at line 52 of file circ_iterator.h.

```
53      : first(copy.first)
54      , last(copy.last)
55      , init(copy.init)
56      , cur(new_cur)
57      {}
```

17.11.3 Member Function Documentation

17.11.3.1 `template<typename ForwardIterator > CircIterator util::CircIterator< ForwardIterator >::end () const [inline]`

Create the end iterator from a current one.

Definition at line 122 of file circ_iterator.h.

Referenced by `graph::VVGraph< VertexContent, EdgeContent, compact >::eraseEdge()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge()`.

```
123    {
124        return CircIterator(first, last);
125    }
```

17.11.3.2 `template<typename ForwardIterator > bool util::CircIterator< ForwardIterator >::isInitIterator (const base_type & cmp) const [inline]`

Check if the base iterator given as argument correspond to the position this circular iterator starts from.

Definition at line 114 of file circ_iterator.h.

Referenced by `graph::VVGraph< VertexContent, EdgeContent, compact >::eraseEdge()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge()`.

```
115    {
116        return cmp == init;
117    }
```

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

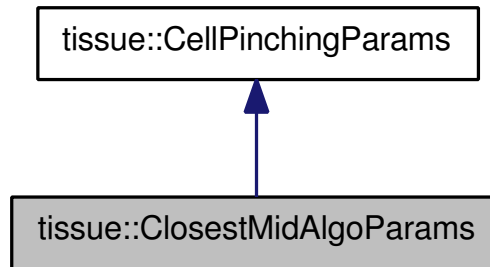
- vvelib/util/circ_iterator.h

17.12 tissue::ClosestMidAlgoParams Struct Reference

Parameters for the closest mid.

```
#include <tissue.h>
```

Inheritance diagram for tissue::ClosestMidAlgoParams:



Public Member Functions

- [ClosestMidAlgoParams](#) (const [ClosestMidAlgoParams](#) ©)

Copy constructor.

- [ClosestMidAlgoParams](#) (double cwm=0, bool scwm=true)

Default and initialization constructor.

Public Attributes

- double [cellWallMin](#)

Minimum size of a wall.

- bool [strictCellWallMin](#)

If true, the minimum size of a wall is strictly enforce, otherwise, it will not imply a change of division wall.

17.12.1 Detailed Description

Parameters for the closest mid.

Definition at line 226 of file tissue.h.

17.12.2 Constructor & Destructor Documentation

17.12.2.1 tissue::ClosestMidAlgoParams::ClosestMidAlgoParams (double *cwm* = 0, bool *scwm* = true) [inline]

Default and initialization constructor.

Definition at line 231 of file tissue.h.

```
232         : CellPinchingParams ()
233         , cellWallMin (cwm)
234         , strictCellWallMin (scwm)
235         { }
```

17.12.2.2 tissue::ClosestMidAlgoParams::ClosestMidAlgoParams (const ClosestMidAlgoParams & *copy*) [inline]

Copy constructor.

Definition at line 240 of file tissue.h.

```
241         : CellPinchingParams (copy)
242         , cellWallMin (copy.cellWallMin)
243         , strictCellWallMin (copy.strictCellWallMin)
244         { }
```

17.12.3 Member Data Documentation

17.12.3.1 double tissue::ClosestMidAlgoParams::cellWallMin

Minimum size of a wall.

Definition at line 249 of file tissue.h.

Referenced by tissue::findDivisionPoints().

17.12.3.2 bool tissue::ClosestMidAlgoParams::strictCellWallMin

If true, the minimum size of a wall is strictly enforce, otherwise, it will not imply a change of division wall.

Definition at line 254 of file tissue.h.

Referenced by tissue::findDivisionPoints().

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

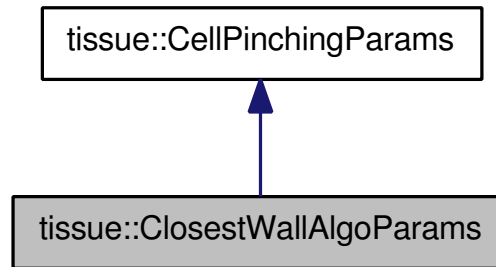
- vvelib/algorithms/[tissue.h](#)

17.13 tissue::ClosestWallAlgoParams Struct Reference

Parameters for the closest wall algorithm.

```
#include <tissue.h>
```

Inheritance diagram for tissue::ClosestWallAlgoParams:



Public Member Functions

- [ClosestWallAlgoParams](#) (const [ClosestWallAlgoParams](#) ©)

Copy constructor.

- [ClosestWallAlgoParams](#) (double cwm=0, bool scwm=true)

Default and initialization constructor.

Public Attributes

- double [cellWallMin](#)

Minimum size of a wall.

- bool [strictCellWallMin](#)

If true, the minimum size of a wall is strictly enforce, otherwise, it will not imply a change of division wall.

17.13.1 Detailed Description

Parameters for the closest wall algorithm.

Definition at line 260 of file `tissue.h`.

17.13.2 Constructor & Destructor Documentation

17.13.2.1 tissue::ClosestWallAlgoParams::ClosestWallAlgoParams (double *cwm* = 0, bool *scwm* = true) [inline]

Default and initialization constructor.

Definition at line 265 of file tissue.h.

```
266         : CellPinchingParams ()
267         , cellWallMin (cwm)
268         , strictCellWallMin (scwm)
269         { }
```

17.13.2.2 tissue::ClosestWallAlgoParams::ClosestWallAlgoParams (const ClosestWallAlgoParams & *copy*) [inline]

Copy constructor.

Definition at line 274 of file tissue.h.

```
275         : CellPinchingParams (copy)
276         , cellWallMin (copy.cellWallMin)
277         , strictCellWallMin (copy.strictCellWallMin)
278         { }
```

17.13.3 Member Data Documentation

17.13.3.1 double tissue::ClosestWallAlgoParams::cellWallMin

Minimum size of a wall.

Definition at line 283 of file tissue.h.

Referenced by tissue::findDivisionPoints().

17.13.3.2 bool tissue::ClosestWallAlgoParams::strictCellWallMin

If true, the minimum size of a wall is strictly enforce, otherwise, it will not imply a change of division wall.

Definition at line 288 of file tissue.h.

Referenced by tissue::findDivisionPoints().

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

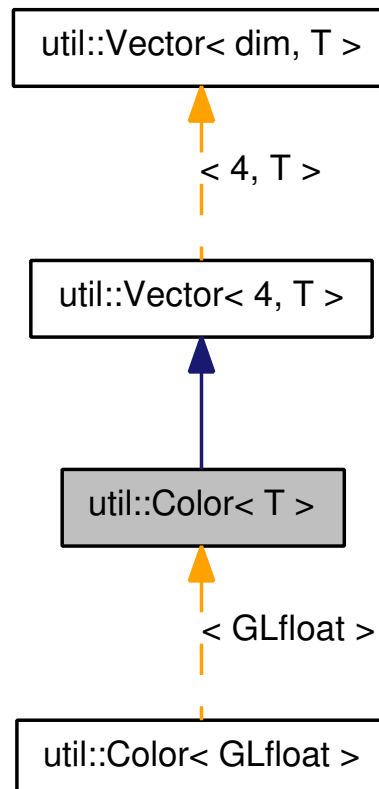
- vvelib/algorithms/[tissue.h](#)

17.14 util::Color< T > Class Template Reference

A utility class to encapsulate color data.

```
#include <util/color.h>
```

Inheritance diagram for util::Color< T >:



Public Member Functions

- void `a` (const T &val)
Set the alpha component.
- const T & `a` () const
Return the alpha component.
- T & `a` ()
Return the alpha component.
- void `b` (const T &val)
Set the blue component.

- `const T & b () const`
Return the blue component.
- `T & b ()`
Return the blue component.
- `Color (const T &r=T(), const T &g=T(), const T &b=T(), const T &a=T())`
Constructor.
- `Color (const QColor &c)`
- `Color (const Vector< 4, T > ©)`
- `template<typename T1 >
Color (const Vector< 4, T1 > &color, const T1 &scale)`
- `template<typename T1 >
Color (const Vector< 4, T1 > &color, const T &scale=1)`
Constructor to convert from one color type to another.
- `void g (const T &val)`
Set the green component.
- `const T & g () const`
Return the green component.
- `T & g ()`
Return the green component.
- `operator QColor () const`
- `Color & operator= (const T &val)`
Set all the elements to value.
- `Color & operator= (const Vector< 4, T > &c)`
- `Color & operator= (const Color &c)`
Assignment of color data.
- `void r (const T &val)`
Set the red component.
- `const T & r () const`
Return the red component.
- `T & r ()`
Return the red component.

17.14.1 Detailed Description

template<class T> class util::Color< T >

A utility class to encapsulate color data.

Definition at line 19 of file color.h.

17.14.2 Constructor & Destructor Documentation

17.14.2.1 template<class T> template<typename T1 > util::Color< T >::Color (const Vector< 4, T1 > & color, const T & scale = 1) [inline, explicit]

Constructor to convert from one color type to another.

Definition at line 26 of file color.h.

```
27         : Vector<4,T>(color[0]*scale, color[1]*scale, color[2]*scale, color[3]*scale)
           e)
28     { }
```

17.14.2.2 template<class T> util::Color< T >::Color (const T & r = T(), const T & g = T(), const T & b = T(), const T & a = T()) [inline]

Constructor.

Parameters

- r* Value for red.
- g* Value for green.
- b* Value for blue.
- a* Value for alpha.

Definition at line 55 of file color.h.

```
56         : Vector<4,T>(r,g,b,a) { }
```

17.14.3 Member Function Documentation

17.14.3.1 template<class T> void util::Color< T >::a (const T & val) [inline]

Set the alpha component.

Definition at line 118 of file color.h.

```
118 {this->t(val);}
```

**17.14.3.2 template<class T> const T& util::Color< T >::a () const
[inline]**

Return the alpha component.

Definition at line 97 of file color.h.

```
97 {return this->t();}
```

17.14.3.3 template<class T> T& util::Color< T >::a () [inline]

Return the alpha component.

Definition at line 76 of file color.h.

Referenced by util::Palette::blend(), util::convertHSVtoRGB(), and util::Palette::getColor().

```
76 {return this->t();}
```

**17.14.3.4 template<class T> void util::Color< T >::b (const T & val)
[inline]**

Set the blue component.

Definition at line 113 of file color.h.

```
113 {this->z(val);}
```

**17.14.3.5 template<class T> const T& util::Color< T >::b () const
[inline]**

Return the blue component.

Definition at line 92 of file color.h.

```
92 {return this->z();}
```

17.14.3.6 template<class T> T& util::Color< T >::b () [inline]

Return the blue component.

Definition at line 71 of file color.h.

Referenced by util::Palette::blend(), util::convertHSVtoRGB(), util::Palette::getColor(), tissue_model::TissueModel< RealModel, TissueClass >::preDraw(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::preDraw(), and util::Palette::reread().

```
71 {return this->z();}
```

17.14.3.7 `template<class T> void util::Color< T >::g (const T & val)` **[inline]**

Set the green component.

Definition at line 108 of file color.h.

```
108 {this->y(val);}
```

17.14.3.8 `template<class T> const T& util::Color< T >::g () const` **[inline]**

Return the green component.

Definition at line 87 of file color.h.

```
87 {return this->y();}
```

17.14.3.9 `template<class T> T& util::Color< T >::g ()` **[inline]**

Return the green component.

Definition at line 66 of file color.h.

Referenced by `util::Palette::blend()`, `util::convertHSVtoRGB()`, `util::Palette::getColor()`, `tissue_model::TissueModel< RealModel, TissueClass >::preDraw()`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::preDraw()`, and `util::Palette::reread()`.

```
66 {return this->y();}
```

17.14.3.10 `template<class T> Color< T > & util::Color< T >::operator=` **(const T & value) [inline]**

Set all the elements to `value`.

Reimplemented from `util::Vector< 4, T >`.

Definition at line 173 of file color.h.

References `util::Color< T >::operator=()`.

```
173                                     {
174     this->Vector<4,T>::operator=(val);
175     return *this;
176 }
```

17.14.3.11 template<class T> Color< T> & util::Color< T>::operator=(const Color< T> & c) [inline]

Assignment of color data.

Definition at line 167 of file color.h.

Referenced by util::Color< T>::operator=().

```

167                                     {
168         this->Vector<4,T>::operator=(c);
169         return *this;
170     }
```

17.14.3.12 template<class T> void util::Color< T>::r (const T & val) [inline]

Set the red component.

Definition at line 103 of file color.h.

```

103 {this->x(val);}
```

17.14.3.13 template<class T> const T& util::Color< T>::r () const [inline]

Return the red component.

Definition at line 82 of file color.h.

```

82 {return this->x();}
```

17.14.3.14 template<class T> T& util::Color< T>::r () [inline]

Return the red component.

Definition at line 61 of file color.h.

Referenced by util::Palette::blend(), util::convertHSVtoRGB(), util::Palette::getColor(), tissue_model::TissueModel< RealModel, TissueClass>::preDraw(), bspline_tissue_model::TissueModel< RealModel, TissueClass>::preDraw(), and util::Palette::reread().

```

61 {return this->x();}
```

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

- vvelib/util/[color.h](#)

17.15 bspline_tissue_model::TissueModel< RealModel, TissueClass >::CompareSize Struct Reference

Operator class to compare the size of cells.

```
#include <bspline_tissue_model.h>
```

Public Member Functions

- **bool operator()** (const cell &c1, const cell &c2) const

17.15.1 Detailed Description

template<typename RealModel, typename TissueClass> struct bspline_tissue_model::TissueModel< RealModel, TissueClass >::CompareSize

Operator class to compare the size of cells.

Definition at line 223 of file bspline_tissue_model.h.

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

- [vvelib/bspline_tissue_model.h](#)

17.16 tissue_model::TissueModel< RealModel, TissueClass >::CompareSize Struct Reference

Operator class to compare the size of cells.

```
#include <tissue_model.h>
```

Public Member Functions

- **bool operator()** (const cell &c1, const cell &c2) const

17.16.1 Detailed Description

```
template<typename RealModel, typename TissueClass> struct tissue_ -  
model::TissueModel< RealModel, TissueClass >::CompareSize
```

Operator class to compare the size of cells.

Definition at line 218 of file tissue_model.h.

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

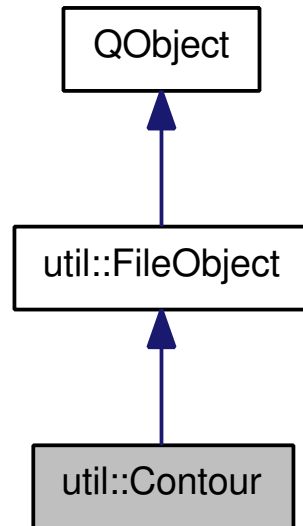
- vvelib/[tissue_model.h](#)

17.17 util::Contour Class Reference

[Contour](#) utility class.

```
#include <util/contour.h>
```

Inheritance diagram for util::Contour:



Public Member Functions

- [Contour](#) (std::string filename)
Constructor from file.
- **Contour** (const [Contour](#) &)
- [Contour](#) ()
Default constructor.
- const [util::Vector](#)< 3, double > & [getMax](#) () const
Return the maximum x, y and z coordinates.
- const [util::Vector](#)< 3, double > & [getMin](#) () const
Return the minimum x, y and z coordinates.
- double [length](#) (double a, double b, double dt=0.01)
Return the arc length for a parameter interval.
- [util::Vector](#)< 3, double > [normal](#) (double t, double dt=0.01)
Finds the normal vector at a given parameter value.

- `util::Vector< 3, double > operator()` (double t) const
Function operator.
- `Contour & operator=` (const `Contour` &)
Copy constructor.
- `void reread ()`
Method to be implemented in subclasses to reread the file.
- `util::Vector< 3, double > tangent` (double t, double dt=0.01)
Finds the tangent vector at a given parameter value.
- `double travel` (double t, double l, double dt=0.01)
Return the paramter after traveling a given arc length.

17.17.1 Detailed Description

`Contour` utility class. The `Contour` class encapsulates b-spline contours specified in the VLAB contour formats. Currently all versions are supported (original, ver. 1.1 and 1.4). Instances of `Contour` behave as function objects.

Definition at line 25 of file `contour.h`.

17.17.2 Constructor & Destructor Documentation

17.17.2.1 util::Contour::Contour ()

Default constructor.

Definition at line 25 of file `contour.cpp`.

```

26      : FileObject()
27      , closed(true)
28      , regular(true)
29      {}

```

17.17.2.2 util::Contour::Contour (std::string filename)

Constructor from file.

Bug

If the specified contains an incomplete or improper specification, the result of construction will be unknown. Most probably, the object will unuseable; however, this is not currently reported.

Parameters

filename The contour file to load.

Definition at line 38 of file contour.cpp.

References reread().

```

39      : FileObject(filename)
40      , closed(true)
41      , regular(true)
42      {
43      reread();
44      }
```

17.17.3 Member Function Documentation**17.17.3.1 const Vector< 3, double > & util::Contour::getMax () const**

Return the maximum x, y and z coordinates.

Definition at line 447 of file contour.cpp.

```

447                                     {
448      return max;
449      }
```

17.17.3.2 const Vector< 3, double > & util::Contour::getMin () const

Return the minimum x, y and z coordinates.

Definition at line 452 of file contour.cpp.

```

452                                     {
453      return min;
454      }
```

17.17.3.3 double util::Contour::length (double *a*, double *b*, double *dt* = 0.01)

Return the arc length for a parameter interval.

Parameters

a Parameter for the start of the interval.

b Parameter for the end of the interval.

dt The step used for forward differencing.

It is expected that *a* and *b* are both in the interval of [0, 1] and that *dt* is less than *b* - *a*.

Definition at line 464 of file contour.cpp.

Referenced by travel().

```

464                                     {
465     if (a < 0.0) a = 0.0;
466     if (a > 1.0) a = 1.0;
467     if (b > 1.0) b = 1.0;
468     if (b < a + dt) return 0.0;
469
470     double l = 0.0;
471     Vector<3,double> p = (*this)(a), q;
472     while (a <= b) {
473         a += dt;
474         q = (*this)(a);
475         l += norm(p-q);
476         p = q;
477     }
478     return l;
479 }
```

17.17.3.4 Vector< 3, double > util::Contour::normal (double *t*, double *dt* = 0.01)

Finds the normal vector at a given parameter value.

Parameters

- t* Parameter value for where to take the normal.
- dt* The interval size used for the numerical evaluation.

Definition at line 519 of file contour.cpp.

References `tangent()`, `util::Vector< dim, T >::x()`, and `util::Vector< dim, T >::y()`.

```

519                                     {
520     Vector<3,double> tvec = tangent(t, dt);
521     return Vector<3,double>(tvec.y(), tvec.x(), 0);
522 }
```

17.17.3.5 Vector< 3, double > util::Contour::operator() (double *t*) const

[Function](#) operator.

Parameters

- t* Time along the curve, between zero and one.

This calculates the 3-dimensional coordinates of the curve at time *t*, where *t* is between zero and one. If *t* is less then zero, it is set to zero. If it is larger than one, it is set to one.

Definition at line 431 of file contour.cpp.

```

431                                     {
432     if (t < 0.0) t = 0.0;
```

```

433     else if (t > 1.0) t = 1.0;
434
435     double k = 1.0 + double(pts.size() - 3) * t;
436     int i = int(k) + 2;
437     double p = k - floor(k);
438
439     return
440         pts[i - 3] * Basis0(p)
441         + pts[i - 2] * Basis1(p)
442         + pts[i - 1] * Basis2(p)
443         + pts[i]      * Basis3(p);
444 }

```

17.17.3.6 Contour & util::Contour::operator= (const Contour & c)

Copy constructor.

Definition at line 415 of file contour.cpp.

```

415                                     {
416     pts = c.pts;
417     max = c.max;
418     min = c.min;
419     closed = c.closed;
420
421     return *this;
422 }

```

17.17.3.7 void util::Contour::reread () [virtual]

Method to be implemented in subclasses to reread the file.

Implements [util::FileObject](#).

Definition at line 56 of file contour.cpp.

References [QFile::close\(\)](#), [QString::fromStdString\(\)](#), [QFile::handle\(\)](#), [QString::mid\(\)](#), [QFile::open\(\)](#), [QString::size\(\)](#), [QString::split\(\)](#), [QString::startsWith\(\)](#), and [QString::trimmed\(\)](#).

Referenced by [Contour\(\)](#).

```

56                                     {
57     closed = false;
58     QString fn = QString::fromStdString(filename);
59     QFile f(fn);
60     if(!f.open(QIODevice::ReadOnly))
61     {
62         err << "Error opening file " << QString::fromStdString(filename) << ": " <<
        f.errorString() << endl;
63         return;
64     }
65     #if defined(__APPLE__) || defined(linux)
66         flock(f.handle(), LOCK_SH);
67     #endif
68     QTextStream ts(&f);

```

```
69     int ptid = 0, line_nb = 1;
70
71     pts.clear();
72
73     QString line = ts.readLine().trimmed();
74     QStringList ver = line.split(" ");
75     if(ver.size() != 3 or ver[0] != "cver")
76     {
77         err << "The file '" << fn << "' doesn't contain a valid contour" << endl;
78         return;
79     }
80
81     bool ok;
82     int vmaj = ver[1].toInt(&ok);
83     if(!ok)
84     {
85         REPORT_ERROR(fn, line_nb, "The major version number is not a valid integer"
86     );
87         return;
88     }
89     int vmin = ver[2].toInt(&ok);
90     if(!ok)
91     {
92         REPORT_ERROR(fn, line_nb, "The minor version number is not a valid integer"
93     );
94         return;
95     }
96     int version = 100*vmaj+vmin;
97     if(version == 101)
98     {
99         while(!ts.atEnd())
100         {
101             ++line_nb;
102             QString line = ts.readLine().trimmed();
103             if(line.startsWith("name:"))
104             {
105                 // Do nothing of the name
106             }
107             else if(line.startsWith("points:"))
108             {
109                 QStringList nbpts = line.mid(7).trimmed().split(" ");
110                 if(nbpts.size() != 2)
111                 {
112                     REPORT_ERROR(fn, line_nb, "Error, there should be two values for the
points number, not " << nbpts.size());
113                     return;
114                 }
115                 bool ok;
116                 int nb_pts = nbpts[1].toInt(&ok);
117                 if(!ok)
118                 {
119                     REPORT_ERROR(fn, line_nb, "Error, number of control points is not a n
umber");
120                     return;
121                 }
122                 pts.resize(nb_pts);
123             }
124             else if(line.startsWith("type:"))
125             {
126                 QString type = line.mid(5).trimmed().toLower();
```

```

127         if(type == "closed")
128             closed = true;
129         else if(type == "open")
130             closed = false;
131         else
132         {
133             REPORT_ERROR(fn, line_nb, "Unknown contour type: " << type);
134             return;
135         }
136     }
137     else
138     {
139         if(ptid >= (int)pts.size())
140         {
141             REPORT_ERROR(fn, line_nb, "There are more control points defined than
declared");
142             return;
143         }
144         QStringList pt = line.split(" ");
145         if(pt.size() != 4)
146         {
147             REPORT_ERROR(fn, line_nb, "A points should be described with four val
ues");
148             return;
149         }
150         bool ok;
151         double x,y,z,m;
152         x = pt[0].toDouble(&ok);
153         if(!ok)
154         {
155             REPORT_ERROR(fn, line_nb, "Error, x coordinate is not a valid floatin
g point value");
156             return;
157         }
158         y = pt[1].toDouble(&ok);
159         if(!ok)
160         {
161             REPORT_ERROR(fn, line_nb, "Error, y coordinate is not a valid floatin
g point value");
162             return;
163         }
164         z = pt[2].toDouble(&ok);
165         if(!ok)
166         {
167             REPORT_ERROR(fn, line_nb, "Error, z coordinate is not a valid floatin
g point value");
168             return;
169         }
170         m = pt[3].toDouble(&ok);
171         if(!ok)
172         {
173             REPORT_ERROR(fn, line_nb, "Error, m coordinate is not a valid floatin
g point value");
174             return;
175         }
176         Point3d v(x,y,z);
177         pts[ptid++] = v;
178     }
179 }
180 }
181 else if(version == 102)
182 {

```

```
183         while(!ts.atEnd())
184         {
185             ++line_nb;
186             QString line = ts.readLine().trimmed();
187             if(line.startsWith("name:"))
188             {
189                 // Do nothing of the name
190             }
191             else if(line.startsWith("points:"))
192             {
193                 QStringList nbpts = line.mid(7).trimmed().split(" ");
194                 if(nbpts.size() != 2)
195                 {
196                     REPORT_ERROR(fn, line_nb, "Error, there should be two values for the
points number, not " << nbpts.size());
197                     return;
198                 }
199                 bool ok;
200                 int nb_pts = nbpts[1].toInt(&ok);
201                 if(!ok)
202                 {
203                     REPORT_ERROR(fn, line_nb, "Error, number of control points is not a n
umber");
204                     return;
205                 }
206                 pts.resize(nb_pts);
207             }
208             else if(line.startsWith("type:"))
209             {
210                 QString type = line.mid(5).trimmed().toLower();
211                 if(type.size() != 2)
212                 {
213                     REPORT_ERROR(fn, line_nb, "Error reading contour type: '" << type <<
"'");
214                     return;
215                 }
216                 if(type[0] == 'c')
217                     closed = true;
218                 else if(type[0] == 'o')
219                     closed = false;
220                 else
221                 {
222                     REPORT_ERROR(fn, line_nb, "Unknown contour type: " << type);
223                     return;
224                 }
225                 if(type[1] == 'e')
226                 {
227                     regular = false;
228                     err << "Warning, this contour reader doesn't handle contours that are
not regular" << endl;
229                 }
230                 else if(type[1] == 'r')
231                 {
232                     regular = true;
233                 }
234                 else
235                 {
236                     REPORT_ERROR(fn, line_nb, "Unknown contour type: " << type);
237                     return;
238                 }
239             }
240             else
```

```

241     {
242         if(ptid >= (int)pts.size())
243         {
244             REPORT_ERROR(fn, line_nb, "There are more control points defined than
declared");
245             return;
246         }
247         QStringList pt = line.split(" ");
248         if(pt.size() != 4)
249         {
250             REPORT_ERROR(fn, line_nb, "A points should be described with four val
ues");
251             return;
252         }
253         bool ok;
254         double x,y,z,m;
255         x = pt[0].toDouble(&ok);
256         if(!ok)
257         {
258             REPORT_ERROR(fn, line_nb, "Error, x coordinate is not a valid floatin
g point value");
259             return;
260         }
261         y = pt[1].toDouble(&ok);
262         if(!ok)
263         {
264             REPORT_ERROR(fn, line_nb, "Error, y coordinate is not a valid floatin
g point value");
265             return;
266         }
267         z = pt[2].toDouble(&ok);
268         if(!ok)
269         {
270             REPORT_ERROR(fn, line_nb, "Error, z coordinate is not a valid floatin
g point value");
271             return;
272         }
273         m = pt[3].toDouble(&ok);
274         if(!ok)
275         {
276             REPORT_ERROR(fn, line_nb, "Error, m coordinate is not a valid floatin
g point value");
277             return;
278         }
279         Point3d v(x,y,z);
280         pts[ptid++] = v;
281     }
282 }
283 }
284 else if(version == 103)
285 {
286     while(!ts.atEnd())
287     {
288         ++line_nb;
289         QString line = ts.readLine().trimmed();
290         if(line.startsWith("name:"))
291         {
292             // Do nothing of the name
293         }
294         else if(line.startsWith("points:"))
295         {
296             QStringList nbpts = line.mid(7).trimmed().split(" ");

```



```
297         if(nbpts.size() != 2)
298         {
299             REPORT_ERROR(fn, line_nb, "Error, there should be two values for the
points number, not " << nbpts.size());
300             return;
301         }
302         bool ok;
303         int nb_pts = nbpts[1].toInt(&ok);
304         if(!ok)
305         {
306             REPORT_ERROR(fn, line_nb, "Error, number of control points is not a n
umber");
307             return;
308         }
309         pts.resize(nb_pts);
310     }
311     else if(line.startsWith("type:"))
312     {
313         QString type = line.mid(5).trimmed().toLower();
314         if(type.size() != 2)
315         {
316             REPORT_ERROR(fn, line_nb, "Error reading contour type: '" << type <<
"'");
317             return;
318         }
319         if(type[0] == 'c')
320             closed = true;
321         else if(type[0] == 'o')
322             closed = false;
323         else
324         {
325             REPORT_ERROR(fn, line_nb, "Unknown contour type: " << type);
326             return;
327         }
328         if(type[1] == 'e')
329         {
330             regular = false;
331             err << "Warning, this contour reader doesn't handle contours that are
not regular" << endl;
332         }
333         else if(type[1] == 'r')
334         {
335             regular = true;
336         }
337         else
338         {
339             REPORT_ERROR(fn, line_nb, "Unknown contour type: " << type);
340             return;
341         }
342     }
343     else if(line.startsWith("samples:"))
344     {
345         // ignore that
346     }
347     else
348     {
349         if(ptid >= (int)pts.size())
350         {
351             REPORT_ERROR(fn, line_nb, "There are more control points defined than
declared");
352             return;
353         }
354     }
```

```

354         QStringList pt = line.split(" ");
355         if(pt.size() != 4)
356         {
357             REPORT_ERROR(fn, line_nb, "A points should be described with four val
ues");
358             return;
359         }
360         bool ok;
361         double x,y,z,m;
362         x = pt[0].toDouble(&ok);
363         if(!ok)
364         {
365             REPORT_ERROR(fn, line_nb, "Error, x coordinate is not a valid floatin
g point value");
366             return;
367         }
368         y = pt[1].toDouble(&ok);
369         if(!ok)
370         {
371             REPORT_ERROR(fn, line_nb, "Error, y coordinate is not a valid floatin
g point value");
372             return;
373         }
374         z = pt[2].toDouble(&ok);
375         if(!ok)
376         {
377             REPORT_ERROR(fn, line_nb, "Error, z coordinate is not a valid floatin
g point value");
378             return;
379         }
380         m = pt[3].toDouble(&ok);
381         if(!ok)
382         {
383             REPORT_ERROR(fn, line_nb, "Error, m coordinate is not a valid floatin
g point value");
384             return;
385         }
386         Point3d v(x,y,z);
387         pts[ptid++] = v;
388     }
389 }
390 }
391 else
392 {
393     REPORT_ERROR(fn, line_nb, "Cannot handle version " << vmaj << " " << vmin)
;
394     return;
395 }
396
397 if(ptid < (int)pts.size())
398 {
399     REPORT_ERROR(fn, line_nb, "There are less control points defined than decla
red");
400     return;
401 }
402
403 if (closed) {
404     pts.push_back(pts[0]);
405     pts.push_back(pts[1]);
406     pts.push_back(pts[2]);
407 }
408 #if defined(__APPLE__) || defined(linux)

```

```

409     flock(f.handle(), LOCK_UN);
410 #endif
411     f.close();
412 }

```

17.17.3.8 Vector<3, double> util::Contour::tangent (double t , double $dt = 0.01$)

Finds the tangent vector at a given parameter value.

Parameters

- t Parameter value for where to take the tangent.
- dt The interval size used for the numerical evaluation.

Definition at line 509 of file contour.cpp.

References util::Vector<dim, T>::normalize().

Referenced by normal().

```

509                                     {
510     Vector<3,double> r = (*this)(t + dt) - (*this)(t - dt);
511     r.normalize();
512     return r;
513 }

```

17.17.3.9 double util::Contour::travel (double t , double l , double $dt = 0.01$)

Return the paramter after traveling a given arc length.

Parameters

- t Parameter for the start of the interval.
- l The arc length to travel.
- dt The step used for forward differencing.

It is expected that t is in the interval of $[0, 1]$ and that l is positive.

Definition at line 489 of file contour.cpp.

References length().

```

489                                     {
490     if (t < 0.0) t = 0.0;
491     if (l < 0.0) return 0.0;
492     if (l < dt) return t;
493
494     Vector<3,double> start = (*this)(t);
495     double t_length = 0.0;
496     double u = t + dt;

```

```
497     while (t_length < 1 && t < 1.0) {
498         t_length += this->length(t, u, dt);
499         t = u;
500         u += dt;
501     }
502     return t;
503 }
```

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

- vvelib/util/[contour.h](#)
- vvelib/util/contour.cpp

17.18 storage::ConvertType< From, To > Struct Template Reference

Convert object of type from to object of type to.

```
#include <types.h>
```

Public Member Functions

- bool [operator\(\)](#) (const From &from, To &to) const

17.18.1 Detailed Description

template<typename From, typename To> struct storage::ConvertType< From, To >

Convert object of type from to object of type to.

Definition at line 158 of file types.h.

17.18.2 Member Function Documentation

17.18.2.1 **template<typename From , typename To > bool storage::ConvertType< From, To >::operator() (const From &from, To &to) const** [[inline](#)]

Returns

True if the conversion was possible, false otherwise.

Definition at line 163 of file types.h.

```
164     {  
165         to = (To)from;  
166         return true;  
167     }
```

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

- [vvelib/storage/types.h](#)

17.19 graph::CountedContent< Content > Struct Template Reference

Type of the reference counted content.

```
#include <vertex.h>
```

Public Attributes

- unsigned int [count](#)
Reference count on the vertex.
- size_t [num](#)

17.19.1 Detailed Description

```
template<typename Content> struct graph::CountedContent< Content >
```

Type of the reference counted content.

Definition at line 76 of file vertex.h.

17.19.2 Member Data Documentation

17.19.2.1 template<typename Content > unsigned int graph::CountedContent< Content >::count

Reference count on the vertex.

Definition at line 89 of file vertex.h.

Referenced by graph::Vertex< VertexContent >::Vertex().

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

- vvelib/graph/[vertex.h](#)

17.20 util::CrossProductType< 2, T > Struct Template Reference

For 2d -> a scalar.

```
#include <vector.h>
```

Public Types

- typedef T type

17.20.1 Detailed Description

template<typename T> struct util::CrossProductType< 2, T >

For 2d -> a scalar.

Definition at line 1255 of file vector.h.

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

- vvelib/util/[vector.h](#)

17.21 `util::CrossProductType< 3, T >` Struct Template Reference

For 3d -> a vector.

```
#include <vector.h>
```

Public Types

- typedef [Vector](#)< 3, T > **type**

17.21.1 Detailed Description

template<typename T> struct `util::CrossProductType< 3, T >`

For 3d -> a vector.

Definition at line 1264 of file `vector.h`.

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

- `vvelib/util/`[vector.h](#)

17.22 util::GraphInset::Data Struct Reference

[Data](#) structure used to store the time stamped values.

```
#include <graph_inset.h>
```

Public Member Functions

- **Data** (const [Data](#) ©)
- **Data** (double t, const std::vector< double > &vs, bool c)

Public Attributes

- bool [changed](#)
This value comes from a different source than the previous one.
- double [time](#)
Time stamp.
- std::vector< double > [values](#)
Vector of values with one value per line.

17.22.1 Detailed Description

[Data](#) structure used to store the time stamped values.

Definition at line 36 of file graph_inset.h.

17.22.2 Member Data Documentation

17.22.2.1 bool util::GraphInset::Data::changed

This value comes from a different source than the previous one.

Definition at line 63 of file graph_inset.h.

17.22.2.2 double util::GraphInset::Data::time

Time stamp.

Definition at line 53 of file graph_inset.h.

17.22.2.3 `std::vector<double> util::GraphInset::Data::values`

[Vector](#) of values with one value per line.

Definition at line 58 of file `graph_inset.h`.

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

- `vvelib/util/`[graph_inset.h](#)

17.23 `vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::division_result_t` Class Reference 269

`vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::division_result_t` Class Reference

Describe the result of a cell division.

```
#include <complex.h>
```

Public Member Functions

- `division_result_t` (const `division_result_t` ©)
- `division_result_t` (`cell` cl, `cell` cr, `junction` u1, `junction` u2, `junction` u, `junction` v1, `junction` v2, `junction` v)
- `operator bool` ()

Public Attributes

- `cell` cl
- `cell` cr
- `junction` u
- `junction` u1
- `junction` u2
- `junction` v
- `junction` v1
- `junction` v2

17.23.1 Detailed Description

```
template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> class vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::division_result_t
```

Describe the result of a cell division. Can be converted to boolean to check the division happened, and return the two new cells.

Definition at line 1359 of file `complex.h`.

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

- `vvlib/algorithms/complex.h`

17.24 `vvcomplex::DivisionData< JunctionContent >` Class Template Reference

Class holding the data needed to actually divide a cell.

```
#include <algorithms/complex.h>
```

Public Types

- typedef `graph::Vertex< JunctionContent >` `junction`
Type of graph vertex.

Public Member Functions

- `DivisionData` (const `DivisionData` ©)
Copy constructor.
- `DivisionData` (Point3d _pu, Point3d _pv, const `junction` &_u1, const `junction` &_v1, bool _du1=false, bool _dv1=false)
Complete constructor.
- `DivisionData` ()
Default constructor do not require pinching.
- `operator bool` ()
Convert the result to true if everything is ok.
- bool `operator!` () const
Convert the result to false if everything is ok.

Public Attributes

- bool `divide_at_u1`
If true, pu is ignored and the division wall goes through u1.
- bool `divide_at_v1`
If true, pv is ignored and the division wall goes through v1.
- Point3d `pu`
Position of the junction inserted in the U wall.
- Point3d `pv`
Position of the junction inserted in the V wall.

- [junction u1](#)

First junction of the 'U' wall, u2 is the vertex after u1 in the neighborhood of c.

- [junction v1](#)

First junction of the 'V' wall, v2 is the vertex after v1 in the neighborhood of c.

17.24.1 Detailed Description

template<typename JunctionContent> class vvcomplex::DivisionData< JunctionContent >

Class holding the data needed to actually divide a cell. This is the return type of the different division algorithm. It provides all the informations needed to modify the graphs to account for the cell division.

The division will cut the walls (u1,u2) and (v1,v2) and will place the new vertexes at positions pu on the (u1,u2) wall and pv on the (v1,v2) wall.

Definition at line 746 of file complex.h.

17.24.2 Member Typedef Documentation

17.24.2.1 template<typename JunctionContent> typedef graph::Vertex<JunctionContent> vvcomplex::DivisionData< JunctionContent >::junction

Type of graph vertex.

Definition at line 751 of file complex.h.

17.24.3 Constructor & Destructor Documentation

17.24.3.1 template<typename JunctionContent> vvcomplex::DivisionData< JunctionContent >::DivisionData () [inline]

Default constructor do not require pinching.

Definition at line 756 of file complex.h.

```

757         : u1(0)
758         , v1(0)
759         , divide_at_u1(false)
760         , divide_at_v1(false)
761     { }
```

17.24.3.2 `template<typename JunctionContent> vvcomplex::DivisionData< JunctionContent >::DivisionData (Point3d _pu, Point3d _pv, const junction & _u1, const junction & _v1, bool _du1 = false, bool _dv1 = false) [inline]`

Complete constructor.

Definition at line 766 of file complex.h.

```

771      : pu(_pu)
772      , pv(_pv)
773      , u1(_u1)
774      , v1(_v1)
775      , divide_at_u1(_du1)
776      , divide_at_v1(_dv1)
777      { }
```

17.24.3.3 `template<typename JunctionContent> vvcomplex::DivisionData< JunctionContent >::DivisionData (const DivisionData< JunctionContent > & copy) [inline]`

Copy constructor.

Definition at line 782 of file complex.h.

```

783      : pu(copy.pu)
784      , pv(copy.pv)
785      , u1(copy.u1)
786      , v1(copy.v1)
787      , divide_at_u1(copy.divide_at_u1)
788      , divide_at_v1(copy.divide_at_v1)
789      { }
```

17.24.4 Member Function Documentation

17.24.4.1 `template<typename JunctionContent> vvcomplex::DivisionData< JunctionContent >::operator bool () [inline]`

Convert the result to true if everything is ok.

If the algorithm fails for any reason, the returned structure should have u1 or v1 set to null vertexes. That way, the boolean operator will convert it to false.

Definition at line 798 of file complex.h.

References `vvcomplex::DivisionData< JunctionContent >::u1`, and `vvcomplex::DivisionData< JunctionContent >::v1`.

```

799      {
800          return u1 and v1;
801      }
```

17.24 **vvcomplex::DivisionData< JunctionContent > Class Template Reference** 373

17.24.4.2 `template<typename JunctionContent> bool
vvcomplex::DivisionData< JunctionContent >::operator! () const
[inline]`

Convert the result to false if everything is ok.

Definition at line 806 of file complex.h.

References `vvcomplex::DivisionData< JunctionContent >::u1`, and
`vvcomplex::DivisionData< JunctionContent >::v1`.

```
807     {  
808         return !u1 or !v1;  
809     }
```

17.24.5 Member Data Documentation

17.24.5.1 `template<typename JunctionContent> vvcomplex::DivisionData<
JunctionContent >::divide_at_u1`

If true, pu is ignored and the division wall goes through u1.

Definition at line 839 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_-
COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass,
Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell(),
vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass,
Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_-
EmptyEdgeContent, graph::_EmptyEdgeContent, false >::divideCell(), and vvcom-
plex::testDivisionOnVertices()`.

17.24.5.2 `template<typename JunctionContent> vvcomplex::DivisionData<
JunctionContent >::divide_at_v1`

If true, pv is ignored and the division wall goes through v1.

Definition at line 839 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_-
COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass,
Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell(),
vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass,
Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_-
EmptyEdgeContent, graph::_EmptyEdgeContent, false >::divideCell(), and vvcom-
plex::testDivisionOnVertices()`.

17.24.5.3 `template<typename JunctionContent> Point3d
vvcomplex::DivisionData< JunctionContent >::pu`

Position of the junction inserted in the U wall.

Definition at line 837 of file complex.h.

Referenced by tissue::cellPinching(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell(), tissue::findDivisionPoints(), cell_system::findDivisionPoints(), vvcomplex::FindOppositeWall(), vvcomplex::testDivisionOnVertices(), and cell_system::volumeLeftCell().

17.24.5.4 **template<typename JunctionContent> Point3d vvcomplex::DivisionData< JunctionContent >::pv**

Position of the junction inserted in the V wall.

Definition at line 837 of file complex.h.

Referenced by tissue::cellPinching(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell(), tissue::findDivisionPoints(), cell_system::findDivisionPoints(), vvcomplex::FindOppositeWall(), vvcomplex::testDivisionOnVertices(), and cell_system::volumeLeftCell().

17.24.5.5 **template<typename JunctionContent> vertex vvcomplex::DivisionData< JunctionContent >::u1**

First junction of the 'U' wall, u2 is the vertex after u1 in the neighborhood of c.

Definition at line 838 of file complex.h.

Referenced by tissue::cellPinching(), vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::divideCell(), tissue::findDivisionPoints(), cell_system::findDivisionPoints(), vvcomplex::FindOppositeWall(), vvcomplex::DivisionData< JunctionContent >::operator bool(), vvcomplex::DivisionData< JunctionContent >::operator!(), vvcomplex::testDivisionOnVertices(), and cell_system::volumeLeftCell().

17.24.5.6 **template<typename JunctionContent> vertex vvcomplex::DivisionData< JunctionContent >::v1**

First junction of the 'V' wall, v2 is the vertex after v1 in the neighborhood of c.

Definition at line 838 of file complex.h.

Referenced by tissue::cellPinching(), vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::divideCell(), tissue::findDivisionPoints(), cell_system::findDivisionPoints(), vvcomplex::FindOppositeWall(), vvcomplex::DivisionData< JunctionContent

17.24 vvcomplex::DivisionData< JunctionContent > Class Template Reference

`>::operator bool()`, `vvcomplex::DivisionData< JunctionContent >::operator!()`, `vvcomplex::testDivisionOnVertices()`, and `cell_system::volumeLeftCell()`.

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

- `vvelib/algorithms/complex.h`

17.25 cell_system::DivisionParams Class Reference

Structure storing the right hand side of a division rule.

```
#include <algorithms/cellsystem.h>
```

Public Member Functions

- **DivisionParams** (const [DivisionParams](#) ©)
- **DivisionParams** ([label_t](#) ll, [label_t](#) rl, double a, double r=0.5)

Public Attributes

- double [angle](#)
Angle between the reference vector and the normal to the new wall.
- [label_t](#) [left_label](#)
Label of the "left" daughter cell.
- double [ratio](#)
Volume ratio of the left cell.
- [label_t](#) [right_label](#)
Label of the "right" daughter cell.

17.25.1 Detailed Description

Structure storing the right hand side of a division rule.

Definition at line 363 of file `cellsystem.h`.

17.25.2 Member Data Documentation

17.25.2.1 double cell_system::DivisionParams::angle

Angle between the reference vector and the normal to the new wall.

Definition at line 397 of file `cellsystem.h`.

Referenced by `cell_system::CellSystem< TissueClass, RealModel >::readRules()`, and `cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division()`.

17.25.2.2 label_t cell_system::DivisionParams::left_label

Label of the "left" daughter cell.

Definition at line 389 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readRules(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division().

17.25.2.3 double cell_system::DivisionParams::ratio

Volume ratio of the left cell.

Volume of the left cell divided by the volume of the original cell

Definition at line 403 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readRules(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division().

17.25.2.4 label_t cell_system::DivisionParams::right_label

Label of the "right" daughter cell.

Definition at line 393 of file cellsystem.h.

Referenced by cell_system::CellSystem< TissueClass, RealModel >::readRules(), and cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division().

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

- vvelib/algorithms/[cellsystem.h](#)

17.26 graph::Edge< EdgeContent > Class Template Reference

[Edge](#) of a vv graph.

```
#include <graph/edge.h>
```

Public Types

- typedef EdgeContent [content_t](#)
Type of the content of the edge.
- typedef [edge_identity_t](#) identity_t
Type of the identity of a vertex.
- typedef EdgeContent * [pointer](#)
Type of the equivalent pointer.

Public Member Functions

- void [clear](#) ()
Reset an edge weak pointer to null.
- [Edge](#) (const [Edge](#) ©)
Get a new weak reference on the copy.
- [Edge](#) ([identity_t](#) src, [identity_t](#) tgt, EdgeContent *content)
Creates an edge from `src` to `tgt` with a given content.
- [Edge](#) ()
Creates a null edge.
- bool [isNull](#) () const
Test if an edge is null.
- [operator bool](#) () const
Convert an edge to `true` if it is not null.
- bool [operator!=](#) (const [Edge](#) &other) const
Comparison operators.
- EdgeContent & [operator*](#) () const
Data access.

- EdgeContent * [operator->](#) () const
Data access.
- template<typename R >
const R & [operator->*](#) (R EdgeContent::*ptr) const
Constant access to the data via pointer to member.
- template<typename R >
R & [operator->*](#) (R EdgeContent::*ptr)
Access to the data via pointer to member.
- bool [operator<](#) (const [Edge](#) &other) const
Comparison operators.
- bool [operator<=](#) (const [Edge](#) &other) const
Comparison operators.
- [Edge](#) & [operator=](#) (const [Edge](#) &other)
Change the reference help by the object.
- bool [operator==](#) (const [Edge](#) &other) const
Comparison operators.
- bool [operator>](#) (const [Edge](#) &other) const
Comparison operators.
- bool [operator>=](#) (const [Edge](#) &other) const
Comparison operators.
- bool [serialize](#) (storage::VVEStorage &)
Serialization method.
- [identity_t](#) [source](#) () const
Returns the identifier of the source of the edge.
- [identity_t](#) [target](#) () const
Returns the identifier of the target of the edge.

Static Public Attributes

- static [Edge](#) [null](#)

Protected Attributes

- `EdgeContent * _content`
Content of the edge.
- `identity_t _source`
Identity of the source of the edge.
- `identity_t _target`
Identity of the target of the edge.

17.26.1 Detailed Description

template<typename EdgeContent> class graph::Edge< EdgeContent >

[Edge](#) of a vv graph. The edges represent weak references on the edges data. The data are owned by the graph. You must never try to access an edge that was deleted from its graph.

Definition at line 33 of file edge.h.

17.26.2 Member Typedef Documentation

**17.26.2.1 template<typename EdgeContent> typedef EdgeContent
graph::Edge< EdgeContent >::content_t**

Type of the content of the edge.

Definition at line 44 of file edge.h.

**17.26.2.2 template<typename EdgeContent> typedef edge_identity_t
graph::Edge< EdgeContent >::identity_t**

Type of the identity of a vertex.

Definition at line 39 of file edge.h.

**17.26.2.3 template<typename EdgeContent> typedef EdgeContent*
graph::Edge< EdgeContent >::pointer**

Type of the equivalent pointer.

Definition at line 49 of file edge.h.

17.26.3 Constructor & Destructor Documentation

17.26.3.1 `template<typename EdgeContent > graph::Edge< EdgeContent >::Edge () [inline]`

Creates a null edge.

Definition at line 222 of file edge.h.

```
223      : _source(0)
224      , _target(0)
225      , _content(0)
226      { }
```

17.26.3.2 `template<typename EdgeContent > graph::Edge< EdgeContent >::Edge (identity_t src, identity_t tgt, EdgeContent * content) [inline]`

Creates an edge from `src` to `tgt` with a given content.

The object do not take ownership of the content which must then be kept alive for as long as needed.

Note

This function is meant to be used by the graph, not really by the user of the VV library.

Definition at line 236 of file edge.h.

```
237      : _source(src)
238      , _target(tgt)
239      , _content(content)
240      { }
```

17.26.3.3 `template<typename EdgeContent > graph::Edge< EdgeContent >::Edge (const Edge< EdgeContent > & copy) [inline]`

Get a new weak reference on the copy.

Definition at line 229 of file edge.h.

```
230      : _source(copy._source)
231      , _target(copy._target)
232      , _content(copy._content)
233      { }
```

17.26.4 Member Function Documentation

17.26.4.1 `template<typename EdgeContent> void graph::Edge< EdgeContent >::clear () [inline]`

Reset an edge weak pointer to null.

Definition at line 185 of file edge.h.

References `graph::Edge< EdgeContent >::_content`, `graph::Edge< EdgeContent >::_source`, and `graph::Edge< EdgeContent >::_target`.

```
186     {
187         _source = 0;
188         _target = 0;
189         _content = 0;
190     }
```

17.26.4.2 `template<typename EdgeContent> bool graph::Edge< EdgeContent >::isNull () const [inline]`

Test if an edge is null.

Definition at line 160 of file edge.h.

References `graph::Edge< EdgeContent >::_content`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::connectFromJunctions()`.

```
160 { return _content == 0; }
```

17.26.4.3 `template<typename EdgeContent> graph::Edge< EdgeContent >::operator bool () const [inline]`

Convert an edge to `true` if it is not null.

Definition at line 165 of file edge.h.

References `graph::Edge< EdgeContent >::_content`.

```
165 { return _content != 0; }
```

17.26.4.4 `template<typename EdgeContent> bool graph::Edge< EdgeContent >::operator!= (const Edge< EdgeContent > & other) const [inline]`

Comparison operators.

Note

The comparison is done on the identity of the edge, not the content.

Definition at line 130 of file edge.h.

References graph::Edge< EdgeContent >::_content.

```
130 { return _content != other._content; }
```

17.26.4.5 **template<typename EdgeContent> EdgeContent& graph::Edge< EdgeContent >::operator* () const [inline]**

Data access.

Warning

Do not try to access the data of the null edge or of an edge that does not exist anymore in its graph.

Definition at line 90 of file edge.h.

References graph::Edge< EdgeContent >::_content.

```
90 { return *_content; }
```

17.26.4.6 **template<typename EdgeContent> EdgeContent* graph::Edge< EdgeContent >::operator-> () const [inline]**

Data access.

Warning

Do not try to access the data of the null edge or of an edge that does not exist anymore in its graph.

Definition at line 82 of file edge.h.

References graph::Edge< EdgeContent >::_content.

```
82 { return _content; }
```

17.26.4.7 **template<typename EdgeContent> template<typename R > const R& graph::Edge< EdgeContent >::operator->* (R EdgeContent::* ptr) const [inline]**

Constant access to the data via pointer to member.

Definition at line 104 of file edge.h.

References graph::Edge< EdgeContent >::_content.

```

105         {
106             return _content->*ptr;
107         }

```

17.26.4.8 `template<typename EdgeContent> template<typename R > R& graph::Edge< EdgeContent >::operator->* (R EdgeContent::* ptr) [inline]`

Access to the data via pointer to member.

Definition at line 96 of file edge.h.

References `graph::Edge< EdgeContent >::_content`.

```

97         {
98             return _content->*ptr;
99         }

```

17.26.4.9 `template<typename EdgeContent> bool graph::Edge< EdgeContent >::operator< (const Edge< EdgeContent > & other) const [inline]`

Comparison operators.

Note

The comparison is done on the identity of the edge, not the content.

Definition at line 142 of file edge.h.

References `graph::Edge< EdgeContent >::_content`.

```

142 { return _content < other._content; }

```

17.26.4.10 `template<typename EdgeContent> bool graph::Edge< EdgeContent >::operator<= (const Edge< EdgeContent > & other) const [inline]`

Comparison operators.

Note

The comparison is done on the identity of the edge, not the content.

Definition at line 155 of file edge.h.

References `graph::Edge< EdgeContent >::_content`.

```

155 { return _content <= other._content; }

```

17.26.4.11 `template<typename EdgeContent > Edge< EdgeContent >
& graph::Edge< EdgeContent >::operator= (const Edge<
EdgeContent > & other) [inline]`

Change the reference help by the object.

Definition at line 243 of file edge.h.

References `graph::Edge< EdgeContent >::_content`, `graph::Edge< EdgeContent >::_source`, and `graph::Edge< EdgeContent >::_target`.

```
244     {  
245         _source = other._source;  
246         _target = other._target;  
247         _content = other._content;  
248         return *this;  
249     }
```

17.26.4.12 `template<typename EdgeContent> bool graph::Edge<
EdgeContent >::operator== (const Edge< EdgeContent > & other)
const [inline]`

Comparison operators.

Note

The comparison is done on the identity of the edge, not the content.

Definition at line 124 of file edge.h.

References `graph::Edge< EdgeContent >::_content`.

```
124 { return _content == other._content; }
```

17.26.4.13 `template<typename EdgeContent> bool graph::Edge<
EdgeContent >::operator> (const Edge< EdgeContent > & other)
const [inline]`

Comparison operators.

Note

The comparison is done on the identity of the edge, not the content.

Definition at line 136 of file edge.h.

References `graph::Edge< EdgeContent >::_content`.

```
136 { return _content > other._content; }
```

17.26.4.14 `template<typename EdgeContent> bool graph::Edge< EdgeContent >::operator>= (const Edge< EdgeContent > & other) const [inline]`

Comparison operators.

Note

The comparison is done on the identity of the edge, not the content.

Definition at line 149 of file edge.h.

References `graph::Edge< EdgeContent >::_content`.

```
149 { return _content >= other._content; }
```

17.26.4.15 `template<typename EdgeContent> bool graph::Edge< EdgeContent >::serialize (storage::VVEStorage &)`

Serialization method.

Warning

You need to include `<storage/graph.h>` to use this serialization method

17.26.4.16 `template<typename EdgeContent> identity_t graph::Edge< EdgeContent >::source () const [inline]`

Returns the identifier of the source of the edge.

Note

You should rather use the `VVGraph::source()` method that returns the source vertex.

Definition at line 173 of file edge.h.

References `graph::Edge< EdgeContent >::_source`.

Referenced by `graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::source()`, and `graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::source()`.

```
173 { return _source; }
```

17.26.4.17 `template<typename EdgeContent> identity_t graph::Edge< EdgeContent >::target () const [inline]`

Returns the identifier of the target of the edge.

Note

You should rather use the [VVGraph::target\(\)](#) method that returns the target vertex.

Definition at line 180 of file edge.h.

References graph::Edge< EdgeContent >::_target.

Referenced by graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::target(), and graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::target().

```
180 { return _target; }
```

17.26.5 Member Data Documentation

17.26.5.1 `template<typename EdgeContent> EdgeContent* graph::Edge< EdgeContent >::_content [protected]`

Content of the edge.

Definition at line 214 of file edge.h.

Referenced by graph::Edge< EdgeContent >::clear(), graph::Edge< EdgeContent >::isNull(), graph::Edge< EdgeContent >::operator bool(), graph::Edge< EdgeContent >::operator!(), graph::Edge< EdgeContent >::operator*(), graph::Edge< EdgeContent >::operator->(), graph::Edge< EdgeContent >::operator->*, graph::Edge< EdgeContent >::operator<(), graph::Edge< EdgeContent >::operator<=(), graph::Edge< EdgeContent >::operator=(), graph::Edge< EdgeContent >::operator==((), graph::Edge< EdgeContent >::operator>(), and graph::Edge< EdgeContent >::operator>=().

17.26.5.2 `template<typename EdgeContent> identity_t graph::Edge< EdgeContent >::_source [protected]`

Identity of the source of the edge.

Definition at line 206 of file edge.h.

Referenced by graph::Edge< EdgeContent >::clear(), graph::Edge< EdgeContent >::operator=(), and graph::Edge< EdgeContent >::source().

17.26.5.3 `template<typename EdgeContent> identity_t graph::Edge< EdgeContent >::_target [protected]`

Identity of the target of the edge.

Definition at line 210 of file edge.h.

Referenced by `graph::Edge< EdgeContent >::clear()`, `graph::Edge< EdgeContent >::operator=()`, and `graph::Edge< EdgeContent >::target()`.

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

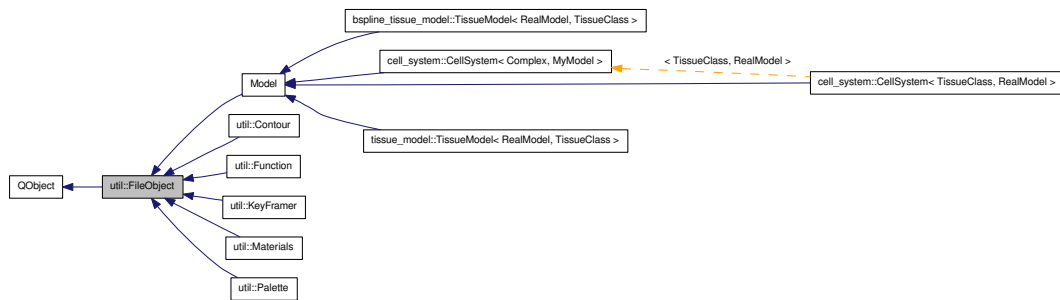
- `vvelib/graph/edge.h`

17.27 util::FileObject Class Reference

This class is the base class for any object that might be watched by the [WatchDog](#).

```
#include <util/watchdog.h>
```

Inheritance diagram for util::FileObject:



Signals

- void [modified](#) ()
Signal emitted to indicate a change in the object.

Public Member Functions

- [FileObject](#) (const [FileObject](#) ©, [QObject](#) *parent=0)
Copy constructor.
- [FileObject](#) ([QString](#) fn, [QObject](#) *parent=0)
Initialized constructor.
- [FileObject](#) (std::string fn, [QObject](#) *parent=0)
Initialized constructor.
- [FileObject](#) ([QObject](#) *parent=0)
Default constructor.
- std::string [getFilename](#) ()
Get the current filename of the object.
- [FileObject](#) & [operator=](#) (const [FileObject](#) &other)
Assignment operator.
- virtual void [reread](#) ()=0
Method to be implemented in subclasses to reread the file.

- void [setFilename](#) (const char *fn)
Change the file handled.
- void [setFilename](#) (QString fn)
Change the file handled.
- void [setFilename](#) (std::string fn)
Change the file handled.
- void **update** ()

Protected Attributes

- std::string **filename**

17.27.1 Detailed Description

This class is the base class for any object that might be watched by the [WatchDog](#). Anytime the `FileObject::update()` method is called, the [FileObject::modified\(\)](#) signal is emitted by the object.

Definition at line 30 of file `watchdog.h`.

17.27.2 Constructor & Destructor Documentation

17.27.2.1 `util::FileObject::FileObject (QObject *parent = 0) [inline]`

Default constructor.

Just initialize the **QObject**.

Definition at line 40 of file `watchdog.h`.

```
40 : QObject(parent) {}
```

17.27.2.2 `util::FileObject::FileObject (std::string fn, QObject *parent = 0) [inline]`

Initialized constructor.

Parameters

fn Name of the file handled by this object

Definition at line 46 of file `watchdog.h`.

```
46 : QObject(parent), filename(fn) {}
```


17.27.2.3 util::FileObject::FileObject (QString *fn*, QObject * *parent* = 0) [inline]

Initialized constructor.

Parameters

fn Name of the file handled by this object

Definition at line 52 of file watchdog.h.

```
52 : QObject(parent), filename(fn.toStdString()) {}
```

17.27.2.4 util::FileObject::FileObject (const FileObject & *copy*, QObject * *parent* = 0) [inline]

Copy constructor.

Parameters

copy Object to copy

Definition at line 58 of file watchdog.h.

```
58 : QObject(parent), filename(copy.filename) {}
```

17.27.3 Member Function Documentation

17.27.3.1 std::string util::FileObject::getFilename () [inline]

Get the current filename of the object.

Definition at line 91 of file watchdog.h.

Referenced by util::WatchDog::addObject(), util::WatchDog::changeFilename(), util::WatchDog::removeObject(), and util::KeyFramer::reread().

```
91 { return filename; }
```

17.27.3.2 void util::FileObject::modified () [signal]

Signal emitted to indicate a change in the object.

17.27.3.3 FileObject& util::FileObject::operator= (const FileObject & *other*) [inline]

Assignment operator.

Parameters

other Object to copy

If the object to copy handle a different filename, then the [FileObject::modified\(\)](#) signal is emitted.

Definition at line 67 of file watchdog.h.

```
68      { if (other.filename != filename) { filename = other.filename; } return *this;
      }
```

17.27.3.4 virtual void util::FileObject::reread () [pure virtual]

Method to be implemented in subclasses to reread the file.

Implemented in [Model](#), [util::Contour](#), [util::Function](#), [util::KeyFramer](#), [util::Materials](#), and [util::Palette](#).

17.27.3.5 void util::FileObject::setFilename (const char * *fn*) [inline]

Change the file handled.

If the call actually change the file handled, then [FileObject::modified\(\)](#) is emitted.

Definition at line 115 of file watchdog.h.

References [setFilename\(\)](#).

Referenced by [setFilename\(\)](#).

```
115 { setFilename(std::string(fn)); }
```

17.27.3.6 void util::FileObject::setFilename (QString *fn*) [inline]

Change the file handled.

If the call actually change the file handled, then [FileObject::modified\(\)](#) is emitted.

Definition at line 107 of file watchdog.h.

References [setFilename\(\)](#).

Referenced by [setFilename\(\)](#).

```
107 { setFilename(fn.toStdString()); }
```

17.27.3.7 void util::FileObject::setFilename (std::string *fn*)

Change the file handled.

If the call actually change the file handled, then [FileObject::modified\(\)](#) is emitted.

Definition at line 15 of file watchdog.cpp.

Referenced by util::WatchDog::changeFilename().

```
16  {  
17      if (fn != filename)  
18      {  
19          filename = fn;  
20          if (!fn.empty())  
21              update();  
22      }  
23  }
```

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

- vvelib/util/[watchdog.h](#)
- vvelib/util/watchdog.cpp

17.28 storage::Frame Struct Reference

Data holding properties of a frame.

```
#include <storage_bin.h>
```

Public Member Functions

- **Frame** (const [Frame](#) ©)
- **Frame** (int u=0, bool h=false)

Public Attributes

- bool **has_fields**
- int **unnamed_subframes**

17.28.1 Detailed Description

Data holding properties of a frame.

Definition at line 27 of file storage_bin.h.

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

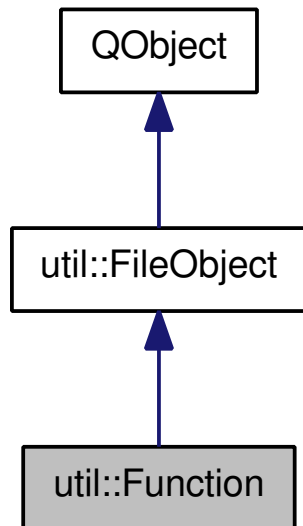
- vvelib/storage/storage_bin.h

17.29 util::Function Class Reference

A utility class for functions.

```
#include <util/function.h>
```

Inheritance diagram for util::Function:



Public Member Functions

- **bool error ()**
- **Function** (const **Function** ©)
- **Function** (std::string filename)
Constructor with initialisation from file.
- **Function** ()
Default constructor.
- const **util::Vector**< 2, double > & **getMax** () const
Return the maximum x and y values.
- const **util::Vector**< 2, double > & **getMin** () const
Return the minimum x and y values.
- void **normalizeX** (bool shift=true)
Make sure the x axis spans from 0 to 1.
- void **normalizeY** (bool shift=true)
Make sure the y axis spans from 0 to 1.

- double `operator()` (double x)
Function operator.
- `Function & operator=` (const `Function` &)
Copy constructor.
- void `reread` ()
Method to be implemented in subclasses to reread the file.
- double `scaleX` () const
Get the current x scaling.
- void `scaleX` (double s)
Scale the x axis by s.
- double `scaleY` () const
Get the current y scaling.
- void `scaleY` (double s)
Scale the y axis by s.
- void `setSamples` (size_t n)
sets the number of samples to be used
- double `shiftX` () const
Get the current x axis shift.
- void `shiftX` (double s)
Shift the x axis by s.
- double `shiftY` () const
Get the current y axis shift.
- void `shiftY` (double s)
Shift the y axis by s.

17.29.1 Detailed Description

A utility class for functions. This class is a function object that encapsulates functions in the VLAB function formats (original and fver 1 1).

Definition at line 24 of file function.h.

17.29.2 Constructor & Destructor Documentation

17.29.2.1 util::Function::Function ()

Default constructor.

Definition at line 33 of file function.cpp.

```
34         :  
35     pts(),  
36     max(),  
37     min(),  
38     scaling_x(1),  
39     scaling_y(1),  
40     shift_x(0),  
41     shift_y(0)  
42     {  
43         init();  
    }
```

17.29.2.2 util::Function::Function (std::string filename)

Constructor with initialisation from file.

Bug

If the specified file contains an incomplete or improper specification, the result of construction will be unknown. Most probably, the object will unuseable; however, this is not currently reported.

Parameters

filename The function file to load.

Definition at line 52 of file function.cpp.

References reread().

```
53     : FileObject(filename)  
54     , pts()  
55     , max()  
56     , min()  
57     , scaling_x(1)  
58     , scaling_y(1)  
59     , shift_x(0)  
60     , shift_y(0)  
61     {  
62         init();  
63         reread();  
64     }
```

17.29.3 Member Function Documentation

17.29.3.1 `const Vector< 2, double > & util::Function::getMax () const`

Return the maximum x and y values.

Definition at line 488 of file function.cpp.

```
488                                     {
489     return max;
490 }
```

17.29.3.2 `const Vector< 2, double > & util::Function::getMin () const`

Return the minimum x and y values.

Definition at line 493 of file function.cpp.

```
493                                     {
494     return min;
495 }
```

17.29.3.3 `void util::Function::normalizeX (bool shift = true)`

Make sure the x axis spans from 0 to 1.

Parameters

shift If true, it will also change the shift of the axis

Definition at line 334 of file function.cpp.

References `util::Vector< dim, T >::x()`.

```
335     {
336         if(shift)
337         {
338             shift_x = min.x();
339             scaling_x = max.x() - min.x();
340         }
341         else
342         {
343             scaling_x = max.x();
344         }
345     }
```

17.29.3.4 `void util::Function::normalizeY (bool shift = true)`

Make sure the y axis spans from 0 to 1.

Parameters

shift If true, it will also change the shift of the axis

Definition at line 351 of file function.cpp.

References util::Vector< dim, T >::x().

```

352     {
353         double ymin = HUGE_VAL, ymax = -HUGE_VAL;
354         double dx = (max.x() - min.x()) / samples;
355         double x0 = min.x();
356         if (! cache_valid) {
357             cache_valid = true;
358             cache.resize (samples+1);
359             for (size_t i = 0 ; i < samples ; i ++)
360                 cache[i].valid = false;
361         }
362         for(size_t i = 0 ; i < samples+1 ; ++i)
363         {
364             double y;
365             if(cache[i].valid)
366             {
367                 y = cache[i].val;
368             }
369             else
370             {
371                 y = getVal(x0+dx*i);
372                 cache[i].valid = true;
373                 cache[i].val = y;
374             }
375             if(y < ymin) ymin = y;
376             if(y > ymax) ymax = y;
377         }
378         if(shift)
379         {
380             shift_y = -ymin;
381             if(ymax > ymin)
382                 scaling_y = 1/(ymax-ymin);
383             else
384                 scaling_y = 1;
385         }
386         else
387         {
388             if(ymax != 0)
389             {
390                 scaling_y = 1/ymax;
391             }
392             else
393             {
394                 scaling_y = 1;
395             }
396         }
397     }

```

17.29.3.5 double util::Function::operator() (double x)

[Function](#) operator.

Parameters

x The position in the function.

This returns the y-value of the function for a given x-value. If x is outside the domain of the function, the value of the function start or end point is returned. Dynamic caching is used to speed things up. Basically, the function is only evaluated at a number of places (specified in 'samples'), and otherwise the values are linearly interpolated inbetween. The real values are obtained by calling `getVal ()`.

Definition at line 295 of file `function.cpp`.

References `util::Vector< dim, T >::x()`.

```

295                                     {
296     x += shift_x;
297     x *= scaling_x;
298     if (x <= min.x()) return scaling_y*pts[0].y() + shift_y;
299     if (x >= max.x()) return scaling_y*pts[pts.size() - 1].y() + shift_y;
300
301     // check for cache
302     if (! cache_valid) {
303         cache_valid = true;
304         cache.resize (samples+1);
305         for (size_t i = 0 ; i < samples ; i ++)
306             cache[i].valid = false;
307     }
308     // lower index
309     double dx = (max.x() - min.x()) / samples;
310     size_t lo = (size_t) ((x - min.x()) / dx);
311     size_t hi = lo + 1;
312     // make sure we have both lo and hi
313     double lox = min.x() + dx * lo;
314     double hix = lox + dx;
315     if (! cache [lo].valid) {
316         cache [lo].valid = true;
317         cache [lo].val = getVal (lox);
318     }
319     if (! cache [hi].valid) {
320         cache [hi].valid = true;
321         cache [hi].val = getVal (hix);
322     }
323     // linearly interpolate
324     double r = (x - lox) / dx;
325     double ret = (1-r) * cache[lo].val + r * cache[hi].val;
326
327     return scaling_y*ret + shift_y;
328 }
```

17.29.3.6 Function & `util::Function::operator= (const Function &f)`

Copy constructor.

Definition at line 83 of file `function.cpp`.

```

83                                     {
84     FileObject::operator=(f);
```

```

85     pts = f.pts;
86     max = f.max;
87     min = f.min;
88     samples = f.samples;
89     cache_valid = f.cache_valid;
90     cache = f.cache;
91     error_occured = f.error_occured;
92     scaling_x = f.scaling_x;
93     scaling_y = f.scaling_y;
94     shift_x = f.shift_x;
95     shift_y = f.shift_y;
96     return *this;
97 }

```

17.29.3.7 void util::Function::reread () [virtual]

Method to be implemented in subclasses to reread the file.

Implements [util::FileObject](#).

Definition at line 103 of file function.cpp.

References [QFile::close\(\)](#), [QString::contains\(\)](#), [QString::fromStdString\(\)](#), [QFile::handle\(\)](#), [QFile::open\(\)](#), [QString::split\(\)](#), [QString::startsWith\(\)](#), [util::Vector< dim, T >::x\(\)](#), and [util::Vector< dim, T >::y\(\)](#).

Referenced by [Function\(\)](#).

```

103     {
104         QFile f(QString::fromStdString(filename));
105         error_occured = false;
106         if(!f.open(QIODevice::ReadOnly))
107         {
108             error_occured = true;
109             err << "Error opening file " << QString::fromStdString(filename) << ": " <<
f.errorString() << endl;
110             return;
111         }
112 #if defined(__APPLE__) || defined(linux)
113         flock(f.handle(), LOCK_SH);
114 #endif
115         QTextStream ts(&f);
116         int ptid = 0, line_nb = 0;
117
118         max = Vector<2,double>(-HUGE_VAL,-HUGE_VAL);
119         min = Vector<2,double>(HUGE_VAL,HUGE_VAL);
120
121         while(!ts.atEnd())
122         {
123             QString line = ts.readLine().trimmed();
124             line_nb++;
125             if(line.startsWith("fver"))
126             {
127                 QStringList fields = line.split(" ");
128                 if(fields.size() != 3)
129                 {
130                     REPORT_ERROR(filename, line_nb, "cannot find major and minor in functio
n version (fver)");
131                     return;

```

```

132     }
133     bool ok;
134     int major = fields[1].toInt(&ok);
135     if(!ok)
136     {
137         REPORT_ERROR(filename, line_nb, "major version number (" << fields[1] <
< " ) is not a valid integer");
138         return;
139     }
140     int minor = fields[2].toInt(&ok);
141     if(!ok)
142     {
143         REPORT_ERROR(filename, line_nb, "minor version number (" << fields[2] <
< " ) is not a valid integer");
144         return;
145     }
146     if(major != 1 or minor != 1)
147     {
148         err << "Warning, this version usually deals with fver 1 1. Processing w
ill continue but may fail" << endl;
149     }
150 }
151 else if(line.startsWith("points:"))
152 {
153     QStringList fields = line.split(":");
154     if(fields.size() != 2)
155     {
156         REPORT_ERROR(filename, line_nb, "Invalid point number specification, th
ere is more than one ':'");
157         return;
158     }
159     bool ok;
160     unsigned int nbpts = fields[1].toUInt(&ok);
161     if(!ok)
162     {
163         REPORT_ERROR(filename, line_nb, "Number of points (" << fields[1] << "
) is not a valid integer");
164         return;
165     }
166     pts.resize(nbpts);
167 }
168 else if(!line.contains(":")) // specification of a points
169 {
170     if(ptid >= (int)pts.size())
171     {
172         REPORT_ERROR(filename, line_nb, "There are more points than specified")
;
173         return;
174     }
175     QStringList coords = line.split(" ");
176     if(coords.size() != 2)
177     {
178         REPORT_ERROR(filename, line_nb, "There are " << coords.size() << " coor
dinates instead of 2");
179         return;
180     }
181     double x, y;
182     bool ok;
183     x = coords[0].toDouble(&ok);
184     if(!ok)
185     {
186         REPORT_ERROR(filename, line_nb, coords[0] << " ' is not a valid double")

```

```

;
187         return;
188     }
189     y = coords[1].toDouble(&ok);
190     if(!ok)
191     {
192         REPORT_ERROR(filename, line_nb, coords[1] << "' is not a valid double")
;
193         return;
194     }
195     if(x < min.x()) min.x() = x;
196     if(y < min.y()) min.y() = y;
197     if(x > max.x()) max.x() = x;
198     if(y > max.y()) max.y() = y;
199     Vector<2,double> p(x, y);
200     pts[ptid++] = p;
201 }
202 // Otherwise, just ignore the line
203 }
204 if(ptid != (int)pts.size())
205 {
206     error_occured = true;
207     err << "Error in file " << QString::fromStdString(filename)
208         << ", there was " << pts.size() << " points announced, but only " << ptid
<< " points declared" << endl;
209     return;
210 }
211
212     cache_valid = false;
213 #if defined(__APPLE__) || defined(linux)
214     flock(f.handle(), LOCK_UN);
215 #endif
216     f.close();
217     // ifstream in(filename.c_str());
218     // string buffer;
219
220     // if (!in || !in.good() || in.eof()) {
221     //     error_occured = true;
222     //     std::cerr << "Function - Cannot open file '" << filename << "'\n";
223     //     return;
224     // }
225     // error_occured = false;
226
227     // in >> ws >> buffer;
228
229     // if (buffer == string("range:")) {
230     //     // old version
231
232     //     double rmin, rmax;
233     //     in >> rmin >> rmax;
234     // }
235     // else if (buffer == string("fver")) {
236     //     int major, minor;
237     //     in >> major >> minor >> ws;
238
239     //     if (major == 1 && minor == 1) {
240     //         getline(in, buffer); // name
241     //         getline(in, buffer); // samples
242     //         getline(in, buffer); // flip
243     //     }
244     // }
245

```

```

246     // unsigned int num;
247     // in >> buffer >> num;
248
249     // pts.reserve(num);
250
251     // bool first = true;
252     // for (unsigned int i = 0; i < num; i++) {
253     //     double x, y;
254     //     in >> x >> y;
255
256     //     if (first) {
257     //         max.set(x, y);
258     //         min.set(x, y);
259     //         first = false;
260     //     }
261     //     else {
262     //         if (x > max.x()) max.x(x);
263     //         if (y > max.y()) max.y(y);
264     //         if (x < min.x()) min.x(x);
265     //         if (y < min.y()) min.y(y);
266     //     }
267
268     //     Vector<2,double> p(x, y);
269     //     pts.push_back(p);
270     // }
271
272     // cache_valid = false;
273 }
```

17.29.3.8 double util::Function::scaleX () const [inline]

Get the current x scaling.

Definition at line 51 of file function.h.

```
51 { return 1/scaling_x; }
```

17.29.3.9 void util::Function::scaleX (double s) [inline]

Scale the x axis by s.

Parameters

s Scaling factor to apply to the axis

Definition at line 44 of file function.h.

```
44 { scaling_x = 1/s; }
```

17.29.3.10 double util::Function::scaleY () const [inline]

Get the current y scaling.

Definition at line 53 of file function.h.

```
53 { return scaling_y; }
```

17.29.3.11 void util::Function::scaleY (double *s*) [inline]

Scale the y axis by *s*.

Parameters

s Scaling factor to apply to the axis

Definition at line 49 of file function.h.

```
49 { scaling_y = s; }
```

17.29.3.12 void util::Function::setSamples (size_t *n*)

sets the number of samples to be used

Parameters

n The number of samples to use

This sets the new number of samples and invalidates the cache.

Definition at line 280 of file function.cpp.

```
281 {  
282     samples = (unsigned int)n;  
283     cache_valid = false;  
284 }
```

17.29.3.13 double util::Function::shiftX () const [inline]

Get the current x axis shift.

Definition at line 71 of file function.h.

```
71 { return -shift_x; }
```

17.29.3.14 void util::Function::shiftX (double *s*) [inline]

Shift the x axis by *s*.

Parameters

s Shift of the axis

Note that the shift happens after the scaling, so it should be written in the scaled reference system.

Definition at line 61 of file function.h.

```
61 { shift_x = -s; }
```

17.29.3.15 double util::Function::shiftY () const [inline]

Get the current y axis shift.

Definition at line 73 of file function.h.

```
73 { return shift_y; }
```

17.29.3.16 void util::Function::shiftY (double s) [inline]

Shift the y axis by s.

Parameters

s Shift of the axis

Note that the shift happens after the scaling, so it should be written in the scaled reference system.

Definition at line 69 of file function.h.

```
69 { shift_y = s; }
```

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

- vvelib/util/[function.h](#)
- vvelib/util/function.cpp

17.30 util::GraphInset Class Reference

Class used to draw a graph of time stamped data in the model window.

```
#include <util/graph_inset.h>
```

Classes

- struct [Data](#)
Data structure used to store the time stamped values.

Public Types

- typedef std::list< [Data](#) >::const_iterator [iterator](#)
Constant iterator on the stamped data.
- typedef const [Data](#) * [pointer](#)
Type of a pointer on the data.
- typedef const [Data](#) & [reference](#)
Type of a reference on the data.
- typedef std::list< [Data](#) >::const_reverse_iterator [reverse_iterator](#)
Constant reverse iterator on the stamped data.
- typedef [Data](#) [value_type](#)
Type of the data.

Public Member Functions

- void [addData](#) (double [time](#), const double *values, bool changed=false)
Add time stamped values.
- template<size_t N>
void [addData](#) (double [time](#), const [util::Vector](#)< N, double > &values, bool changed=false)
Add time stamped values.
- void [addData](#) (double [time](#), double value1, double value2, double value3, double value4, double value5, bool changed=false)
Helper to add time stamped values.
- void [addData](#) (double [time](#), double value1, double value2, double value3, double value4, bool changed=false)

Helper to add time stamped values.

- void **addData** (double **time**, double value1, double value2, double value3, bool changed=false)

Helper to add time stamped values.

- void **addData** (double **time**, double value1, double value2, bool changed=false)

Helper to add time stamped values.

- void **addData** (double **time**, double value, bool changed=false)

Helper to add time stamped values.

- void **addData** (double **time**, const std::vector< double > &values, bool changed=false)

Add time stamped values.

- void **check** ()

Check if the object is correctly initialized.

- void **cleanData** ()

Remove all data that won't be drawn because they are older than the time span of the graph.

- void **clear** ()

Remove all data from the graph.

- void **closeGraph** ()

Order the graph to close.

- void **dataSourceChanged** ()

- void **draw** (**Viewer** *viewer, const **util::Palette** &palette)

Draw the graph in the viewer.

- void **drawFrame** (size_t id, **Viewer** *viewer)

Draw the frame only in the viewer.

- void **drawWithName** (size_t id, **Viewer** *viewer)

Draw the graph in the viewer with a given name.

- bool **graphClosed** () const

Tell if the graph is closed.

- **GraphInset** (int nb_lines=0)

Construct a new graph inset.

- int **numberOfLines** () const

Returns the numbers of lines drawn.

- void [openGraph](#) ()
Order the graph to open up.
- void [readParms](#) (const **QString** §ion, [util::Parms](#) parms)
Read the parameters for the graph.
- void [setNumberOfLines](#) (size_t nb_lines)
Change the number of lines.
- void [setTime](#) (double t)
Kepp the graph informed of the current time in the system.
- bool [writeCSV](#) (const **QString** &filename)
Write the data in a .csv file.

STL-like methods

- [reference back](#) () const
- [iterator begin](#) () const
- [iterator end](#) () const
- [reference front](#) () const
- [reverse_iterator rbegin](#) () const
- [reverse_iterator rend](#) () const
- size_t [size](#) () const
Number of values stored.

Public Attributes

- bool [autoOpenClose](#)
Automatically open/close the graph.
- int [backColor](#)
Color of the back of the graph.
- double [backgroundOffset](#)
Offset to draw the background of the graph.
- int [changeSourceColor](#)
Color of the line used to indicate a data change (typically if the data correspond to another cell).
- double [changeSourceOffset](#)
Offset to draw the data source change line.

- int [changeSourceWidth](#)
Width of the line used to indicate a data change (typically if the data correspond to another cell).
- int [contourColor](#)
Color of the contour.
- double [contourOffset](#)
Offset to draw the contour of the graph.
- int [contourWidth](#)
Width of the line used for the contour.
- bool [keepAllData](#)
Order the object to keep all the data.
- std::vector< int > [linesColor](#)
Color of the lines.
- std::vector< [util::Vector](#)< 2, double > > [linesMinMaxView](#)
Extrema value shown for each line.
- std::vector< double > [linesOffset](#)
Offset to draw the lines.
- std::vector< double > [linesWidth](#)
Width of the lines.
- bool [showAll](#)
If true, show all the data.
- std::vector< size_t > [showIndices](#)
If not showing all data, contains the indices to show.
- double [time](#)
Time of the system.
- double [timeSpan](#)
Time span to keep the data.
- [util::Vector](#)< 2, int > [windowPosition](#)
Position of the top-left corner of the graph window in pixels.
- [util::Vector](#)< 2, size_t > [windowSize](#)
Size of the graph window in pixels.

Protected Attributes

- size_t [_numberOfLines](#)
Number of lines drawn by the graph.
- bool [closed](#)
State if the graph window should be closed.
- std::list< [Data](#) > [data](#)
Data to be draw.
- bool [initialized](#)
Check the system is correctly initialized.

17.30.1 Detailed Description

Class used to draw a graph of time stamped data in the model window.

Definition at line 30 of file graph_inset.h.

17.30.2 Member Typedef Documentation

17.30.2.1 typedef std::list<Data>::const_iterator util::GraphInset::iterator

Constant iterator on the stamped data.

Definition at line 69 of file graph_inset.h.

17.30.2.2 typedef const Data* util::GraphInset::pointer

Type of a pointer on the data.

Definition at line 83 of file graph_inset.h.

17.30.2.3 typedef const Data& util::GraphInset::reference

Type of a reference on the data.

Definition at line 87 of file graph_inset.h.

17.30.2.4 typedef std::list<Data>::const_reverse_iterator util::GraphInset::reverse_iterator

Constant reverse iterator on the stamped data.

Definition at line 74 of file graph_inset.h.

17.30.2.5 typedef Data util::GraphInset::value_type

Type of the data.

Definition at line 79 of file graph_inset.h.

17.30.3 Constructor & Destructor Documentation

17.30.3.1 util::GraphInset::GraphInset (int *nb_lines* = 0)

Construct a new graph inset.

Parameters

nb_lines Number of lines to draw (i.e. nb of values given each time)

Definition at line 7 of file graph_inset.cpp.

Referenced by readParms().

```
16 {
```

17.30.4 Member Function Documentation

17.30.4.1 void util::GraphInset::addData (double *time*, const double * *values*, bool *changed* = false) [inline]

Add time stamped values.

Definition at line 187 of file graph_inset.h.

References `_numberOfLines`, and `addData()`.

```
188 {
189     std::vector<double> vec_values(_numberOfLines);
190     for(size_t i = 0 ; i < _numberOfLines ; ++i)
191     {
192         vec_values[i] = values[i];
193     }
194     addData(time, vec_values, changed);
195 }
```

17.30.4.2 template<size_t N> void util::GraphInset::addData (double *time*, const util::Vector< N, double > & *values*, bool *changed* = false) [inline]

Add time stamped values.

Definition at line 174 of file graph_inset.h.

References `addData()`.

```
175     {
176         std::vector<double> vec_values(N);
177         for(size_t i = 0 ; i < N ; ++i)
178         {
179             vec_values[i] = values[i];
180         }
181         addData(time, vec_values, changed);
182     }
```

17.30.4.3 void util::GraphInset::addData (double *time*, double *value1*, double *value2*, double *value3*, double *value4*, double *value5*, bool *changed* = **false**) [**inline**]

Helper to add time stamped values.

Definition at line 160 of file graph_inset.h.

References addData().

```
161     {
162         std::vector<double> vs(5, value1);
163         vs[1] = value2;
164         vs[2] = value3;
165         vs[3] = value4;
166         vs[4] = value5;
167         addData(time, vs, changed);
168     }
```

17.30.4.4 void util::GraphInset::addData (double *time*, double *value1*, double *value2*, double *value3*, double *value4*, bool *changed* = **false**) [**inline**]

Helper to add time stamped values.

Definition at line 148 of file graph_inset.h.

References addData().

```
149     {
150         std::vector<double> vs(4, value1);
151         vs[1] = value2;
152         vs[2] = value3;
153         vs[3] = value4;
154         addData(time, vs, changed);
155     }
```

17.30.4.5 void util::GraphInset::addData (double *time*, double *value1*, double *value2*, double *value3*, bool *changed* = **false**) [**inline**]

Helper to add time stamped values.

Definition at line 137 of file graph_inset.h.

References addData().

```

138     {
139         std::vector<double> vs(3, value1);
140         vs[1] = value2;
141         vs[2] = value3;
142         addData(time, vs, changed);
143     }

```

17.30.4.6 void util::GraphInset::addData (double *time*, double *value1*, double *value2*, bool *changed* = false) [inline]

Helper to add time stamped values.

Definition at line 127 of file graph_inset.h.

References addData().

```

128     {
129         std::vector<double> vs(2, value1);
130         vs[1] = value2;
131         addData(time, vs, changed);
132     }

```

17.30.4.7 void util::GraphInset::addData (double *time*, double *value*, bool *changed* = false) [inline]

Helper to add time stamped values.

Definition at line 118 of file graph_inset.h.

References addData().

```

119     {
120         std::vector<double> vs(1, value);
121         addData(time, vs, changed);
122     }

```

17.30.4.8 void util::GraphInset::addData (double *time*, const std::vector<double> & *values*, bool *changed* = false)

Add time stamped values.

Parameters

time Timestamp

values Values to add, must be in equal number as declared before

changed Indicate the source of the value changed (a line will be drawn at that time on the graph)

Definition at line 86 of file graph_inset.cpp.

Referenced by addData().


```
94 {
```

17.30.4.9 void util::GraphInset::check ()

Check if the object is correctly initialized.

This function must be called before using the object. It is automatically called within readParms.

Definition at line 219 of file graph_inset.cpp.

```
226 {
227     util::Palette palette;
228     glPushName((GLuint)id);
229     draw(viewer,palette);
230     glPopName();
231 }
232
233 void GraphInset::check()
234 {
235     initialized = false;
236     if(_numberOfLines <= 0)
237     {
238         cerr << "Number of lines <= 0" << endl;
239         return;
240     }
241     forall(const Data& d, data)
242     {
243         if(d.values.size() != _numberOfLines)
244         {
245             cerr << "Number of values inconsistant with the number of lines" << endl;
246             return;
247         }
248     }
249     if(linesWidth.size() < showIndices.size())
250     {
251         cerr << "Number of line widths < number of shown lines" << endl;
252         return;
253     }
254     if(linesColor.size() < showIndices.size())
255     {
256         cerr << "Number of line colors < number of shown lines" << endl;
257         return;
258     }
259     if(linesMinMaxView.size() < showIndices.size())
260     {
261         cerr << "Number of line min max views < number of shown lines" << endl;
262         return;
263     }
264     if(linesOffset.size() < showIndices.size())
265     {
266         cerr << "Number of line offsets < number of shown lines" << endl;
267         return;
268     }
269     forall(double& w, linesWidth)
270     {
271         if(w < 0)
272         {
```

```

273         cerr << "Line width < 0" << endl;
274         return;
275     }
276 }
277 forall(int& c, linesColor)
278 {
279     if(c < 0 || c > 255)
280     {
281         cerr << "Line color < 0 or > 255" << endl;
282         return;
283     }
284 }
285 if(contourWidth < 0)
286 {
287     cerr << "Contour width < 0" << endl;
288     return;
289 }
290 if(contourColor < 0 || contourColor > 255)
291 {
292     cerr << "Contour color < 0 or > 255" << endl;

```

17.30.4.10 void util::GraphInset::cleanData ()

Remove all data that won't be drawn because they are older than the time span of the graph.

Definition at line 101 of file graph_inset.cpp.

References autoOpenClose, data, openGraph(), and time.

```

101 {
102     data.push_back(Data(t, values, changed));
103     if(autoOpenClose)
104         openGraph();
105     if(time < t)
106         time = t;
107 }
108
109 void GraphInset::dataSourceChanged()

```

17.30.4.11 void util::GraphInset::clear ()

Remove all data from the graph.

Definition at line 79 of file graph_inset.cpp.

```

81 {
82     for(size_t i = 0 ; i < showIndices.size() ; ++i)
83     {
84         if(showIndices[i] > nb_lines)

```

17.30.4.12 void util::GraphInset::closeGraph ()

Order the graph to close.

Definition at line 294 of file graph_inset.cpp.

```
296     {
297         cerr << "Source width < 0" << endl;
```

17.30.4.13 void util::GraphInset::draw (Viewer *viewer, const util::Palette &palette)

Draw the graph in the viewer.

Definition at line 111 of file graph_inset.cpp.

```
116     {
117         while(!data.empty() && (time - data.front().time) > timeSpan)
118         {
119             data.pop_front();
120         }
121         if(autoOpenClose && data.empty())
122             closeGraph();
123     }
124
125     void GraphInset::draw(Viewer* viewer, const util::Palette& palette)
126     {
127         if(!initialized)
128         {
129             cerr << "Trying to draw an uninitialized graph" << endl;
130             return;
131         }
132         if(!keepAllData)
133             cleanData();
134         if(!closed)
135         {
136             glPushAttrib(GL_LIGHTING_BIT|GL_LINE_BIT|GL_POLYGON_BIT);
137             glDisable(GL_LIGHTING);
138             QSize wndSize = viewer->size();
139             int x1,y1,x2,y2;
140             if(windowPosition.x() < 0)
141                 x1 = wndSize.width() + windowPosition.x();
142             else
143                 x1 = windowPosition.x();
144             if(windowPosition.y() < 0)
145                 y1 = wndSize.height() + windowPosition.y();
146             else
147                 y1 = windowPosition.y();
148             x2 = x1+(int)windowSize.x();
149             y2 = y1+(int)windowSize.y();
150             viewer->startScreenCoordinatesSystem();
151             // First draw a square
152             glPolygonMode(GL_FRONT, GL_FILL);
153             if(backColor >= 0)
154             {
155                 palette.useColor(backColor);
156                 glBegin(GL_POLYGON);
157                 glVertex3d(x1,y1,backgroundOffset);
158                 glVertex3d(x1,y2,backgroundOffset);
159                 glVertex3d(x2,y2,backgroundOffset);
160                 glVertex3d(x2,y1,backgroundOffset);
161                 glEnd();
```

```

162     }
163     glPolygonMode(GL_FRONT, GL_LINE);
164     palette.useColor(contourColor);
165     glLineWidth(contourWidth);
166     glBegin(GL_POLYGON);
167     glVertex3d(x1,y1,contourOffset);
168     glVertex3d(x1,y2,contourOffset);
169     glVertex3d(x2,y2,contourOffset);
170     glVertex3d(x2,y1,contourOffset);
171     glEnd();
172     if((data.size() > 1) && (time - data.back().time) < timeSpan)
173     {
174         std::list<Data>::reverse_iterator it;
175         double ratio_x = windowSize.x()/timeSpan;
176         for(size_t idx = 0 ; idx < showIndices.size() ; ++idx)
177         {
178             size_t k = showIndices[idx]-1;
179             if(k >= _numberOfLines)
180                 continue;
181             glLineWidth(linesWidth[k]);
182             double ratio_y = (windowSize.y()-contourWidth)/(linesMinMaxView[k][1] -
linesMinMaxView[k][0]);
183             double shift_y = linesMinMaxView[k][0];
184             palette.useColor(linesColor[k]);
185             glBegin(GL_LINE_STRIP);
186             for(it = data.rbegin() ; it != data.rend() ; ++it)
187             {
188                 const Data& d = *it;
189                 if((time - d.time) > timeSpan)
190                     break;
191                 const std::vector<double>& values = d.values;
192                 double x = x2 + (d.time - time)*ratio_x;
193                 glVertex3d(x, y2 - std::max(std::min((values[k]-shift_y)*ratio_y, (double)windowSize.y()), 0.0), linesOffset[k]);
194                 if(d.changed)
195                 {
196                     glEnd();
197                     glBegin(GL_LINE_STRIP);
198                 }
199             }
200             glEnd();
201         }
202         palette.useColor(changeSourceColor);
203         glLineWidth(changeSourceWidth);
204         glBegin(GL_LINES);
205         for(it = data.rbegin() ; it != data.rend() ; ++it)
206         {
207             const Data& d = *it;
208             if((time - d.time) > timeSpan)
209                 break;

```

17.30.4.14 void util::GraphInset::drawFrame (size_t id, Viewer * viewer)

Draw the frame only in the viewer.

Used for selection purposes.

17.30.4.15 void util::GraphInset::drawWithName (size_t id, Viewer * viewer)

Draw the graph in the viewer with a given name.

Used for selection purposes. Useful if the background is selected as transparent.

Definition at line 211 of file graph_inset.cpp.

References `changeSourceOffset`, and `time`.

```
211      {
212          //          double x = x1 + (d.time-shift_time)*ratio_x;
213          double x = x2 + (d.time - time)*ratio_x;
214          glVertex3d(x, y1, changeSourceOffset);
215          glVertex3d(x, y2, changeSourceOffset);
216      }
217  }
```

17.30.4.16 bool util::GraphInset::graphClosed () const [inline]

Tell if the graph is closed.

Definition at line 238 of file graph_inset.h.

References `closed`.

```
238 { return closed; }
```

17.30.4.17 int util::GraphInset::numberOfLines () const [inline]

Returns the numbers of lines drawn.

Definition at line 253 of file graph_inset.h.

References `_numberOfLines`.

```
253 { return (int)_numberOfLines; }
```

17.30.4.18 void util::GraphInset::openGraph ()

Order the graph to open up.

Definition at line 299 of file graph_inset.cpp.

Referenced by `cleanData()`.

```
301      {
302          cerr << "Source color < 0 or > 255" << endl;
```

17.30.4.19 void util::GraphInset::readParms (const QString & section, util::Parms parms)

Read the parameters for the graph.

Parameters

section Section to read the parameters from

parms [util::Parms](#) object to read the parameters from

Definition at line 14 of file graph_inset.cpp.

References [_numberOfLines](#), [util::Parms::all\(\)](#), [GraphInset\(\)](#), [linesColor](#), [linesMinMaxView](#), [linesOffset](#), [linesWidth](#), [showAll](#), [showIndices](#), [windowPosition](#), and [windowSize](#).

```

16 {
17
18     using std::endl;
19     using std::cerr;
20
21     GraphInset::GraphInset(int nb_lines)
22         : _numberOfLines(nb_lines)
23         , initialized(false)
24         , closed(true)
25     {
26     }
27
28     void GraphInset::readParms(const QString& section, util::Parms parms)
29     {
30         parms(section, "Size", windowSize);
31         parms(section, "Position", windowPosition);
32
33         parms(section, "ShowAll", showAll);
34         if(!showAll)
35         {
36             parms(section, "Show", showIndices);
37         }
38         else
39         {
40             showIndices.clear();
41             for(size_t i = 0 ; i < _numberOfLines ; ++i)
42             {
43                 showIndices.push_back(i+1);
44             }
45         }
46         cerr << "Number of lines to be shown: " << showIndices.size() << endl;
47
48         parms.all(section, "LineColor", linesColor);
49         parms.all(section, "LineWidth", linesWidth);
50         parms.all(section, "LineMinMaxView", linesMinMaxView);
51         parms.all(section, "LineOffset", linesOffset);

```

17.30.4.20 void util::GraphInset::setNumberOfLines (size_t nb_lines)

Change the number of lines.

If the number of lines is actually changed, then the system is reset

Definition at line 53 of file graph_inset.cpp.

```
68 {
69     if(_numberOfLines != nb_lines)
70     {
71         _numberOfLines = nb_lines;
72         if(showAll)
73         {
74             showIndices.clear();
75             for(size_t i = 0 ; i < nb_lines ; ++i)
76             {
77                 showIndices.push_back(i);
```

17.30.4.21 void util::GraphInset::setTime (double *t*) [inline]

Kepp the graph informed of the current time in the system.

Definition at line 243 of file graph_inset.h.

References time.

```
243 { time = t; }
```

17.30.4.22 size_t util::GraphInset::size () const [inline]

Number of values stored.

Definition at line 280 of file graph_inset.h.

References data.

```
280 { return data.size(); }
```

17.30.4.23 bool util::GraphInset::writeCSV (const QString &*filename*)

Write the data in a .csv file.

Definition at line 304 of file graph_inset.cpp.

```
309 {
310     closed = true;
311 }
312
313 void GraphInset::openGraph()
314 {
315     closed = false;
316 }
317
318 bool GraphInset::writeCSV(const QString& filename)
319 {
```

```

320     QFile f(filename);
321     if(!f.open(QIODevice::WriteOnly))
322     {
323         cerr << "Error, cannot open file " << filename.toStdString() << " for writi
ng" << endl;
324         return false;
325     }
326     QTextStream ts(&f);
327     ts << "Time";
328     for(size_t i = 0 ; i < _numberOfLines ; ++i)
329     {
330         ts << ",Data" << i;
331     }
332     ts << ",Source Changed\r\n";

```

17.30.5 Member Data Documentation

17.30.5.1 `size_t util::GraphInset::_numberOfLines` [protected]

Number of lines drawn by the graph.

Definition at line 299 of file `graph_inset.h`.

Referenced by `addData()`, `numberOfLines()`, and `readParms()`.

17.30.5.2 `bool util::GraphInset::autoOpenClose`

Automatically open/close the graph.

Graph is opened as soon as some data is added to it.

Graph is closed when the data set is empty.

Definition at line 420 of file `graph_inset.h`.

Referenced by `cleanData()`.

17.30.5.3 `int util::GraphInset::backColor`

Color of the back of the graph.

If negative, the background will be transparent

Definition at line 328 of file `graph_inset.h`.

17.30.5.4 `double util::GraphInset::backgroundOffset`

Offset to draw the background of the graph.

Definition at line 386 of file `graph_inset.h`.

17.30.5.5 int util::GraphInset::changeSourceColor

Color of the line used to indicate a data change (typically if the data correspond to another cell).

Definition at line 375 of file graph_inset.h.

17.30.5.6 double util::GraphInset::changeSourceOffset

Offset to draw the data source change line.

Definition at line 396 of file graph_inset.h.

Referenced by drawWithName().

17.30.5.7 int util::GraphInset::changeSourceWidth

Width of the line used to indicate a data change (typically if the data correspond to another cell).

Definition at line 381 of file graph_inset.h.

17.30.5.8 bool util::GraphInset::closed [protected]

State if the graph window should be closed.

Definition at line 309 of file graph_inset.h.

Referenced by graphClosed().

17.30.5.9 int util::GraphInset::contourColor

Color of the contour.

Definition at line 348 of file graph_inset.h.

17.30.5.10 double util::GraphInset::contourOffset

Offset to draw the contour of the graph.

Definition at line 391 of file graph_inset.h.

17.30.5.11 int util::GraphInset::contourWidth

Width of the line used for the contour.

Definition at line 353 of file graph_inset.h.

17.30.5.12 `std::list<Data> util::GraphInset::data` `[protected]`

[Data](#) to be draw.

Definition at line 314 of file `graph_inset.h`.

Referenced by `cleanData()`, and `size()`.

17.30.5.13 `bool util::GraphInset::initialized` `[protected]`

Check the system is correctly initialized.

Definition at line 304 of file `graph_inset.h`.

17.30.5.14 `bool util::GraphInset::keepAllData`

Order the object to keep all the data.

Definition at line 411 of file `graph_inset.h`.

17.30.5.15 `std::vector<int> util::GraphInset::linesColor`

[Color](#) of the lines.

Definition at line 333 of file `graph_inset.h`.

Referenced by `readParms()`.

17.30.5.16 `std::vector<util::Vector<2, double> > util::GraphInset::linesMinMaxView`

Extrema value shown for each line.

Definition at line 343 of file `graph_inset.h`.

Referenced by `readParms()`.

17.30.5.17 `std::vector<double> util::GraphInset::linesOffset`

Offset to draw the lines.

Definition at line 401 of file `graph_inset.h`.

Referenced by `readParms()`.

17.30.5.18 `std::vector<double> util::GraphInset::linesWidth`

Width of the lines.

Definition at line 338 of file `graph_inset.h`.

Referenced by `readParms()`.

17.30.5.19 bool util::GraphInset::showAll

If true, show all the data.

Definition at line 425 of file graph_inset.h.

Referenced by readParms().

17.30.5.20 std::vector<size_t> util::GraphInset::showIndices

If not showing all data, contains the indices to show.

Definition at line 430 of file graph_inset.h.

Referenced by readParms().

17.30.5.21 double util::GraphInset::time

Time of the system.

Definition at line 406 of file graph_inset.h.

Referenced by cleanData(), drawWithName(), and setTime().

17.30.5.22 double util::GraphInset::timeSpan

Time span to keep the data.

Definition at line 321 of file graph_inset.h.

17.30.5.23 util::Vector<2,int> util::GraphInset::windowPosition

Position of the top-left corner of the graph window in pixels.

(0,0) is the top-left corner of the screen.

Negative values are considered distance from the bottom-right corner. Positives and negatives values can be mixed (i.e. (-100,30) = 100 pixel from the right, 30 pixels from the top)

Definition at line 364 of file graph_inset.h.

Referenced by readParms().

17.30.5.24 util::Vector<2,size_t> util::GraphInset::windowSize

Size of the graph window in pixels.

Definition at line 369 of file graph_inset.h.

Referenced by readParms().

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

- [vvelib/util/graph_inset.h](#)
- [vvelib/util/graph_inset.cpp](#)

17.31 `complex_factory::HexFiller< Complex, Model >` Struct Template Reference

For internal use only!

```
#include <complex_grid.h>
```

Public Member Functions

- **HexFiller** (const [Point3d](#) &bl, const [Point3d](#) &u, const [Point3d](#) &v, Complex &T, [Model](#) &model)
- **IMPORT_COMPLEX_TYPES** (Complex)
- void **initGridVertex** (size_t i, size_t j, const cell &v, cell_graph &)

Public Attributes

- [Point3d](#) **bottom_left**
- [Point3d](#) **delta_odd**
- [Model](#) & **model**
- [Point3d](#) **shift_right**
- [Point3d](#) **shift_up**
- Complex & **T**

17.31.1 Detailed Description

```
template<typename Complex, class Model> struct complex_factory::HexFiller<
Complex, Model >
```

For internal use only! Compute the position of each point

Definition at line 236 of file `complex_grid.h`.

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

- `vvelib/factory/complex_grid.h`

17.32 `vvcomplex::InModelDivisionParam` Class Reference

Class used to define the division parameters from within the model class.

```
#include <algorithms/complex.h>
```

17.32.1 Detailed Description

Class used to define the division parameters from within the model class. Use this class parameters when dividing a cell and the following method will be used in your model:

```
DivisionData<VertexContent> divisionParameters(const vertex& c, vvgraph& S, const vvcomplex::InModelDivisionParam& param);
```

A typical use will be to inherit this class to put parameters:

```
struct MyDivisionParams : public vvcomplex::InModelDivisionParam
{
    int divisionType;
}
```

And use this to divide according to the parameters:

```
vvcomplex::DivisionData<VertexContent> divisionParameters(const vertex& c,
vvgraph& S, const vvcomplex::InModelDivisionParam& p)
{
    DivisionData<VertexContent> result;
    MyDivisionParams& param = dynamic_cast<const InModelDivisionParam&>(p);
    if(param.divisionType == 0)
    {
        // First type of division
    }
    else
    {
        // Other type of division
    }
    return result;
}
```

Definition at line 1726 of file `complex.h`.

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

- `vvelib/algorithms/complex.h`

17.33 algorithms::Insert< vvgraph, do_checks > Class Template Reference

[Insert](#) a new vertex on an edge.

```
#include <algorithms/insert.h>
```

17.33.1 Detailed Description

```
template<class vvgraph, bool do_checks = true> class algorithms::Insert< vvgraph, do_checks >
```

[Insert](#) a new vertex on an edge.

Parameters

- a* an existing vertex
- b* an existing vertex
- S* the vv graph to edit

This function creates a new vertex and inserts it between two existing vertices by replacing it into the existing neighbourhoods. If the vertices have no relation or an asymmetric relation, a warning is printed to stderr and an empty vertex is returned.

Example:

```
vvgraph S;
vertex v1, v2;
S.insert(v1);
S.insert(v2);
S.insertEdge(v1,v2);
S.insertEdge(v2,v1);
Insert<vvgraph> insert;
vertex v = insert(v1, v2, S);
```

The pre-condition is that edges (v1,v2) and (v2,v1) exist.

The post-condition is that v replaced v2 in the neighborhood of v1 and v1 in the neighborhood of v2, keeping the double connections.

Definition at line 50 of file insert.h.

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

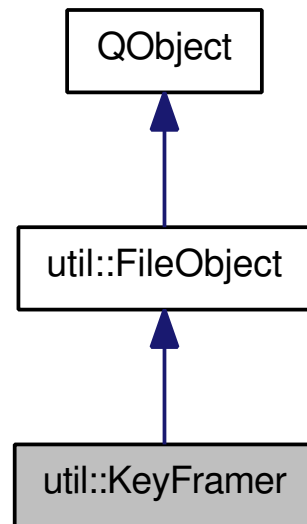
- [vvelib/algorithms/insert.h](#)

17.34 util::KeyFramer Class Reference

Class describing the evolution of a bspline surface through time.

```
#include <util/key_framer.h>
```

Inheritance diagram for util::KeyFramer:



Public Member Functions

- void **AddFrame** ()
- void **AdvanceTime** (double time_increment)
- void **DeleteFrame** ()
- void **DisplayFrame** (unsigned int frame)
- void **DisplayTime** (double time)
- void **DisplayTimeline** ()
- int **FindFrame** (double time)
- double **FindTime** (unsigned int frame)
- unsigned int **GetFrame** ()
- BSplineSurface **GetSurfaceFrame** ()
- BSplineSurface **GetSurfaceFrame** (unsigned int frame)
- BSplineSurface **GetSurfaceTime** ()
- BSplineSurface **GetSurfaceTime** (double time)
- double **GetTime** ()
- **KeyFramer** (const [KeyFramer](#) &)
- bool **Load** (const char *fname)
- void **LoadSurface** (const char *fname)
- void **MoveSelect** (Point3d pos)
- void [reread](#) ()

Method to be implemented in subclasses to reread the file.

- bool **Save** (const char *fname)
- void **SelectClosest** (Point3d pos)
- void **SetBoundingBox** (Point2d min, Point2d max)
- void **SetFrame** (unsigned int frame)
- void **SetParent** (QGLWidget *parent)
- void **SetPrecision** (int u, int v)
- void **SetScale** (double scale)
- void **SetTime** (Point3d pos)
- void **SetTime** (double time)

17.34.1 Detailed Description

Class describing the evolution of a bspline surface through time. The description consists in a list of 'BSplineSurface' objects at given time. Intermediate time are computed using linear interpolation between key frames.

Definition at line 31 of file key_framer.h.

17.34.2 Member Function Documentation

17.34.2.1 void util::KeyFramer::reread () [inline, virtual]

Method to be implemented in subclasses to reread the file.

Implements [util::FileObject](#).

Definition at line 42 of file key_framer.h.

References [util::FileObject::getFilename\(\)](#).

```

43     {
44         std::string fn = getFilename();
45         Load(fn.c_str());
46     }
```

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

- vvelib/util/key_framer.h
- vvelib/util/key_framer.cpp

17.35 util::Materials::Material Struct Reference

The material data structure.

```
#include <materials.h>
```

Public Attributes

- GLfloat **ambient** [4]
- GLfloat **diffuse** [4]
- GLfloat **emission** [4]
- bool **isDefault**
- GLfloat **shiny**
- GLfloat **specular** [4]
- GLfloat **transparency**

17.35.1 Detailed Description

The material data structure.

Definition at line 30 of file materials.h.

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

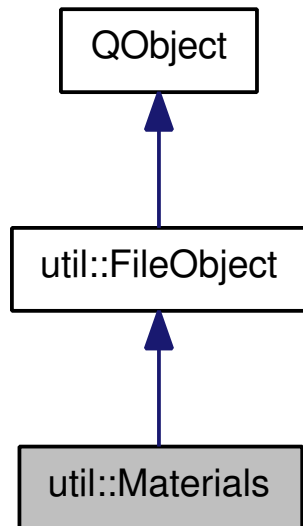
- vvelib/util/[materials.h](#)

17.36 util::Materials Class Reference

A utility class for materials.

```
#include <util/materials.h>
```

Inheritance diagram for util::Materials:



Classes

- struct [Material](#)
The material data structure.

Public Member Functions

- void [blend](#) (unsigned int ind1, unsigned int ind2, float t)
Returns a material structure that is a weighted average of two materials.
- const std::string & [getFilename](#) () const
- const [Material](#) & [getMaterial](#) (unsigned int index)
A call to get the material structure for a particular index.
- [Materials](#) (std::string filename)
Constructor.
- void [reread](#) ()
Reread the set material file.

- void `useMaterial` (unsigned int index)

A call to use a particular material.

17.36.1 Detailed Description

A utility class for materials. This class provides an interface for VLAB material files and their use for OpenGL. The material files can contain up to 255 materials. The class allows access to these by an index. If a material index is called that is not in the file, a default material is provided that uses the default values for OpenGL 1.2.

Definition at line 27 of file materials.h.

17.36.2 Constructor & Destructor Documentation

17.36.2.1 `util::Materials::Materials (std::string filename)`

Constructor.

Parameters

filename The VLAB material file to read.

Definition at line 29 of file materials.cpp.

References `reread()`.

```
30 : FileObject(filename)
31 {
32     for(int i = 0 ; i < 256 ; ++i)
33     {
34         mats[i].isDefault = true;
35     }
36     reread();
37 }
```

17.36.3 Member Function Documentation

17.36.3.1 `void util::Materials::blend (unsigned int ind1, unsigned int ind2, float t)`

Returns a material structure that is a weighted average of two materials.

Parameters

ind1 The first material index.

ind2 The second material index.

t The weight for the first material. If *t* is clamped to the range [0, 1]. The second material is given the weight (1.0 - *t*).

Definition at line 167 of file materials.cpp.

```

167                                     {
168     Material m;
169
170     if (t < 0.0) t = 0.0;
171     else if (t > 1.0) t = 1.0;
172
173     float t1 = 1.0 - t;
174
175     m.ambient[0] = t * mats[ind1].ambient[0] + t1 * mats[ind2].ambient[0];
176     m.ambient[1] = t * mats[ind1].ambient[1] + t1 * mats[ind2].ambient[1];
177     m.ambient[2] = t * mats[ind1].ambient[2] + t1 * mats[ind2].ambient[2];
178     m.ambient[3] = t * mats[ind1].ambient[3] + t1 * mats[ind2].ambient[3];
179
180     m.diffuse[0] = t * mats[ind1].diffuse[0] + t1 * mats[ind2].diffuse[0];
181     m.diffuse[1] = t * mats[ind1].diffuse[1] + t1 * mats[ind2].diffuse[1];
182     m.diffuse[2] = t * mats[ind1].diffuse[2] + t1 * mats[ind2].diffuse[2];
183     m.diffuse[3] = t * mats[ind1].diffuse[3] + t1 * mats[ind2].diffuse[3];
184
185     m.specular[0] = t * mats[ind1].specular[0] + t1 * mats[ind2].specular[0];
186     m.specular[1] = t * mats[ind1].specular[1] + t1 * mats[ind2].specular[1];
187     m.specular[2] = t * mats[ind1].specular[2] + t1 * mats[ind2].specular[2];
188     m.specular[3] = t * mats[ind1].specular[3] + t1 * mats[ind2].specular[3];
189
190     m.emission[0] = t * mats[ind1].emission[0] + t1 * mats[ind2].emission[0];
191     m.emission[1] = t * mats[ind1].emission[1] + t1 * mats[ind2].emission[1];
192     m.emission[2] = t * mats[ind1].emission[2] + t1 * mats[ind2].emission[2];
193     m.emission[3] = t * mats[ind1].emission[3] + t1 * mats[ind2].emission[3];
194
195     m.shiny = t * mats[ind1].shiny + t1 * mats[ind2].shiny;
196
197     glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, m.ambient);
198     REPORT_GL_ERROR("glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, m.ambient);");
199     glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, m.diffuse);
200     REPORT_GL_ERROR("glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, m.diffuse);");
201     glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, m.emission);
202     REPORT_GL_ERROR("glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, m.emission);");
203     glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, m.specular);
204     REPORT_GL_ERROR("glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, m.specular);");
205     glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, m.shiny);
206     REPORT_GL_ERROR("glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, m.shiny);");
207 }
```

17.36.3.2 const util::Materials::Material & util::Materials::getMaterial (unsigned int *index*)

A call to get the material structure for a particular index.

Parameters

index The material index.

Definition at line 155 of file materials.cpp.

```

155                                     {
156     if (index > 255) index = 255;
```

```

157     return mats[index];
158 }

```

17.36.3.3 void util::Materials::reread() [virtual]

Reread the set material file.

Implements [util::FileObject](#).

Definition at line 40 of file materials.cpp.

Referenced by Materials().

```

40     {
41     unsigned int index = 0;
42     std::ifstream in(filename.c_str(), std::ios::binary);
43
44     while (!in.eof() && in.good() && in) {
45         unsigned char mat[15];
46
47         for (int i = 0; i < 15; i++) {
48             mat[i] = in.get();
49         }
50
51         index = mat[0];
52
53         mats[index].isDefault = false;
54
55         mats[index].transparency = float(mat[1]) / 255.0;
56
57         mats[index].ambient[0] = float(mat[2]) / 255.0;
58         if (mats[index].ambient[0] < 0.0 || mats[index].ambient[0] > 1.0) mats[index]
.ambient[0] = 0.2f;
59         mats[index].ambient[1] = float(mat[3]) / 255.0;
60         if (mats[index].ambient[1] < 0.0 || mats[index].ambient[1] > 1.0) mats[index]
.ambient[1] = 0.2f;
61         mats[index].ambient[2] = float(mat[4]) / 255.0;
62         if (mats[index].ambient[2] < 0.0 || mats[index].ambient[2] > 1.0) mats[index]
.ambient[2] = 0.2f;
63         mats[index].ambient[3] = 1.0f - mats[index].transparency;
64
65         mats[index].diffuse[0] = float(mat[5]) / 255.0;
66         if (mats[index].diffuse[0] < 0.0 || mats[index].diffuse[0] > 1.0) mats[index]
.diffuse[0] = 0.8f;
67         mats[index].diffuse[1] = float(mat[6]) / 255.0;
68         if (mats[index].diffuse[1] < 0.0 || mats[index].diffuse[1] > 1.0) mats[index]
.diffuse[1] = 0.8f;
69         mats[index].diffuse[2] = float(mat[7]) / 255.0;
70         if (mats[index].diffuse[2] < 0.0 || mats[index].diffuse[2] > 1.0) mats[index]
.diffuse[2] = 0.8f;
71         mats[index].diffuse[3] = 1.0f - mats[index].transparency;
72
73         mats[index].emission[0] = float(mat[8]) / 255.0;
74         if (mats[index].emission[0] < 0.0 || mats[index].emission[0] > 1.0) mats[inde
x].emission[0] = 0.0f;
75         mats[index].emission[1] = float(mat[9]) / 255.0;
76         if (mats[index].emission[1] < 0.0 || mats[index].emission[1] > 1.0) mats[inde
x].emission[1] = 0.0f;
77         mats[index].emission[2] = float(mat[10]) / 255.0;
78         if (mats[index].emission[2] < 0.0 || mats[index].emission[2] > 1.0) mats[inde

```

```

    x].emission[2] = 0.0f;
79     mats[index].emission[3] = 1.0f - mats[index].transparency;
80
81     mats[index].specular[0] = float(mat[11]) / 255.0;
82     if (mats[index].specular[0] < 0.0 || mats[index].specular[0] > 1.0) mats[inde
x].specular[0] = 0.0f;
83     mats[index].specular[1] = float(mat[12]) / 255.0;
84     if (mats[index].specular[1] < 0.0 || mats[index].specular[1] > 1.0) mats[inde
x].specular[1] = 0.0f;
85     mats[index].specular[2] = float(mat[13]) / 255.0;
86     if (mats[index].specular[2] < 0.0 || mats[index].specular[2] > 1.0) mats[inde
x].specular[2] = 0.0f;
87     mats[index].specular[3] = 1.0f - mats[index].transparency;
88
89     mats[index].shiny = float(mat[14]) / 255.0; if (mats[index].shiny < 0.0) mats
[index].shiny = 0.0f;
90 }
91
92 for (index = 0; index < 256; index++) {
93     if(mats[index].isDefault)
94     {
95         mats[index].ambient[0] = 0.2f;
96         mats[index].ambient[1] = 0.2f;
97         mats[index].ambient[2] = 0.2f;
98         mats[index].ambient[3] = 1.0f;
99
100        mats[index].diffuse[0] = 0.8f;
101        mats[index].diffuse[1] = 0.8f;
102        mats[index].diffuse[2] = 0.8f;
103        mats[index].diffuse[3] = 1.0f;
104
105        mats[index].emission[0] = 0.0f;
106        mats[index].emission[1] = 0.0f;
107        mats[index].emission[2] = 0.0f;
108        mats[index].emission[3] = 1.0f;
109
110        mats[index].specular[0] = 0.0f;
111        mats[index].specular[1] = 0.0f;
112        mats[index].specular[2] = 0.0f;
113        mats[index].specular[3] = 1.0f;
114
115        mats[index].shiny = 0.0f;
116        mats[index].transparency = 0.0f;
117    }
118 }
119 }
```

17.36.3.4 void util::Materials::useMaterial (unsigned int *index*)

A call to use a particular material.

Parameters

index The material index.

Definition at line 124 of file materials.cpp.

```
124                                     {
```

```
125     if (index > 255) index = 255;
126
127     // glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mats[index].ambient);
128     // glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mats[index].diffuse);
129     // glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, mats[index].emission);
130     // glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mats[index].specular);
131     // glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, mats[index].shiny);
132     glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);
133     REPORT_GL_ERROR("glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);");
134     glColor4fv(mats[index].diffuse);
135     REPORT_GL_ERROR("glColor4fv(mats[index].diffuse);");
136     glColorMaterial(GL_FRONT_AND_BACK, GL_EMISSION);
137     REPORT_GL_ERROR("glColorMaterial(GL_FRONT_AND_BACK, GL_EMISSION);");
138     glColor4fv(mats[index].emission);
139     REPORT_GL_ERROR("glColor4fv(mats[index].emission);");
140     glColorMaterial(GL_FRONT_AND_BACK, GL_SPECULAR);
141     REPORT_GL_ERROR("glColorMaterial(GL_FRONT_AND_BACK, GL_SPECULAR);");
142     glColor4fv(mats[index].specular);
143     REPORT_GL_ERROR("glColor4fv(mats[index].specular);");
144     glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, mats[index].shiny);
145     REPORT_GL_ERROR(QString("glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, %1);").arg(
146         mats[index].shiny));
146     glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT);
147     REPORT_GL_ERROR("glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT);");
148     glColor4fv(mats[index].ambient);
149     REPORT_GL_ERROR("glColor4fv(mats[index].ambient);");
150 }
```

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

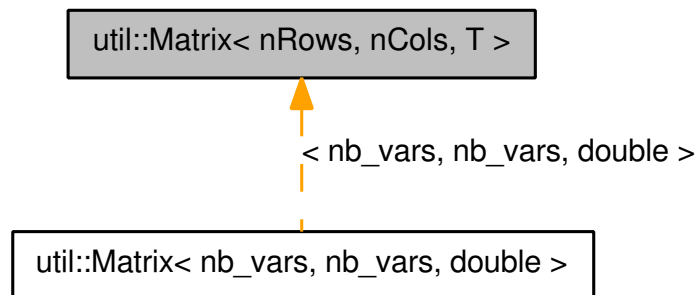
- [vvelib/util/materials.h](#)
- [vvelib/util/materials.cpp](#)

17.37 util::Matrix< nRows, nCols, T > Class Template Reference

Class representing a fixed-size matrix.

```
#include <util/matrix.h>
```

Inheritance diagram for util::Matrix< nRows, nCols, T >:



Public Types

- `typedef const T * const_iterator`
- `typedef const T * const_pointer_type`
- `typedef const T & const_reference_type`
- `typedef T * iterator`
- `typedef T * pointer_type`
- `typedef T & reference_type`
- `typedef T value_type`

Public Member Functions

- `const T * c_data () const`
Returns a constant raw pointer on the data.
- `T * data ()`
Returns a raw pointer on the data.
- `void fillArray (T *array, bool row_first=true)`
- `Matrix (const T &value)`
Create a diagonal matrix.
- `Matrix (const T *values, bool c_style=true)`
Fill in the matrix with the values.

- `template<typename T1 >`
`Matrix` (const `Vector`< nCols, T1 > *vecs)
Fill the matrix with the array of vectors.
- `template<size_t nRows1, size_t nCols1, typename T1 >`
`Matrix` (const `Matrix`< nRows1, nCols1, T1 > &mat)
Copy a matrix.
- `template<typename T1 >`
`Matrix` (const `Matrix`< nRows, nCols, T1 > &mat)
Copy a matrix.
- `Matrix` (const `Matrix` &mat)
Copy constructor.
- `Matrix` (void)
Create a matrix filled with 0s.
- `bool operator!=` (const `Matrix` &mat) const
- `T operator()` (size_t i, size_t j) const
Return the value at row i, column j.
- `T & operator()` (size_t i, size_t j)
Return the value at row i, column j.
- `Vector`< nRows, T > `operator*` (const `Vector`< nCols, T > &vec) const
*Matrix*Vector.*
- `Matrix operator*` (const T &scalar) const
Matrix-scalar multiplication.
- `Matrix & operator*=` (const `Matrix` &mat)
- `Matrix & operator*=` (const T &scalar)
- `Matrix operator+` (const `Matrix` &mat) const
Matrix addition.
- `Matrix & operator+=` (const `Matrix` &mat)
- `Matrix operator-` (const `Matrix` &mat) const
- `Matrix operator-` (void) const
Matrix subtraction.
- `Matrix & operator-=` (const `Matrix` &mat)
- `Matrix operator/` (const T &scalar) const
Matrix-scalar division.
- `Matrix & operator/=` (const T &scalar)

- **Matrix** & **operator=** (const T &value)
Set the matrix to a diagonal matrix.
- **Matrix** & **operator=** (const **Matrix** &mat)
- bool **operator==** (const **Matrix** &mat) const
- **Vector**< nCols, T > **operator[]** (size_t idx) const
Returns the nth row.
- **Vector**< nCols, T > & **operator[]** (size_t idx)
Returns the nth row.
- **Matrix**< nCols, nRows, T > **operator~** ()
Transpose the matrix.
- T **trace** () const
Trace of the matrix.
- **Matrix** & **zero** (void)
Set the matrix to all zero.

Static Public Member Functions

- static **Matrix** **identity** ()
Returns an identity matrix.
- static size_t **nbColumns** ()
Returns the number of columns of the matrix.
- static size_t **nbRows** ()
Returns the number of rows of the matrix.
- static **Matrix**< 4, 4, T > **rotation** (const **Vector**< 4, T > &direction, T angle)
Creates the 4x4 matrix corresponding to a rotation.
- static **Matrix**< 3, 3, T > **rotation** (const **Vector**< 3, T > &direction, T angle)
Creates the 3x3 matrix corresponding to a rotation.
- static **Vector**< 2, size_t > **size** ()
Returns the size of the matrix.

Friends

- **Matrix operator*** (const T &scalar, const **Matrix** &mat)
Matrix-scalar multiplication.
- std::ostream & **operator<<** (std::ostream &out, const **Matrix** &mat)
- **QTextStream** & **operator<<** (**QTextStream** &out, const **Matrix** &mat)
- std::istream & **operator>>** (std::istream &in, **Matrix** &mat)
- **QTextStream** & **operator>>** (**QTextStream** &in, **Matrix** &mat)

Related Functions

(Note that these are not member functions.)

- template<size_t nRows, typename T >
T **cofactor** (const **Matrix**< nRows, nRows, T > &mat, size_t i, size_t j)
Returns the cofactor of the matrix for position (i,j).
- template<size_t nRows, typename T >
T **det** (const **Matrix**< nRows, nRows, T > &mat)
Determinant of the matrix.
- template<typename T >
T **det** (const **Matrix**< 3, 3, T > &mat)
Determinant of the matrix.
- template<typename T >
T **det** (const **Matrix**< 2, 2, T > &mat)
Determinant of the matrix.
- template<typename T >
T **det** (const **Matrix**< 1, 1, T > &mat)
Determinant of the matrix.
- template<size_t nRows, typename T >
Matrix< nRows, nRows, T > **inverse** (const **Matrix**< nRows, nRows, T > &mat)
Inverse the matrix.
- template<typename T >
Matrix< 3, 3, T > **inverse** (const **Matrix**< 3, 3, T > &mat)
Inverse the matrix.
- template<typename T >
Matrix< 2, 2, T > **inverse** (const **Matrix**< 2, 2, T > &mat)
Inverse the matrix.

- template<typename T >
Matrix< 1, 1, T > **inverse** (const **Matrix**< 1, 1, T > &mat)
Inverse the matrix.
- template<size_t nRows, size_t nCols, typename T, typename T1, typename T2 >
Matrix< nRows, nCols, T > **map** (const T &(*fct)(const T1 &, const T2 &), const **Matrix**< nRows, nCols, T1 > &m1, const **Matrix**< nRows, nCols, T2 > &m2)
Apply a binary function to each element of the matrix.
- template<size_t nRows, size_t nCols, typename T, typename T1, typename T2 >
Matrix< nRows, nCols, T > **map** (T(*fct)(const T1 &, const T2 &), const **Matrix**< nRows, nCols, T1 > &m1, const **Matrix**< nRows, nCols, T2 > &m2)
Apply a binary function to each element of the matrix.
- template<size_t nRows, size_t nCols, typename T, typename T1, typename T2 >
Matrix< nRows, nCols, T > **map** (T(*fct)(T1, T2), const **Matrix**< nRows, nCols, T1 > &m1, const **Matrix**< nRows, nCols, T2 > &m2)
Apply a binary function to each element of the matrix.
- template<size_t nRows, size_t nCols, typename T >
Matrix< nRows, nCols, T > **map** (const T &(*fct)(const T &, const T &), const **Matrix**< nRows, nCols, T > &m1, const **Matrix**< nRows, nCols, T > &m2)
Apply a binary function to each element of the matrix.
- template<size_t nRows, size_t nCols, typename T >
Matrix< nRows, nCols, T > **map** (T(*fct)(const T &, const T &), const **Matrix**< nRows, nCols, T > &m1, const **Matrix**< nRows, nCols, T > &m2)
Apply a binary function to each element of the matrix.
- template<size_t nRows, size_t nCols, typename T >
Matrix< nRows, nCols, T > **map** (T(*fct)(T, T), const **Matrix**< nRows, nCols, T > &m1, const **Matrix**< nRows, nCols, T > &m2)
Apply a binary function to each element of the matrix.
- template<size_t nRows, size_t nCols, typename T, typename T1 >
Matrix< nRows, nCols, T > **map** (T(*fct)(const T1 &), const **Matrix**< nRows, nCols, T1 > &m)
Apply a unary function to each element of the matrix.
- template<size_t nRows, size_t nCols, typename T, typename T1 >
Matrix< nRows, nCols, T > **map** (T(*fct)(T1), const **Matrix**< nRows, nCols, T1 > &m)
Apply a unary function to each element of the matrix.
- template<size_t nRows, size_t nCols, typename T, typename T1 >
Matrix< nRows, nCols, T > **map** (const T &(*fct)(const T1 &), const **Matrix**< nRows, nCols, T1 > &m)

Apply a unary function to each element of the matrix.

- `template<size_t nRows, size_t nCols, typename T >`
`Matrix< nRows, nCols, T > map (T(*fct)(const T &), const Matrix< nRows,`
`nCols, T > &m)`

Apply a unary function to each element of the matrix.

- `template<size_t nRows, size_t nCols, typename T >`
`Matrix< nRows, nCols, T > map (T(*fct)(T), const Matrix< nRows, nCols, T`
`> &m)`

Apply a unary function to each element of the matrix.

- `template<size_t nRows, size_t nCols, typename T >`
`Matrix< nRows, nCols, T > map (const T &(*fct)(const T &), const Matrix<`
`nRows, nCols, T > &m)`

Apply a unary function to each element of the matrix.

- `template<size_t nRows, size_t nCols, typename T >`
`T norm (const Matrix< nRows, nCols, T > &mat)`

Return the norm of the matrix.

- `template<size_t nRows, size_t nCols, typename T >`
`T normsq (const Matrix< nRows, nCols, T > &mat)`

Return the square norm of the matrix.

- `template<size_t nRows, size_t nSize, size_t nCols, typename T >`
`Matrix< nRows, nCols, T > operator* (const Matrix< nRows, nSize, T >`
`&mat1, const Matrix< nSize, nCols, T > &mat2)`

Matrix multiplication.

- `template<size_t nRows, size_t nCols, typename T >`
`Matrix< nCols, nRows, T > transpose (const Matrix< nRows, nCols, T >`
`&mat)`

Transpose a matrix.

17.37.1 Detailed Description

`template<size_t nRows, size_t nCols, typename T = double> class util::Matrix<`
`nRows, nCols, T >`

Class representing a fixed-size matrix. This class is optimized for small-sized matrix (3x3 or 4x4).

Definition at line 25 of file matrix.h.

17.37.2 Constructor & Destructor Documentation

17.37.2.1 `template<size_t nRows, size_t nCols, typename T = double>
util::Matrix< nRows, nCols, T >::Matrix (void) [inline]`

Create a matrix filled with 0s.

Definition at line 42 of file matrix.h.

```

43     {
44         for( size_t i = 0 ; i < nRows ; i++ )
45             for( size_t j = 0 ; j < nCols ; j++ )
46                 rows[ i ][ j ] = 0;
47     }
```

17.37.2.2 `template<size_t nRows, size_t nCols, typename T = double>
util::Matrix< nRows, nCols, T >::Matrix (const Matrix< nRows,
nCols, T > &mat) [inline]`

Copy constructor.

Definition at line 52 of file matrix.h.

```

53     {
54         for(size_t i = 0 ; i < nRows ; i++)
55             rows[i] = mat.rows[i];
56     }
```

17.37.2.3 `template<size_t nRows, size_t nCols, typename T = double>
template<typename T1 > util::Matrix< nRows, nCols, T
>::Matrix (const Matrix< nRows, nCols, T1 > &mat) [inline,
explicit]`

Copy a matrix.

Can be used to cast a matrix onto another type.

Definition at line 64 of file matrix.h.

```

65     {
66         for(size_t i = 0 ; i < nRows ; i++)
67             rows[i] = mat.rows[i];
68     }
```

17.37.2.4 `template<size_t nRows, size_t nCols, typename T = double>
template<size_t nRows1, size_t nCols1, typename T1 > util::Matrix<
nRows, nCols, T >::Matrix (const Matrix< nRows1, nCols1, T1 > &
mat) [inline, explicit]`

Copy a matrix.

Can be used to copy a matrix with more or less dimensions.

If the input has more dimensions, then the extra ones are just ignored. If the input has less dimensions, 1s are inserted in the diagonal, and 0s outside

Definition at line 79 of file matrix.h.

```

80      {
81          int rowMax = nRowsl < nRows ? nRowsl : nRows;
82          int colMax = nColsl < nCols ? nColsl : nCols;
83          for(size_t i = 0 ; i < rowMax ; i++)
84          {
85              Vector<nCols,T>& c = rows[i];
86              const Vector<nColsl,T1>& c1 = mat[i];
87              for(size_t j = 0 ; j < colMax ; j++)
88                  c[j] = c1[j];
89              if(colMax < nCols)
90                  for(size_t j = colMax ; j < nCols ; ++j)
91                      c[j] = i==j ? T(1) : T(0);
92          }
93          if(rowMax < nRows)
94          {
95              for(size_t i = rowMax ; i < nRows ; ++i)
96              {
97                  Vector<nCols,T>& c = rows[i];
98                  for(size_t j = colMax ; j < nCols ; ++j)
99                      c[j] = i==j ? T(1) : T(0);
100              }
101          }
102      }

```

17.37.2.5 `template<size_t nRows, size_t nCols, typename T = double>
util::Matrix< nRows, nCols, T >::Matrix
(const Vector< nCols, T1 > * vecs) [inline]`

Fill the matrix with the array of vectors.

Parameters

vecs Array of nRows vectors.

Definition at line 115 of file matrix.h.

```

116      {
117          for(size_t i = 0 ; i < nRows ; i++)
118              rows[i] = vecs[i];
119      }

```

17.37.2.6 `template<size_t nRows, size_t nCols, typename T = double>
util::Matrix< nRows, nCols, T >::Matrix (const T * values, bool
c_style = true) [inline]`

Fill in the matrix with the values.

Parameters

values nRows*nCols array. If `c_style` is true, `values` is rows first (i.e. the first values correspond to the first row). Otherwise, values are columns first.

c_style Determine the ordering of values.

Definition at line 130 of file matrix.h.

```

131     {
132         if(c_style)
133         {
134             for(size_t i = 0 ; i < nRows ; i++)
135             {
136                 rows[i] = Vector<nCols,T>(values + (i*nCols));
137             }
138         }
139         else
140         {
141             for(size_t i = 0 ; i < nRows ; i++)
142                 for(size_t j = 0 ; j < nCols ; j++)
143                 {
144                     rows[i][j] = values[i+j*nRows];
145                 }
146         }
147     }

```

17.37.2.7 `template<size_t nRows, size_t nCols, typename T = double> util::Matrix< nRows, nCols, T >::Matrix (const T & value) [inline]`

Create a diagonal matrix.

Parameters

value Value placed on the diagonal.

Definition at line 154 of file matrix.h.

```

155     {
156         for(size_t i = 0 ; i < nRows ; i++)
157             for(size_t j = 0 ; j < nCols ; j++)
158                 rows[i][j] = (i==j)?value:0;
159     }

```

17.37.3 Member Function Documentation

17.37.3.1 `template<size_t nRows, size_t nCols, typename T = double> const T* util::Matrix< nRows, nCols, T >::c_data () const [inline]`

Returns a constant raw pointer on the data.

The data are organised on by rows. (i.e. in the same order than OpenGL)

Definition at line 183 of file matrix.h.

```
183 { return rows[0].c_data(); }
```

17.37.3.2 `template<size_t nRows, size_t nCols, typename T = double> T* util::Matrix<nRows, nCols, T>::data() [inline]`

Returns a raw pointer on the data.

The data are organised on by rows. (i.e. in the same order than OpenGL)

Definition at line 190 of file matrix.h.

```
190 { return rows[0].data(); }
```

17.37.3.3 `template<size_t nRows, size_t nCols, typename T = double> static Matrix util::Matrix<nRows, nCols, T>::identity() [inline, static]`

Returns an identity matrix.

Definition at line 410 of file matrix.h.

```
411 {
412     Matrix mat(1);
413     return mat;
414 }
```

17.37.3.4 `template<size_t nRows, size_t nCols, typename T = double> static size_t util::Matrix<nRows, nCols, T>::nbColumns() [inline, static]`

Returns the number of columns of the matrix.

Definition at line 176 of file matrix.h.

```
176 { return nCols; }
```

17.37.3.5 `template<size_t nRows, size_t nCols, typename T = double> static size_t util::Matrix<nRows, nCols, T>::nbRows() [inline, static]`

Returns the number of rows of the matrix.

Definition at line 171 of file matrix.h.

```
171 { return nRows; }
```

17.37.3.6 `template<size_t nRows, size_t nCols, typename T = double> T
util::Matrix< nRows, nCols, T >::operator() (size_t i, size_t j) const
[inline]`

Return the value at row *i*, column *j*.

Definition at line 402 of file matrix.h.

```
403     {
404         return rows[i][j];
405     }
```

17.37.3.7 `template<size_t nRows, size_t nCols, typename T = double> T&
util::Matrix< nRows, nCols, T >::operator() (size_t i, size_t j)
[inline]`

Return the value at row *i*, column *j*.

Definition at line 394 of file matrix.h.

```
395     {
396         return rows[i][j];
397     }
```

17.37.3.8 `template<size_t nRows, size_t nCols, typename T = double>
Vector<nRows,T> util::Matrix< nRows, nCols, T >::operator*
(const Vector< nCols, T > & vec) const [inline]`

Matrix*Vector.

Definition at line 273 of file matrix.h.

```
274     {
275         Vector<nRows,T> ans;
276         for(size_t i = 0 ; i < nRows ; ++i)
277         {
278             T value = 0;
279             for(size_t j = 0 ; j < nCols ; ++j)
280                 value += rows[i][j] * vec[j];
281             ans[i] = value;
282         }
283         return ans;
284     }
```

17.37.3.9 `template<size_t nRows, size_t nCols, typename T = double> Matrix
util::Matrix< nRows, nCols, T >::operator* (const T & scalar) const
[inline]`

Matrix-scalar multiplication.

Definition at line 234 of file matrix.h.

```

235     {
236         Matrix ans;
237
238         for(size_t i = 0 ; i < nRows ; i++)
239             ans[i] = rows[i] * scalar;
240
241         return ans;
242     }

```

17.37.3.10 `template<size_t nRows, size_t nCols, typename T = double> Matrix
util::Matrix< nRows, nCols, T >::operator+ (const Matrix< nRows,
nCols, T > &mat) const [inline]`

[Matrix](#) addition.

[Matrix](#) subtraction

Definition at line 208 of file matrix.h.

```

209     {
210         Matrix ans;
211
212         for(size_t i = 0 ; i < nRows ; i++)
213             ans.rows[i] = rows[i] + mat.rows[i];
214
218         return ans;
219     }

```

17.37.3.11 `template<size_t nRows, size_t nCols, typename T = double>
Matrix util::Matrix< nRows, nCols, T >::operator- (void) const
[inline]`

[Matrix](#) subtraction.

Definition at line 195 of file matrix.h.

```

196     {
197         Matrix ans;
198
199         for(size_t i = 0 ; i < nRows ; i++)
200             ans.rows[i] = -rows[i];
201
202         return ans;
203     }

```

17.37.3.12 `template<size_t nRows, size_t nCols, typename T = double> Matrix
util::Matrix< nRows, nCols, T >::operator/ (const T &scalar) const
[inline]`

Matrix-scalar division.

Definition at line 247 of file matrix.h.

```

248     {
249         Matrix ans;
250
251         for(size_t i = 0 ; i < nRows ; i++)
252             ans[i] = rows[i] / scalar;
253
254         return ans;
255     }

```

17.37.3.13 `template<size_t nRows, size_t nCols, typename T = double> Matrix& util::Matrix< nRows, nCols, T >::operator= (const T & value) [inline]`

Set the matrix to a diagonal matrix.

Parameters

value Value to put on the diagonal

Definition at line 432 of file matrix.h.

```

433     {
434         for( size_t i = 0 ; i < nRows ; ++i )
435         {
436             for( size_t j = 0 ; j < nCols ; ++j )
437             {
438                 if( i == j )
439                     rows[ i ][ j ] = value;
440                 else
441                     rows[ i ][ j ] = 0;
442             }
443         }
444         return *this;
445     }

```

17.37.3.14 `template<size_t nRows, size_t nCols, typename T = double> Vector<nCols,T> util::Matrix< nRows, nCols, T >::operator[] (size_t idx) const [inline]`

Returns the nth row.

Parameters

idx Index of the returned row

Definition at line 386 of file matrix.h.

```

387     {
388         return rows[idx];
389     }

```

17.37.3.15 `template<size_t nRows, size_t nCols, typename T = double>
Vector<nCols,T>& util::Matrix< nRows, nCols, T >::operator[]
(size_t idx) [inline]`

Returns the *nth* row.

Parameters

idx Index of the returned row

Definition at line 376 of file matrix.h.

```
377     {
378         return rows[idx];
379     }
```

17.37.3.16 `template<size_t nRows, size_t nCols, typename T = double>
Matrix<nCols,nRows,T> util::Matrix< nRows, nCols, T
>::operator~ () [inline]`

Transpose the matrix.

Definition at line 450 of file matrix.h.

```
451     {
452         Matrix<nCols,nRows,T> t;
453         for( size_t i = 0 ; i < nRows ; ++i )
454             for( size_t j = 0 ; j < nCols ; ++j )
455                 t[ i ][ j ] = rows[ j ][ i ];
456         return t;
457     }
```

17.37.3.17 `template<size_t nRows, size_t nCols, typename T = double> static
Matrix<4,4,T> util::Matrix< nRows, nCols, T >::rotation (const
Vector< 4, T > & direction, T angle) [inline, static]`

Creates the 4x4 matrix corresponding to a rotation.

Parameters

direction Axes of the rotation

angle Angle of the rotation

Definition at line 485 of file matrix.h.

```
486     {
487         T ca = std::cos( angle );
488         T sa = std::sin( angle );
489         Matrix<4,4,T> r;
490         double x = direction.x()/direction.t();
```

```

491     double y = direction.y()/direction.t();
492     double z = direction.z()/direction.t();
493     r[ 0 ].set( ca+(1-ca)*x*x,    (1-ca)*x*y-sa*z, (1-ca)*z*x+sa*y, 0 );
494     r[ 1 ].set( (1-ca)*y*x+sa*z, ca+(1-ca)*y*y,    (1-ca)*z*y-sa*x, 0 );
495     r[ 2 ].set( (1-ca)*x*z-sa*y, (1-ca)*y*z+sa*x, ca+(1-ca)*z*z, 0 );
496     r[ 3 ].set( 0, 0, 0, 1 );
497     return r;
498 }

```

17.37.3.18 `template<size_t nRows, size_t nCols, typename T = double> static
Matrix<3,3,T> util::Matrix< nRows, nCols, T >::rotation (const
Vector< 3, T > & direction, T angle) [inline, static]`

Creates the 3x3 matrix corresponding to a rotation.

Parameters

direction Axes of the rotation

angle Angle of the rotation

Definition at line 465 of file matrix.h.

```

466     {
467         T ca = std::cos( angle );
468         T sa = std::sin( angle );
469         Matrix<3,3,T> r;
470         double x = direction.x();
471         double y = direction.y();
472         double z = direction.z();
473         r[ 0 ].set( ca+(1-ca)*x*x,    (1-ca)*x*y-sa*z, (1-ca)*z*x+sa*y );
474         r[ 1 ].set( (1-ca)*y*x+sa*z, ca+(1-ca)*y*y,    (1-ca)*z*y-sa*x );
475         r[ 2 ].set( (1-ca)*x*z-sa*y, (1-ca)*y*z+sa*x, ca+(1-ca)*z*z );
476         return r;
477     }

```

17.37.3.19 `template<size_t nRows, size_t nCols, typename T = double>
static Vector<2,size_t> util::Matrix< nRows, nCols, T >::size ()
[inline, static]`

Returns the size of the matrix.

First element is number of rows, second is number of columns

Definition at line 166 of file matrix.h.

```

166 { return Vector<2,size_t>(nRows, nCols); }

```

17.37.3.20 `template<size_t nRows, size_t nCols, typename T = double> T
util::Matrix< nRows, nCols, T >::trace () const [inline]`

Trace of the matrix.

Definition at line 503 of file matrix.h.

```

504     {
505         T acc = 0;
506         for(size_t i = 0 ; i < std::min(nRows,nCols) ; ++i)
507         {
508             acc += rows[i][i];
509         }
510         return acc;
511     }

```

17.37.3.21 `template<size_t nRows, size_t nCols, typename T = double>` `Matrix& util::Matrix< nRows, nCols, T >::zero (void) [inline]`

Set the matrix to all zero.

Definition at line 419 of file matrix.h.

```

420     {
421         for(size_t i = 0 ; i < nRows ; i++)
422             for(size_t j = 0 ; j < nCols ; j++)
423                 rows[i][j] = 0.0;
424         return (*this);
425     }

```

17.37.4 Friends And Related Function Documentation

17.37.4.1 `template<size_t nRows, typename T > T cofactor (const Matrix<` `nRows, nRows, T > & mat, size_t i, size_t j) [related]`

Returns the cofactor of the matrix for position (i,j).

17.37.4.2 `template<size_t nRows, typename T > T det (const Matrix< nRows,` `nRows, T > & mat) [related]`

Determinant of the matrix.

Warning

the method used is $O(n^3)$ complexity !

17.37.4.3 `template<typename T > T det (const Matrix< 3, 3, T > & mat)` `[related]`

Determinant of the matrix.

17.37.4.4 `template<typename T > T det (const Matrix< 2, 2, T > & mat)` `[related]`

Determinant of the matrix.

17.37.4.5 `template<typename T > T det (const Matrix< 1, 1, T > & mat)` [related]

Determinant of the matrix.

17.37.4.6 `template<size_t nRows, typename T > Matrix< nRows, nRows, T >` `inverse (const Matrix< nRows, nRows, T > & mat)` [related]

Inverse the matrix.

Warning

This algorithm is sub-optimal

17.37.4.7 `template<typename T > Matrix< 3, 3, T > inverse (const Matrix< 3,` `3, T > & mat)` [related]

Inverse the matrix.

17.37.4.8 `template<typename T > Matrix< 2, 2, T > inverse (const Matrix< 2,` `2, T > & mat)` [related]

Inverse the matrix.

17.37.4.9 `template<typename T > Matrix< 1, 1, T > inverse (const Matrix< 1,` `1, T > & mat)` [related]

Inverse the matrix.

17.37.4.10 `template<size_t nRows, size_t nCols, typename T , typename T1` `, typename T2 > Matrix< nRows, nCols, T > map (const T` `&(*)(const T1 &, const T2 &).fct, const Matrix< nRows, nCols, T1` `> & m1, const Matrix< nRows, nCols, T2 > & m2)` [related]

Apply a binary function to each element of the matrix.

Definition at line 876 of file matrix.h.

```

877     {
878         Matrix<nRows,nCols,T> result;
879         for(size_t i = 0 ; i < nRows ; ++i)
880         {
881             const Vector<nCols,T1>& mrow1 = m1[i];
882             const Vector<nCols,T2>& mrow2 = m2[i];
883             Vector<nCols,T>& rrow = result[i];
884             for(size_t j = 0 ; j < nCols ; ++j)
885             {
886                 rrow[j] = (*fct)(mrow1[j], mrow2[j]);

```

```

887     }
888 }
889 return result;
890 }
```

17.37.4.11 `template<size_t nRows, size_t nCols, typename T , typename T1 ,
typename T2 > Matrix< nRows, nCols, T > map (T(*) (const T1 &
const T2 &) fct, const Matrix< nRows, nCols, T1 > & m1, const
Matrix< nRows, nCols, T2 > & m2)` [**related**]

Apply a binary function to each element of the matrix.

Definition at line 854 of file matrix.h.

```

855 {
856     Matrix<nRows,nCols,T> result;
857     for(size_t i = 0 ; i < nRows ; ++i)
858     {
859         const Vector<nCols,T1>& mrow1 = m1[i];
860         const Vector<nCols,T2>& mrow2 = m2[i];
861         Vector<nCols,T>& rrow = result[i];
862         for(size_t j = 0 ; j < nCols ; ++j)
863         {
864             rrow[j] = (*fct) (mrow1[j], mrow2[j]);
865         }
866     }
867     return result;
868 }
```

17.37.4.12 `template<size_t nRows, size_t nCols, typename T , typename T1 ,
typename T2 > Matrix< nRows, nCols, T > map (T(*) (T1, T2) fct,
const Matrix< nRows, nCols, T1 > & m1, const Matrix< nRows,
nCols, T2 > & m2)` [**related**]

Apply a binary function to each element of the matrix.

Definition at line 832 of file matrix.h.

```

833 {
834     Matrix<nRows,nCols,T> result;
835     for(size_t i = 0 ; i < nRows ; ++i)
836     {
837         const Vector<nCols,T1>& mrow1 = m1[i];
838         const Vector<nCols,T2>& mrow2 = m2[i];
839         Vector<nCols,T>& rrow = result[i];
840         for(size_t j = 0 ; j < nCols ; ++j)
841         {
842             rrow[j] = (*fct) (mrow1[j], mrow2[j]);
843         }
844     }
845     return result;
846 }
```

17.37.4.13 `template<size_t nRows, size_t nCols, typename T > Matrix< nRows, nCols, T > map (const T &(*)(const T &, const T &) fct, const Matrix< nRows, nCols, T > & m1, const Matrix< nRows, nCols, T > & m2)` [**related**]

Apply a binary function to each element of the matrix.

Definition at line 810 of file matrix.h.

```

811 {
812     Matrix<nRows,nCols,T> result;
813     for(size_t i = 0 ; i < nRows ; ++i)
814     {
815         const Vector<nCols,T>& mrow1 = m1[i];
816         const Vector<nCols,T>& mrow2 = m2[i];
817         Vector<nCols,T>& rrow = result[i];
818         for(size_t j = 0 ; j < nCols ; ++j)
819         {
820             rrow[j] = (*fct)(mrow1[j], mrow2[j]);
821         }
822     }
823     return result;
824 }
```

17.37.4.14 `template<size_t nRows, size_t nCols, typename T > Matrix< nRows, nCols, T > map (T(*)(const T &, const T &) fct, const Matrix< nRows, nCols, T > & m1, const Matrix< nRows, nCols, T > & m2)` [**related**]

Apply a binary function to each element of the matrix.

Definition at line 788 of file matrix.h.

```

789 {
790     Matrix<nRows,nCols,T> result;
791     for(size_t i = 0 ; i < nRows ; ++i)
792     {
793         const Vector<nCols,T>& mrow1 = m1[i];
794         const Vector<nCols,T>& mrow2 = m2[i];
795         Vector<nCols,T>& rrow = result[i];
796         for(size_t j = 0 ; j < nCols ; ++j)
797         {
798             rrow[j] = (*fct)(mrow1[j], mrow2[j]);
799         }
800     }
801     return result;
802 }
```

17.37.4.15 `template<size_t nRows, size_t nCols, typename T > Matrix< nRows, nCols, T > map (T(*)(T, T) fct, const Matrix< nRows, nCols, T > & m1, const Matrix< nRows, nCols, T > & m2)` [**related**]

Apply a binary function to each element of the matrix.

Definition at line 766 of file matrix.h.

```

767  {
768      Matrix<nRows,nCols,T> result;
769      for(size_t i = 0 ; i < nRows ; ++i)
770      {
771          const Vector<nCols,T>& mrow1 = m1[i];
772          const Vector<nCols,T>& mrow2 = m2[i];
773          Vector<nCols,T>& rrow = result[i];
774          for(size_t j = 0 ; j < nCols ; ++j)
775          {
776              rrow[j] = (*fct)(mrow1[j], mrow2[j]);
777          }
778      }
779      return result;
780  }
```

17.37.4.16 `template<size_t nRows, size_t nCols, typename T , typename T1 > Matrix< nRows, nCols, T > map (T(*) (const T1 &)) fct, const Matrix< nRows, nCols, T1 > & m)` [**related**]

Apply a unary function to each element of the matrix.

Definition at line 745 of file matrix.h.

```

746  {
747      Matrix<nRows,nCols,T> result;
748      for(size_t i = 0 ; i < nRows ; ++i)
749      {
750          const Vector<nCols,T1>& mrow = m[i];
751          Vector<nCols,T>& rrow = result[i];
752          for(size_t j = 0 ; j < nCols ; ++j)
753          {
754              rrow[j] = (*fct)(mrow[j]);
755          }
756      }
757      return result;
758  }
```

17.37.4.17 `template<size_t nRows, size_t nCols, typename T , typename T1 > Matrix< nRows, nCols, T > map (T(*) (T1)) fct, const Matrix< nRows, nCols, T1 > & m)` [**related**]

Apply a unary function to each element of the matrix.

Definition at line 723 of file matrix.h.

```

724  {
725      Matrix<nRows,nCols,T> result;
726      for(size_t i = 0 ; i < nRows ; ++i)
727      {
728          const Vector<nCols,T1>& mrow = m[i];
729          Vector<nCols,T>& rrow = result[i];
730          for(size_t j = 0 ; j < nCols ; ++j)
```

```

731     {
732         rrow[j] = (*fct)(mrow[j]);
733     }
734 }
735 return result;
736 }
```

17.37.4.18 `template<size_t nRows, size_t nCols, typename T , typename T1 >
Matrix< nRows, nCols, T > map (const T &*)(const T1 &) fct,
const Matrix< nRows, nCols, T1 > & m)` [**related**]

Apply a unary function to each element of the matrix.

Definition at line 702 of file matrix.h.

```

703 {
704     Matrix<nRows,nCols,T> result;
705     for(size_t i = 0 ; i < nRows ; ++i)
706     {
707         const Vector<nCols,T1>& mrow = m[i];
708         Vector<nCols,T>& rrow = result[i];
709         for(size_t j = 0 ; j < nCols ; ++j)
710         {
711             rrow[j] = (*fct)(mrow[j]);
712         }
713     }
714     return result;
715 }
```

17.37.4.19 `template<size_t nRows, size_t nCols, typename T > Matrix<
nRows, nCols, T > map (T*)(const T &) fct, const Matrix< nRows,
nCols, T > & m)` [**related**]

Apply a unary function to each element of the matrix.

Definition at line 681 of file matrix.h.

```

682 {
683     Matrix<nRows,nCols,T> result;
684     for(size_t i = 0 ; i < nRows ; ++i)
685     {
686         const Vector<nCols,T>& mrow = m[i];
687         Vector<nCols,T>& rrow = result[i];
688         for(size_t j = 0 ; j < nCols ; ++j)
689         {
690             rrow[j] = (*fct)(mrow[j]);
691         }
692     }
693     return result;
694 }
```

17.37.4.20 `template<size_t nRows, size_t nCols, typename T > Matrix<nRows, nCols, T > map (T(*)(T)fct, const Matrix< nRows, nCols, T > &m)` [**related**]

Apply a unary function to each element of the matrix.

Definition at line 659 of file matrix.h.

```

660  {
661      Matrix<nRows,nCols,T> result;
662      for(size_t i = 0 ; i < nRows ; ++i)
663      {
664          const Vector<nCols,T>& mrow = m[i];
665          Vector<nCols,T>& rrow = result[i];
666          for(size_t j = 0 ; j < nCols ; ++j)
667          {
668              rrow[j] = (*fct) (mrow[j]);
669          }
670      }
671      return result;
672  }
```

17.37.4.21 `template<size_t nRows, size_t nCols, typename T > Matrix<nRows, nCols, T > map (const T &(*)(const T &)fct, const Matrix< nRows, nCols, T > &m)` [**related**]

Apply a unary function to each element of the matrix.

Definition at line 638 of file matrix.h.

```

639  {
640      Matrix<nRows,nCols,T> result;
641      for(size_t i = 0 ; i < nRows ; ++i)
642      {
643          const Vector<nCols,T>& mrow = m[i];
644          Vector<nCols,T>& rrow = result[i];
645          for(size_t j = 0 ; j < nCols ; ++j)
646          {
647              rrow[j] = (*fct) (mrow[j]);
648          }
649      }
650      return result;
651  }
```

17.37.4.22 `template<size_t nRows, size_t nCols, typename T > T norm (const Matrix< nRows, nCols, T > &mat)` [**related**]

Return the norm of the matrix.

The norm is defined as the square-root of the sum of the square of the values.

17.37.4.23 `template<size_t nRows, size_t nCols, typename T > T normsq
(const Matrix< nRows, nCols, T > & mat) [related]`

Return the square norm of the matrix.

See also

[norm\(const Matrix&\)](#)

17.37.4.24 `template<size_t nRows, size_t nSize, size_t nCols, typename T >
Matrix< nRows, nCols, T > operator* (const Matrix< nRows,
nSize, T > & mat1, const Matrix< nSize, nCols, T > & mat2)
[related]`

[Matrix](#) multiplication.

17.37.4.25 `template<size_t nRows, size_t nCols, typename T = double> Matrix
operator* (const T & scalar, const Matrix< nRows, nCols, T > &
mat) [friend]`

Matrix-scalar multiplication.

Definition at line 260 of file matrix.h.

```
261     {  
262         Matrix ans;  
263  
264         for(size_t i = 0 ; i < nRows ; i++)  
265             ans[i] = scalar * mat.rows[i];  
266  
267         return ans;  
268     }
```

17.37.4.26 `template<size_t nRows, size_t nCols, typename T > Matrix< nCols,
nRows, T > transpose (const Matrix< nRows, nCols, T > & mat)
[related]`

Transpose a matrix.

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

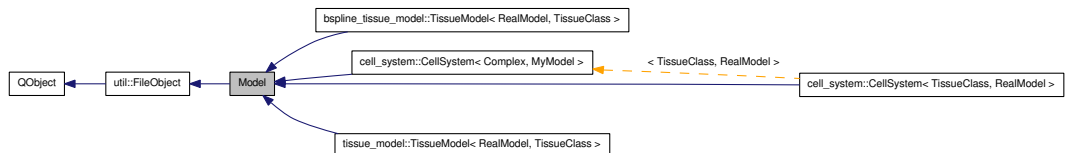
- [vvelib/util/matrix.h](#)

17.38 Model Class Reference

Simulation class.

```
#include <model.h>
```

Inheritance diagram for Model:



Public Slots

- void **registerFile** (std::string filename)
This function registers a file to the file alteration monitor.
- void **setAnimationPeriod** (int ms)
Set the animation period, i.e.
- void **setExitCode** (int code)
Set the exit code.
- void **unregisterFile** (std::string filename)
This function unregisters a file from the file alteration monitor.

Snapshots methods

- bool **loadSnapshot** (const **QString** &filename)
Load a snapshot.
- bool **loadSnapshot** ()
Load a snapshot specified by the user of the model.
- bool **saveNextSnapshot** ()
Save the current model as a snapshot on the current recording.
- bool **saveSnapshot** (const **QString** &filename)
Save the current model as a snapshot.

Screenshot methods

- void **screenshot** (const **QString** &fileName, bool overwrite=false)
Save a screenshot.

Signals

Signals for internal usage only

- void [changedExitCode](#) (int)
Signal emitted when the user changes the exit code.
- void [fileRegister](#) (std::string)
Signal emitted anytime the user is registering a file to the file checker.
- void [fileUnregister](#) (std::string)
Signal emitted anytime the user is unregistering a file .
- void [loadingSnapshot](#) (const **QString** &filename, **Model** *)
Signal emitted when the user wants to load a snapshot.
- void [loadingSnapshot](#) (bool auto_naming, **Model** *)
Signal emitted when the user wants to load a snapshot.
- void [menuActionInserted](#) (**QAction** *before, **QAction** *act)
Signal emitted when an action is inserted.
- void [menuActionRemoved](#) (**QAction** *act)
Signal emitted when an action is removed.
- void [restartModel](#) ()
Signal emitted when the model request the end of computations.
- void [runModel](#) ()
Signal emitted when the model request the end of computations.
- void [savingNextSnapshot](#) (bool, **Model** *)
Signal emitted when the user want to save the next snapshot.
- void [savingScreenshot](#) (const **QString** &filename, bool overwrite)
Signal emitted when the user request the saving of a snapshot.
- void [savingSnapshot](#) (const **QString** &filename, **Model** *)
Signal emitted when the user wants to save a snapshot.
- void [stopModel](#) ()
Signal emitted when the model request the end of computations.

Public Member Functions

- const **QStringList** & [arguments](#) () const
Retrieve the arguments left for the model on the command line (i.e.
- **QString** [errorString](#) () const

Drawing and GUI interaction methods

- **QList< QAction * > actions ()** const
Get the list of actions used to create the context menu on the viewer.
- void **addAction (QAction *action)**
Add an action to the context menu.
- void **addActions (QList< QAction * > actions)**
Add a list of actions to the context menu.
- **QAction * addItem (const QString &text)**
Add a menu item, without connecting any signal.
- **QAction * addItem (const QString &text, const char *slot)**
Add a menu item, the receiver being the model.
- **QAction * addItem (const QString &text, QObject *receiver, const char *slot)**
Add a menu item to attach to the given slot.
- int **animationPeriod ()**
Returns the current animation period.
- virtual void **draw ()**
Convenience function used if the viewer is not necessary.
- virtual void **draw (Viewer *)**
This function should be redefined to draw the representation.
- virtual void **drawWithNames ()**
Convenience function used if the viewer is not necessary.
- virtual void **drawWithNames (Viewer *)**
Draw with object naming.
- virtual void **finalizeDraw ()**
Convenience function used if the viewer is not necessary.
- virtual void **finalizeDraw (Viewer *)**
This function is called if the OpenGL environment is destroyed or if the model is destroyed.
- virtual void **initDraw ()**
Convenience function used if the viewer is not necessary.
- virtual void **initDraw (Viewer *)**
This function is called everytime a new OpenGL environment is created, before the first `preDraw(Viewer)`.*
- void **insertAction (QAction *before, QAction *action)**
Insert an action in the context menu.

- void [insertActions](#) (**QAction** *before, **QList**< **QAction** * > action)
Insert a list of actions in the context menu.
- **QAction** * [menuItem](#) (const **QString** &text) const
Returns the action representing the menu item `text`.
- virtual void [postDraw](#) ()
Convenience function used if the viewer is not necessary.
- virtual void [postDraw](#) (**Viewer** *)
This function should be redefined to restore the original OpenGL context.
- virtual void [postSelection](#) (const **QPoint** &pos, **Viewer** *viewer)
This function is used to resolve a selection event.
- virtual void [preDraw](#) ()
Convenience function used if the viewer is not necessary.
- virtual void [preDraw](#) (**Viewer** *)
This function should be redefined to setup the OpenGL context.
- void [removeAction](#) (**QAction** *action)
Remove an action from the context menu.
- bool [removeMenuItem](#) (const **QString** &text, const char *slot)
Disconnect a menu item, the receiver is the current model.
- bool [removeMenuItem](#) (const **QString** &text, **QObject** *receiver, const char *slot)
Remove the slot from a menu item, if there are more connection, the item is not deleted.
- bool [removeMenuItem](#) (const **QString** &text)
Remove a menu item.
- void [setStatusMessage](#) (const **QString** &msg)
Change the status message.
- **QString** [statusMessage](#) () const
Get the current status message.

Textual output methods

- virtual void [finalizePrint](#) ()
This method is called once when the model is about to be destroyed in batch mode.
- virtual void [initPrint](#) ()
This method is called once before the model is ran in batch mode.

- virtual void `print` ()
This function is called after each step of a model launched in batch mode.

Methods to redefine to change the behavior of the model

- virtual `QString helpString` () const
This function is used to setup the help in the help box.
- virtual void `modifiedFiles` ()
Convenience function to be used if the list of files is ignored.
- virtual void `modifiedFiles` (const std::set< std::string > &)
This function is called anytime one or more registered files are modified.
- virtual void `reread` ()
Reimplement the reread method to allow using the model without a parameter file.
- virtual void `step` ()=0
A step correspond to all the computations between two drawings.

Constructor/destructor

- `Model` (`QObject *parent`)
Constructor of the object.
- virtual `~Model` ()
Virtual destructor ensure the destruction occurring is the one of the leaf class.

Model execution interaction

- virtual void `restart` ()
This function requests a restart of the simulation.
- virtual void `run` ()
This function requests the model to run.
- virtual void `stop` ()
This function requests the end of the simulation.

Snapshot methods

- virtual bool `serialize` (`storage::VVEStorage &`)
Reimplement this method to load/save the current state of the model.
- virtual `QString version` () const
- virtual int `versionNumber` (const `QString &`)

Methods for internal usage only

- void `setError` (bool error, `QString err=QString()`)

17.38.1 Detailed Description

Simulation class. This class has to be the base class of the main class of the project.

It handles all steps specific to the simulation and the drawing of the result. Initialization of the model should occur in the constructor of the class. And any resource deallocation should occur in the destructor. An object of this class will be created at the beginning of each simulation, and destroyed at the end.

The main [Model](#) object has to be registered using the `DEFINE_MODEL(classname)` macro command defined in `<vval.h>`.

Definition at line 44 of file `model.h`.

17.38.2 Constructor & Destructor Documentation

17.38.2.1 `Model::Model (QObject *parent)`

Constructor of the object.

Parameters

parent is the object handling the life of this object. It is required to handle file registration.

Definition at line 20 of file `model_.cpp`.

References `QObject::connect()`, `fileRegister()`, `fileUnregister()`, `QVariant::isValid()`, `loadingSnapshot()`, `loadSnapshot()`, `menuActionInserted()`, `menuActionRemoved()`, `QObject::property()`, `registerFile()`, `restartModel()`, `runModel()`, `saveSnapshot()`, `savingNextSnapshot()`, `savingScreenshot()`, `savingSnapshot()`, `screenshot()`, `stopModel()`, `QVariant::toStringList()`, `QVariant::type()`, `unregisterFile()`, and `graph::vertex_counter`.

```

21 : FileObject(parent)
22 {
23 #ifndef NO_NUMBER_VERTICES
24     graph::vertex_counter = 0;
25 #endif
26     connect(this, SIGNAL(fileRegister(std::string)), parent, SLOT(registerFile(std:
        :string)));
27     connect(this, SIGNAL(fileUnregister(std::string)), parent, SLOT(unregisterFile(
        std::string)));
28     connect(this, SIGNAL(stopModel()), parent, SLOT(stopModel()));
29     connect(this, SIGNAL(runModel()), parent, SLOT(runModel()));
30     connect(this, SIGNAL(restartModel()), parent, SLOT(rewind()));
31     connect(this, SIGNAL(menuActionInserted(QAction*,QAction*)), parent, SLOT(insert
        MenuAction(QAction*,QAction*)));
32     connect(this, SIGNAL(menuActionRemoved(QAction*)), parent, SLOT(removeMenuActio
        n(QAction*)));
33     connect(this, SIGNAL(savingSnapshot(const QString&, Model*)), parent, SLOT(
        saveSnapshot(const QString&, Model*)));
34     connect(this, SIGNAL(savingNextSnapshot(bool, Model*)), parent, SLOT(
        saveSnapshot(bool, Model*)));
35     connect(this, SIGNAL(loadingSnapshot(const QString&, Model*)), parent, SLOT(
        loadSnapshot(const QString&, Model*)));
36     connect(this, SIGNAL(loadingSnapshot(bool, Model*)), parent, SLOT(loadSnapshot(

```

```

        bool, Model*))) ;
37  connect(this, SIGNAL(savingScreenshot(const QString&, bool)), parent, SLOT(
        screenshot(const QString&, bool)));
38
39  QVariant args = parent->property("modelArguments");
40  if(args.isValid())
41  {
42      if(args.type() == QVariant::StringList)
43      {
44          _arguments = args.toStringList();
45      }
46  }
47 }

```

17.38.2.2 virtual Model::~~Model() [inline, virtual]

Virtual destructor ensure the destruction occurring is the one of the leaf class.

Definition at line 63 of file model.h.

```
63 {}
```

17.38.3 Member Function Documentation

17.38.3.1 QList< QAction * > Model::actions() const

Get the list of actions used to create the context menu on the viewer.

Definition at line 125 of file model_.cpp.

Referenced by menuItem().

```

126 {
127     return _actions;
128 }

```

17.38.3.2 void Model::addAction(QAction * action)

Add an action to the context menu.

Definition at line 196 of file model_.cpp.

References insertAction().

Referenced by addItem().

```

197 {
198     insertAction(0, action);
199 }

```

17.38.3.3 void Model::addActions (QList< QAction * > *actions*)

Add a list of actions to the context menu.

Definition at line 201 of file model_.cpp.

References insertActions().

```
202 {  
203     insertActions(0, actions);  
204 }
```

17.38.3.4 QAction * Model::addMenuItem (const QString & *text*)

Add a menu item, without connecting any signal.

Definition at line 140 of file model_.cpp.

References addAction(), and menuItem().

```
141 {  
142     QAction *act = menuItem(text);  
143     if(not act)  
144     {  
145         act = new QAction(text, this);  
146         addAction(act);  
147     }  
148     return act;  
149 }
```

17.38.3.5 QAction * Model::addMenuItem (const QString & *text*, const char * *slot*)

Add a menu item, the receiver being the model.

Definition at line 167 of file model_.cpp.

References addMenuItem().

```
168 {  
169     return addMenuItem(text, this, slot);  
170 }
```

17.38.3.6 QAction * Model::addMenuItem (const QString & *text*, QObject * *receiver*, const char * *slot*)

Add a menu item to attach to the given slot.

Returns

the pointer to the action

Definition at line 151 of file model_.cpp.

References addAction(), QObject::connect(), and menuItem().

Referenced by addMenuItem().

```
152 {  
153     QAction *act = menuItem(text);  
154     if(act)  
155     {  
156         connect(act, SIGNAL(triggered()), receiver, slot);  
157     }  
158     else  
159     {  
160         act = new QAction(text, this);  
161         connect(act, SIGNAL(triggered()), receiver, slot);  
162         addAction(act);  
163     }  
164     return act;  
165 }
```

17.38.3.7 int Model::animationPeriod ()

Returns the current animation period.

Definition at line 116 of file model_.cpp.

References QObject::parent().

```
117 {  
118     bool ok;  
119     int period = parent()->property("animationPeriod").toInt(&ok);  
120     if(!ok)  
121         return -1;  
122     return period;  
123 }
```

17.38.3.8 const QStringList& Model::arguments () const [inline]

Retrieve the arguments left for the model on the command line (i.e.

arguments after the '--')

Definition at line 399 of file model.h.

```
399 { return _arguments; }
```

17.38.3.9 void Model::changedExitCode (int) [signal]

Signal emitted when the user changes the exit code.

Referenced by setExitCode().

17.38.3.10 virtual void Model::draw () [inline, virtual]

Convenience function used if the viewer is not necessary.

Definition at line 171 of file model.h.

```
171 {}
```

17.38.3.11 void Model::draw (Viewer * viewer) [inline, virtual]

This function should be redefined to draw the representation.

Parameters

viewer [Viewer](#) object. Useful to use the capabilities of [QGLViewer](#) object.

Reimplemented in [bspline_tissue_model::TissueModel< RealModel, TissueClass >](#), and [tissue_model::TissueModel< RealModel, TissueClass >](#).

Definition at line 167 of file model.h.

References [draw\(\)](#).

Referenced by [draw\(\)](#).

```
167 { this->draw(); }
```

17.38.3.12 virtual void Model::drawWithNames () [inline, virtual]

Convenience function used if the viewer is not necessary.

Reimplemented in [bspline_tissue_model::TissueModel< RealModel, TissueClass >](#), and [tissue_model::TissueModel< RealModel, TissueClass >](#).

Definition at line 238 of file model.h.

```
238 {}
```

17.38.3.13 void Model::drawWithNames (Viewer * viewer) [inline, virtual]

Draw with object naming.

Used for object selection. The name is to be setup using [glPushName\(int\)](#) / [glPopName\(\)](#)

Parameters

viewer [Viewer](#) object. Useful to use the capabilities of [QGLViewer](#) object.

Definition at line 234 of file model.h.

References drawWithNames().

Referenced by drawWithNames().

```
234 { this->drawWithNames(); }
```

17.38.3.14 void Model::fileRegister (std::string) [signal]

Signal emitted anytime the user is registering a file to the file checker.

Referenced by Model(), and registerFile().

17.38.3.15 void Model::fileUnregister (std::string) [signal]

Signal emitted anytime the user is unregistering a file .

..

Referenced by Model(), and unregisterFile().

17.38.3.16 virtual void Model::finalizeDraw () [inline, virtual]

Convenience function used if the viewer is not necessary.

Definition at line 223 of file model.h.

```
223 {}
```

17.38.3.17 void Model::finalizeDraw (Viewer * viewer) [inline, virtual]

This function is called if the OpenGL environment is destroyed or if the model is destroyed.

Parameters

viewer [Viewer](#) object. Useful to use the capabilities of [QGLViewer](#) object.

Definition at line 219 of file model.h.

References finalizeDraw().

Referenced by finalizeDraw().

```
219 { this->finalizeDraw(); }
```

17.38.3.18 virtual void Model::finalizePrint () [inline, virtual]

This method is called once when the model is about to be destroyed in batch mode.

It is intended to initialize any variable or to print an initial message. Any output should be directed on the standard output of the application.

Definition at line 372 of file model.h.

```
372 {}
```

17.38.3.19 QString Model::helpString () const [virtual]

This function is used to setup the help in the help box.

It should typically return a description of the model. Default implementation will read the description.txt file if available.

Definition at line 59 of file model_.cpp.

References `QString::append()`, `QFile::exists()`, `QFile::open()`, and `QString::prepend()`.

```
60 {
61     if(QFile::exists("description.html"))
62     {
63         QFile f("description.html");
64         if(f.open(QIODevice::ReadOnly | QIODevice::Text))
65         {
66             QTextStream fss(&f);
67             QString content = f.readAll();
68             return content;
69         }
70     }
71     else if(QFile::exists("description.txt"))
72     {
73         QFile f("description.txt");
74         if(f.open(QIODevice::ReadOnly | QIODevice::Text))
75         {
76             QTextStream fss(&f);
77             QString content = f.readAll();
78             content.prepend("<pre>");
79             content.append("</pre>");
80             return content;
81         }
82     }
83     return QString("Undocumented VV model");
84 }
```

17.38.3.20 virtual void Model::initDraw () [inline, virtual]

Convenience function used if the viewer is not necessary.

Reimplemented in [bspline_tissue_model::TissueModel< RealModel, TissueClass >](#), and [tissue_model::TissueModel< RealModel, TissueClass >](#).

Definition at line 209 of file model.h.

```
209 {}
```

17.38.3.21 void Model::initDraw (Viewer * viewer) [inline, virtual]

This function is called everytime a new OpenGL environment is created, before the first [preDraw\(Viewer*\)](#).

Warning

It may be called multiple times if the OpenGL environment is destroyed and recreated.

Useful to initialize camera and lights or to gather data about the rendering capabilities. Also used for any initialization requiring a valid OpenGL context.

Parameters

viewer [Viewer](#) object. Useful to use the capabilities of [QGLViewer](#) object.

Definition at line 205 of file model.h.

References [initDraw\(\)](#).

Referenced by [initDraw\(\)](#).

```
205 { this->initDraw(); }
```

17.38.3.22 virtual void Model::initPrint () [inline, virtual]

This method is called once before the model is ran in batch mode.

It is intended to initialize any variable or to print an initial message. Any output should be directed on the standard output of the application.

Definition at line 354 of file model.h.

```
354 {}
```

17.38.3.23 void Model::insertAction (QAction * before, QAction * action)

Insert an action in the context menu.

Definition at line 206 of file model_.cpp.

References [QList::indexOf\(\)](#), [QList::insert\(\)](#), [menuActionInserted\(\)](#), and [QList::size\(\)](#).

Referenced by [addAction\(\)](#), and [insertActions\(\)](#).

```
207 {
208     int pos = _actions.indexOf(before);
209     if(pos < 0)
210     {
211         pos = _actions.size();
212     }
213     _actions.insert(pos, action);
214     emit menuActionInserted(before, action);
215 }
```

17.38.3.24 void Model::insertActions (QAction * *before*, QList< QAction * > *action*)

Insert a list of actions in the context menu.

Definition at line 217 of file model_.cpp.

References insertAction(), and QList::size().

Referenced by addActions().

```
218 {
219     for(int i = 0 ; i < actions.size() ; ++i)
220     {
221         insertAction(before, actions[i]);
222     }
223 }
```

17.38.3.25 void Model::loadingSnapshot (const QString & *filename*, Model *) [signal]

Signal emitted when the user wants to load a snapshot.

Parameters

filename Name of the snapshot. The extension will be used to determine the format.

17.38.3.26 void Model::loadingSnapshot (bool *auto_naming*, Model *) [signal]

Signal emitted when the user wants to load a snapshot.

Referenced by loadSnapshot(), and Model().

17.38.3.27 bool Model::loadSnapshot (const QString & *filename*) [slot]

Load a snapshot.

be careful with the state of the model after this call

Definition at line 252 of file model_.cpp.

References loadingSnapshot().

```
253 {  
254     _error = false;  
255     emit loadingSnapshot(filename, this);  
256     return !_error;  
257 }
```

17.38.3.28 bool Model::loadSnapshot () [slot]

Load a snapshot specified by the user of the model.

be careful with the state of the model after this call

Definition at line 245 of file model_.cpp.

References loadingSnapshot().

Referenced by Model().

```
246 {  
247     _error = false;  
248     emit loadingSnapshot(false, this);  
249     return !_error;  
250 }
```

17.38.3.29 void Model::menuActionInserted (QAction * *before*, QAction * *act*) [signal]

Signal emitted when an action is inserted.

Referenced by insertAction(), and Model().

17.38.3.30 void Model::menuActionRemoved (QAction * *act*) [signal]

Signal emitted when an action is removed.

Referenced by Model(), and removeAction().

17.38.3.31 QAction * Model::menuItem (const QString & *text*) const

Returns the action representing the menu item *text*.

Returns

0 if no such item exist

Definition at line 130 of file model_.cpp.

References actions().

Referenced by addItem(), and removeMenuItem().

```
131 {
132     foreach(QAction* act, actions())
133     {
134         if(act->text() == text)
135             return act;
136     }
137     return 0;
138 }
```

17.38.3.32 virtual void Model::modifiedFiles () [inline, virtual]

Convenience function to be used if the list of files is ignored.

Definition at line 87 of file model.h.

```
87 {}
```

17.38.3.33 void Model::modifiedFiles (const std::set< std::string > &files) [inline, virtual]

This function is called anytime one or more registered files are modified.

Parameters

files Registered files modified since last call to this function.

Reimplemented in [bspline_tissue_model::TissueModel< RealModel, TissueClass >](#), and [tissue_model::TissueModel< RealModel, TissueClass >](#).

Definition at line 82 of file model.h.

References modifiedFiles().

Referenced by modifiedFiles().

```
83     { this->modifiedFiles(); }
```

17.38.3.34 virtual void Model::postDraw () [inline, virtual]

Convenience function used if the viewer is not necessary.

Reimplemented in [bspline_tissue_model::TissueModel< RealModel, TissueClass >](#), and [tissue_model::TissueModel< RealModel, TissueClass >](#).

Definition at line 188 of file model.h.

```
188 {}
```

17.38.3.35 void Model::postDraw (Viewer * viewer) [inline, virtual]

This function should be redefined to restore the original OpenGL context.

It is called right after the [draw\(Viewer*\)](#) method and should restore the OpenGL context to allow for correct drawing of the decorators (grid, axes, ...)

Parameters

viewer [Viewer](#) object. Useful to use the capabilities of [QGLViewer](#) object.

Definition at line 184 of file model.h.

References [postDraw\(\)](#).

Referenced by [postDraw\(\)](#).

```
184 { this->postDraw(); }
```

17.38.3.36 void Model::postSelection (const QPoint & pos, Viewer * viewer) [virtual]

This function is used to resolve a selection event.

Parameters

pos 2D position of the user click event

viewer [Viewer](#) object on which the selection occurred

It should implement the action associated with a selection. The user can call [viewer->selectedName\(\)](#) to get the id of the selected item. The number returned depend on the number allocated within the [drawWithNames\(\)](#) method.

Definition at line 101 of file model_.cpp.

```
102 {
103 }
```

17.38.3.37 virtual void Model::preDraw () [inline, virtual]

Convenience function used if the viewer is not necessary.

Reimplemented in [bspline_tissue_model::TissueModel< RealModel, TissueClass >](#), and [tissue_model::TissueModel< RealModel, TissueClass >](#).

Definition at line 158 of file model.h.

```
158 {}
```


17.38.3.38 void Model::preDraw (Viewer * *viewer*) [inline, virtual]

This function should be redefined to setup the OpenGL context.

Called at the beginning of every drawing, right after the initialization of the OpenGL context.

Parameters

viewer [Viewer](#) object. Useful to use the capabilities of [QGLViewer](#) object.

Definition at line 154 of file model.h.

References [preDraw\(\)](#).

Referenced by [preDraw\(\)](#).

```
154 { this->preDraw(); }
```

17.38.3.39 virtual void Model::print () [inline, virtual]

This function is called after each step of a model launched in batch mode.

The output is expected to be on the standard output (i.e. stdout or cout).

Definition at line 363 of file model.h.

```
363 {}
```

17.38.3.40 void Model::registerFile (std::string *filename*) [slot]

This function registers a file to the file alteration monitor.

Definition at line 49 of file model_.cpp.

References [fileRegister\(\)](#).

Referenced by [Model\(\)](#), and [bspline_tissue_model::TissueModel< RealModel, TissueClass >::registerFiles\(\)](#).

```
50 {  
51     emit fileRegister(filename);  
52 }
```

17.38.3.41 void Model::removeAction (QAction * *action*)

Remove an action from the context menu.

Definition at line 225 of file model_.cpp.

References [menuActionRemoved\(\)](#), and [QList::removeAll\(\)](#).

Referenced by [removeMenuItem\(\)](#).

```
226 {
227     if(_actions.removeAll(action))
228         emit menuActionRemoved(action);
229 }
```

17.38.3.42 **bool Model::removeMenuItem (const QString & *text*, const char * *slot*)**

Disconnect a menu item, the receiver is the current model.

Definition at line 183 of file model_.cpp.

References removeMenuItem().

```
184 {
185     return removeMenuItem(text, this, slot);
186 }
```

17.38.3.43 **bool Model::removeMenuItem (const QString & *text*, QObject * *receiver*, const char * *slot*)**

Remove the slot from a menu item, if there are more connection, the item is not deleted.

Returns

true if the menu item and the connection are found

Definition at line 188 of file model_.cpp.

References QObject::disconnect(), and menuItem().

```
189 {
190     QAction* toremove = menuItem(text);
191     if(toremove and disconnect(toremove, SIGNAL(triggered()), receiver, slot))
192         return true;
193     return false;
194 }
```

17.38.3.44 **bool Model::removeMenuItem (const QString & *text*)**

Remove a menu item.

Returns

true is the menu item is found

Definition at line 172 of file model_.cpp.

References menuItem(), and removeAction().

Referenced by removeMenuItem().

```
173 {
174     QAction* toremove = menuItem(text);
175     if(toremove)
176     {
177         removeAction(toremove);
178         return true;
179     }
180     return false;
181 }
```

17.38.3.45 virtual void Model::reread () [inline, virtual]

Reimplement the reread method to allow using the model without a parameter file.

Implements [util::FileObject](#).

Definition at line 93 of file model.h.

```
93 {}
```

17.38.3.46 void Model::restart () [virtual]

This function requests a restart of the simulation.

Definition at line 96 of file model_.cpp.

References `restartModel()`.

```
97 {
98     emit restartModel();
99 }
```

17.38.3.47 void Model::restartModel () [signal]

Signal emitted when the model request the end of computations.

Referenced by `Model()`, and `restart()`.

17.38.3.48 void Model::run () [virtual]

This function requests the model to run.

Definition at line 86 of file model_.cpp.

References `runModel()`.

```
87 {
88     emit runModel();
89 }
```

17.38.3.49 void Model::runModel () [signal]

Signal emitted when the model request the end of computations.

Referenced by Model(), and run().

17.38.3.50 bool Model::saveNextSnapshot () [slot]

Save the current model as a snapshot on the current recording.

If the model is not being recorded, this method has no effect

Definition at line 238 of file model_.cpp.

References savingNextSnapshot().

```
239 {  
240     _error = false;  
241     emit savingNextSnapshot(true, this);  
242     return !_error;  
243 }
```

17.38.3.51 bool Model::saveSnapshot (const QString &filename) [slot]

Save the current model as a snapshot.

Definition at line 231 of file model_.cpp.

References savingSnapshot().

Referenced by Model().

```
232 {  
233     _error = false;  
234     emit savingSnapshot(filename, this);  
235     return !_error;  
236 }
```

17.38.3.52 void Model::savingNextSnapshot (bool, Model *) [signal]

Signal emitted when the user want to save the next snapshot.

Useful to save intermediate state ...

Referenced by Model(), and saveNextSnapshot().

17.38.3.53 void Model::savingScreenshot (const QString &filename, bool overwrite) [signal]

Signal emitted when the user request the saving of a snapshot.

Referenced by Model(), and screenshot().

17.38.3.54 `void Model::savingSnapshot (const QString &filename, Model *)` [**signal**]

Signal emitted when the user wants to save a snapshot.

Parameters

filename Name of the snapshot. The extension will be used to determine the format.

Referenced by `Model()`, and `saveSnapshot()`.

17.38.3.55 `void Model::screenshot (const QString &fileName, bool overwrite = false)` [**slot**]

Save a screenshot.

See also

`QGLViewer::saveSnapshot(const QString&, bool)`

Definition at line 259 of file `model_.cpp`.

References `savingScreenshot()`.

Referenced by `Model()`.

```
260 {
261     emit savingScreenshot(filename, overwrite);
262 }
```

17.38.3.56 `virtual bool Model::serialize (storage::VVEStorage &) [inline, virtual]`

Reimplement this method to load/save the current state of the model.

Returns

true if serialization was successful, false otherwise.

Parameters

store Object to store the state in the model in.

Reimplemented in [cell_system::CellSystem< Complex, MyModel >](#), and [cell_system::CellSystem< TissueClass, RealModel >](#).

Definition at line 120 of file `model.h`.

Referenced by `storage::VVEStorage::field()`, `storage::VVEStorage_XMLReader::serialize()`, `storage::VVEStorage_XMLWriter::serialize()`, `storage::VVEStorage_BINReader::serialize()`, `storage::old::VVEStorage_BINReader::serialize()`, and `storage::VVEStorage_BINWriter::serialize()`.

```
120 { return false; }
```

17.38.3.57 void Model::setAnimationPeriod (int *ms*) [slot]

Set the animation period, i.e.

the period in between two calls to the [step\(\)](#) method, in ms.

Definition at line 110 of file `model_.cpp`.

References `QObject::parent()`.

```
111 {  
112     if (parent())  
113         parent()->setProperty("animationPeriod", ms);  
114 }
```

17.38.3.58 void Model::setExitCode (int *code*) [slot]

Set the exit code.

Useful for testing purposes.

Definition at line 105 of file `model_.cpp`.

References `changedExitCode()`.

```
106 {  
107     emit changedExitCode(code);  
108 }
```

17.38.3.59 void Model::setStatusMessage (const QString & *msg*)

Change the status message.

On the gui, display the message on the status bar. On batch mode, prints the message to the console.

Definition at line 264 of file `model_.cpp`.

References `QObject::parent()`.

```
265 {  
266     bool success = parent()->setProperty("statusMessage", msg);  
267     assert(success);  
268 }
```

17.38.3.60 QString Model::statusMessage () const

Get the current status message.

Definition at line 270 of file model_.cpp.

References `QObject::parent()`.

```
271 {  
272     QString result = parent()->property("statusMessage").toString();  
273     return result;  
274 }
```

17.38.3.61 `virtual void Model::step ()` [**pure virtual**]

A step correspond to all the computations between two drawings.

It is the only strictly required method.

Implemented in `cell_system::CellSystem< Complex, MyModel >`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >`, `tissue_model::TissueModel< RealModel, TissueClass >`, and `cell_system::CellSystem< TissueClass, RealModel >`.

17.38.3.62 `void Model::stop ()` [**virtual**]

This function requests the end of the simulation.

Definition at line 91 of file model_.cpp.

References `stopModel()`.

```
92 {  
93     emit stopModel();  
94 }
```

17.38.3.63 `void Model::stopModel ()` [**signal**]

Signal emitted when the model request the end of computations.

Referenced by `Model()`, and `stop()`.

17.38.3.64 `void Model::unregisterFile (std::string filename)` [**slot**]

This function unregisters a file from the file alteration monitor.

Definition at line 54 of file model_.cpp.

References `fileUnregister()`.

Referenced by `Model()`.

```
55 {  
56     emit fileUnregister(filename);  
57 }
```

17.38.3.65 virtual QString Model::version () const [inline, virtual]**Returns**

the current version of the class.

The default implementation returns an empty string

Reimplemented in [cell_system::CellSystem< Complex, MyModel >](#), and [cell_system::CellSystem< TissueClass, RealModel >](#).

Definition at line 137 of file model.h.

Referenced by [storage::VVEStorage_XMLWriter::serialize\(\)](#), and [storage::VVEStorage_BINWriter::serialize\(\)](#).

```
137 { return ""; }
```

17.38.3.66 virtual int Model::versionNumber (const QString &) [inline, virtual]**Returns**

an integer representing the version string for the model.

The default implementation always return 0.

The number has no meaning outside the model.

Reimplemented in [cell_system::CellSystem< Complex, MyModel >](#), and [cell_system::CellSystem< TissueClass, RealModel >](#).

Definition at line 130 of file model.h.

Referenced by [storage::VVEStorage_XMLReader::serialize\(\)](#), [storage::VVEStorage_XMLWriter::serialize\(\)](#), [storage::VVEStorage_BINReader::serialize\(\)](#), [storage::old::VVEStorage_BINReader::serialize\(\)](#), and [storage::VVEStorage_BINWriter::serialize\(\)](#).

```
130 { return 0; }
```

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

- [vvelib/model.h](#)
- [vvelib/model_.cpp](#)

17.39 graph::VVGraph< VertexContent, EdgeContent, compact >::neighbor_t Struct Reference

Structure maintaining the data for a single neighbor.

```
#include <vvgraph.h>
```

Inherits graph::__vvgraph_EdgeCache< VVGRAPH_COMPACT_EDGE, EdgeContent, VVGraph, single_neighborhood_t, std::list< neighbor_t > >.

Public Types

- typedef __vvgraph_EdgeCache< VVGRAPH_COMPACT_EDGE, EdgeContent, VVGraph, single_neighborhood_t, std::list< neighbor_t > > **base**
- typedef std::list< neighbor_t > **edge_list_t**

Type of the list of outgoing neighbors.

Public Member Functions

- void **clear_edge** ()
- **neighbor_t** (const neighbor_t ©)
- **neighbor_t** (const vertex_t &tgt, const EdgeContent &e, single_neighborhood_t &n, intptr_t gid)

Constructor.

- **neighbor_t** & **operator=** (const neighbor_t ©)
- bool **operator==** (const neighbor_t &other) const

Equality of two neighbors/edge.

Public Attributes

- **vertex_t** **target**

Target of the edge.

17.39.1 Detailed Description

```
template<typename VertexContent, typename EdgeContent = _-  
EmptyEdgeContent, bool compact = false> struct graph::VVGraph< Ver-  
texContent, EdgeContent, compact >::neighbor_t
```

Structure maintaining the data for a single neighbor.

Definition at line 431 of file vvgraph.h.

17.39.2 Member Typedef Documentation

17.39.2.1 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> typedef std::list<neighbor_t> graph::VVGraph< VertexContent, EdgeContent, compact >::neighbor_t::edge_list_t`

Type of the list of outgoing neighbors.

Definition at line 437 of file vvgraph.h.

17.39.3 Constructor & Destructor Documentation

17.39.3.1 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> graph::VVGraph< VertexContent, EdgeContent, compact >::neighbor_t::neighbor_t (const vertex_t & tgt, const EdgeContent & e, single_neighborhood_t & n, intptr_t gid) [inline]`

Constructor.

Definition at line 442 of file vvgraph.h.

```

445             : base(e, &n, gid)
446             , target(tgt)
447             {
448             }
```

17.39.4 Member Function Documentation

17.39.4.1 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> bool graph::VVGraph< VertexContent, EdgeContent, compact >::neighbor_t::operator==(const neighbor_t & other) const [inline]`

Equality of two neighbors/edge.

A single neighbor is equal to another if the vertex and the content are the same

Definition at line 485 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::neighbor_t::target`.

```

486         {
487             return (target == other.target) &&
488                 (static_cast<const EdgeContent&>(*this) == static_cast<const EdgeCo
489                 ntent&>(other));
489         }
```

17.39.5 Member Data Documentation

17.39.5.1 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> vertex_t
graph::VVGraph< VertexContent, EdgeContent, compact
>::neighbor_t::target`

Target of the edge.

i.e. neighbor itself

Definition at line 477 of file vvgraph.h.

Referenced by `graph::VVGraph< VertexContent, EdgeContent, compact
>::neighbor_t::operator==()`, and `graph::VVGraph< VertexContent, EdgeContent,
compact >::updateEdgeCache()`.

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

- [vvelib/graph/vvgraph.h](#)

17.40 complex_factory::ObjReaderError Class Reference

Error object returned by the [complex_factory::objreader](#) functions.

```
#include <factory/objreader.h>
```

Public Types

- enum [Type](#) {
 NO_ERRORS, **BAD_NUMBER_OF_DIMENSIONS**, **FILE_FORMAT**,
 DOUBLE_EXPECTED,
 INTEGER_EXPECTED, **INVALID_FACE**, **CANNOT_OPEN_FILE** }

Type of the error.

Public Member Functions

- **ObjReaderError** (const [ObjReaderError](#) ©)
- **ObjReaderError** ([Type](#) t, **QString** msg)
- **ObjReaderError** ([Type](#) t)
- **operator bool** ()
Convert to true if the object correspond to an error.
- virtual **QString** [string](#) ()
Returns a human-readable string describing the error.
- virtual int [type](#) ()
Returns the type of the error.

Protected Member Functions

- void **initFromType** ()

Protected Attributes

- **QString** [_string](#)
- [Type](#) [_type](#)

17.40.1 Detailed Description

Error object returned by the [complex_factory::objreader](#) functions.

Definition at line 30 of file [objreader.h](#).

17.40.2 Member Enumeration Documentation

17.40.2.1 enum complex_factory::ObjReaderError::Type

Type of the error.

Definition at line 36 of file objreader.h.

```
37     {
38         NO_ERRORS, // Due to windows crap, NO_ERROR is not usable (but equivalent
39         ?)
38         BAD_NUMBER_OF_DIMENSIONS,
40         FILE_FORMAT,
41         DOUBLE_EXPECTED,
42         INTEGER_EXPECTED,
43         INVALID_FACE,
44         CANNOT_OPEN_FILE
45     };
```

17.40.3 Member Function Documentation

17.40.3.1 complex_factory::ObjReaderError::operator bool () [inline]

Convert to true if the object correspond to an error.

Definition at line 68 of file objreader.h.

```
68 { return _type != NO_ERRORS; }
```

17.40.3.2 virtual QString complex_factory::ObjReaderError::string () [inline, virtual]

Returns a human-readable string describing the error.

Definition at line 58 of file objreader.h.

```
58 { return _string; }
```

17.40.3.3 virtual int complex_factory::ObjReaderError::type () [inline, virtual]

Returns the type of the error.

Definition at line 63 of file objreader.h.

```
63 { return _type; }
```

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

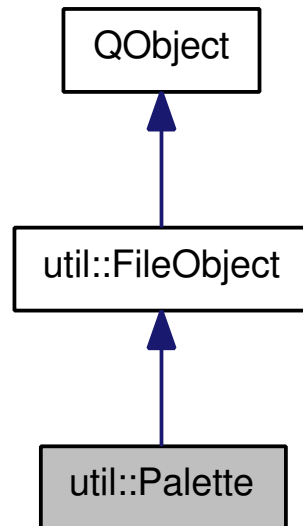
- vvelib/factory/[objreader.h](#)
- vvelib/factory/[objreader.cpp](#)

17.41 util::Palette Class Reference

A utility class for palettes.

```
#include <util/palette.h>
```

Inheritance diagram for util::Palette:



Public Types

- typedef [util::Color](#)< GLfloat > **Color**

Public Member Functions

- void [blend](#) (unsigned int ind1, unsigned int ind2, double w, double alpha=1)
const
Blend two colors together and make the result the current OpenGL color.
- [Color getColor](#) (unsigned int ind1, unsigned int ind2, double w, double alpha=1)
const
Blend two colors in the palette together.
- [Color getColor](#) (unsigned int index, double alpha=1) const
Get a color from the palette.
- [Palette](#) ()
Construct an empty palette with only black colors.
- [Palette](#) (std::string filename)

Construct the palette from a file.

- void `reread()`
Reread the file describing the palette.
- void `select` (unsigned int start, unsigned int end, double w, double alpha=1) const
Select a color from a range in a palette and make it the current OpenGL color.
- `Color selectColor` (unsigned int start, unsigned int end, double w, double alpha=1) const
Select a color from a range in a palette and returns it.
- void `useColor` (unsigned int index, double alpha=1) const
Get a color and make it the current OpenGL color.

Static Public Member Functions

- static `Color blend` (const `Color` &c1, const `Color` &c2, double w)
Blend two colors together.

17.41.1 Detailed Description

A utility class for palettes. This class provides an interface for VLAB palette files and their use for OpenGL.

Definition at line 23 of file `palette.h`.

17.41.2 Constructor & Destructor Documentation

17.41.2.1 util::Palette::Palette (std::string *filename*)

Construct the palette from a file.

Parameters

filename File to read the palette from

Definition at line 10 of file `palette.cpp`.

References `reread()`.

```
11      : FileObject(filename)
12      {
13          reread();
14      }
```

17.41.2.2 util::Palette::Palette ()

Construct an empty palette with only black colors.

Definition at line 16 of file palette.cpp.

```

17      : FileObject()
18      {
19          for(int i = 0 ; i < 255 ; ++i)
20          {
21              colors[i] = 0.0f;
22          }
23      }
```

17.41.3 Member Function Documentation

17.41.3.1 Palette::Color util::Palette::blend (const Color & *c1*, const Color & *c2*, double *w*) [static]

Blend two colors together.

Definition at line 87 of file palette.cpp.

References util::Color< T >::a(), util::Color< T >::b(), util::Color< T >::g(), and util::Color< T >::r().

```

88      {
89          Color result( GLfloat(a.r() * (1.0 - w) + b.r() * w),
90                      GLfloat(a.g() * (1.0 - w) + b.g() * w),
91                      GLfloat(a.b() * (1.0 - w) + b.b() * w),
92                      GLfloat(a.a() * (1.0 - w) + b.a() * w));
93          return result;
94      }
```

17.41.3.2 void util::Palette::blend (unsigned int *ind1*, unsigned int *ind2*, double *w*, double *alpha* = 1) const

Blend two colors together and make the result the current OpenGL color.

Parameters

ind1 Index of the first color

ind2 Index of the second color

w Blending coefficient, from 0 to 1. The color is linearly interpolated from *ind1* to *ind2*.

alpha Alpha value to use for this color.

Definition at line 82 of file palette.cpp.

References getColor().


```
83 {
84     glColor4fv( getColor(ind1, ind2, w, alpha).c_data() );
85 }
```

17.41.3.3 Palette::Color util::Palette::getColor (unsigned int *ind1*, unsigned int *ind2*, double *w*, double *alpha* = 1) const

Blend two colors in the palette together.

Parameters

ind1 Index of the first color (between 0 and 255)

ind2 Index of the second color (between 0 and 255)

w Blending coefficient, from 0 to 1. The color is linearly interpolated from *ind1* to *ind2*. ***alpha*** Alpha value to use for this color (between 0 and 1)

Definition at line 54 of file palette.cpp.

References util::Color< T >::b(), util::Color< T >::g(), getColor(), and util::Color< T >::r().

```
55 {
56     if (ind1 > 255)
57         ind1 = 255;
58     if (ind2 > 255)
59         ind2 = 255;
60     if (w > 1.0)
61         w = 1.0;
62     else if (w < 0.0)
63         w = 0.0;
64
65     if(ind1 == ind2)
66         return getColor(ind1, alpha);
67     else if(w < COLOR_EPSILON)
68         return getColor(ind1, alpha);
69     else if(w > 1-COLOR_EPSILON)
70         return getColor(ind2, alpha);
71     else
72     {
73         Color a = colors[ind1], b = colors[ind2];
74         Color result( GLfloat(a.r() * (1.0 - w) + b.r() * w),
75                     GLfloat(a.g() * (1.0 - w) + b.g() * w),
76                     GLfloat(a.b() * (1.0 - w) + b.b() * w),
77                     GLfloat(alpha));
78         return result;
79     }
80 }
```

17.41.3.4 Palette::Color util::Palette::getColor (unsigned int *index*, double *alpha* = 1) const

Get a color from the palette.

Parameters

index Index of the color in the palette (between 0 and 255)

alpha Alpha value to use for this color (between 0 and 1)

Definition at line 40 of file palette.cpp.

References util::Color< T >::a().

Referenced by blend(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawWalledCell(), getColor(), tissue_model::TissueModel< RealModel, TissueClass >::preDraw(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::preDraw(), selectColor(), and useColor().

```

41  {
42      if (index > 255)
43          index = 255;
44      Color color = colors[index];
45      color.a(GLfloat(alpha));
46      return color;
47  }
```

17.41.3.5 void util::Palette::reread () [virtual]

Reread the file describing the palette.

Implements [util::FileObject](#).

Definition at line 25 of file palette.cpp.

References util::Color< T >::b(), util::Color< T >::g(), and util::Color< T >::r().

Referenced by Palette().

```

26  {
27      std::ifstream in(filename.c_str(), std::ios::binary);
28
29      unsigned int index = 0;
30      while (!in.eof() || !in.good() || !in)
31      {
32          if (index > 255) break;
33          colors[index].r(GLfloat(in.get())/255.f);
34          colors[index].g(GLfloat(in.get())/255.f);
35          colors[index].b(GLfloat(in.get())/255.f);
36          index++;
37      }
38  }
```

17.41.3.6 void util::Palette::select (unsigned int start, unsigned int end, double w, double alpha = 1) const

Select a color from a range in a palette and make it the current OpenGL color.

Parameters

- start* Index of the first color of the range
- end* Index of the last color of the range
- w* Position within the range, if the value correspond to a color in between two defined colors, they will be linearly interpolated.
- alpha* Alpha value to use for this color.

Definition at line 119 of file palette.cpp.

References `selectColor()`.

```

120  {
121      glColor4fv(selectColor(start, end, w, alpha).c_data());
122  }
```

17.41.3.7 Palette::Color util::Palette::selectColor (unsigned int *start*, unsigned int *end*, double *w*, double *alpha* = 1) const

Select a color from a range in a palette and returns it.

Parameters

- start* Index of the first color of the range
- end* Index of the last color of the range
- w* Position within the range, if the value correspond to a color in between two defined colors, they will be linearly interpolated.
- alpha* Alpha value to use for this color.

Definition at line 96 of file palette.cpp.

References `getColor()`.

Referenced by `select()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::valueCenterColor()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::valueColor()`.

```

97  {
98      if (start > 255)
99          start = 255;
100     if (end > 255)
101         end = 255;
102     if (w > 1.0)
103         w = 1.0;
104     else if (w < 0.0)
105         w = 0.0;
106
107     int delta_color = (int)end-(int)start;
108     double pos_w = delta_color*w + start;
```

```
109     int before = (int)std::floor(pos_w);
110     int after = before+1;
111     double ratio = pos_w - (double)before;
112     if(ratio < COLOR_EPSILON)
113         return getColor(before, alpha);
114     if(ratio > 1-COLOR_EPSILON)
115         return getColor(after, alpha);
116     return getColor(before, after, ratio, alpha);
117 }
```

17.41.3.8 void util::Palette::useColor (unsigned int *index*, double *alpha* = 1) const

Get a color and make it the current OpenGL color.

Parameters

index Index of the color in the palette

alpha Alpha value fo use for this color

Definition at line 49 of file palette.cpp.

References getColor().

Referenced by tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell(), and tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCellContour().

```
50 {
51     glColor4fv(getColor(index, alpha).c_data());
52 }
```

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

- vvelib/util/[palette.h](#)
- vvelib/util/palette.cpp

17.42 ParallelControler Class Reference

This class is used to control how many threads are used for parallel processing.

```
#include <algorithms/parallel.h>
```

17.42.1 Detailed Description

This class is used to control how many threads are used for parallel processing. It also detect automatically how many processors are available on the machine.

Definition at line 31 of file parallel.h.

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

- [vvelib/algorithms/parallel.h](#)

17.43 util::Parms Class Reference

A utility class to parse L-Studio like parameter files.

```
#include <util/parms.h>
```

Public Member Functions

- bool [all](#) (const **QString** §ion, std::map< **QString**, **QStringList** > &value)
Retrieve a all parameters with same [section]key.
- bool [all](#) (const **QString** §ion, std::map< **QString**, std::vector< **QString** > > &value)
Retrieve a all parameters with same [section]key.
- bool [all](#) (const **QString** §ion, std::map< **QString**, std::vector< std::string > > &value)
Retrieve a all parameters with same [section]key.
- bool [all](#) (const **QString** §ion, std::map< **QString**, std::vector< double > > &value)
Retrieve a all parameters with same [section].
- bool [all](#) (const **QString** §ion, std::map< **QString**, std::vector< float > > &value)
Retrieve a all parameters with same [section].
- bool [all](#) (const **QString** §ion, std::map< **QString**, std::vector< int > > &value)
Retrieve a all parameters with same [section].
- bool [all](#) (const **QString** §ion, std::map< **QString**, std::vector< bool > > &value)
Retrieve a all parameters with same [section].
- template<typename T , typename Container >
bool [all](#) (const **QString** §ion, std::map< **QString**, Container > &values)
This operator retrieves all parameters with same [section].
- bool [all](#) (const **QString** §ion, const **QString** &key, **QStringList** &value)
Retrieve a all parameters with same [section]key.
- bool [all](#) (const **QString** §ion, const **QString** &key, std::vector< **QString** > &value)
Retrieve a all parameters with same [section]key.

- `bool all (const QString §ion, const QString &key, std::vector< std::string > &value)`
Retrieve a all parameters with same [section]key.
- `bool all (const QString §ion, const QString &key, std::vector< double > &value)`
Retrieve a all parameters with same [section]key.
- `bool all (const QString §ion, const QString &key, std::vector< float > &value)`
Retrieve a all parameters with same [section]key.
- `bool all (const QString §ion, const QString &key, std::vector< int > &value)`
Retrieve a all parameters with same [section]key.
- `bool all (const QString §ion, const QString &key, std::vector< bool > &value)`
Retrieve a all parameters with same [section]key.
- `template<typename Container >`
`bool all (const QString §ion, const QString &key, Container &value)`
This operator retrieves all parameters with same [section]key.
- `bool isLoading () const`
Returns true if the parameter object has been correctly loaded.
- `template<typename T >`
`bool operator() (const QString §ion, const QString &key, T &value, const T &def)`
Variation on the previous, but if the [section]key is not found, an information message is issued (instead of an error) and value is set up to def.
- `bool operator() (const QString §ion, const QString &key, QString &value)`
Retrieve a single parameter.
- `bool operator() (const QString §ion, const QString &key, std::string &value)`
Retrieve a single parameter.
- `bool operator() (const QString §ion, const QString &key, double &value)`
Retrieve a single parameter.
- `bool operator() (const QString §ion, const QString &key, float &value)`
Retrieve a single parameter.

- bool [operator\(\)](#) (const **QString** §ion, const **QString** &key, int &value)
Retrieve a single parameter.
- bool [operator\(\)](#) (const **QString** §ion, const **QString** &key, bool &value)
Retrieve a single parameter.
- template<typename T >
bool [operator\(\)](#) (const **QString** §ion, const **QString** &key, T &value)
This operator retrieve a single parameter.
- [Parms](#) (const **QString** &parmFile, int verboseLevel=1)
Constructor of the parameter file.
- void [verboseLevel](#) (int vl)
Change the verbosity level.

17.43.1 Detailed Description

A utility class to parse L-Studio like parameter files.

Format of the parameter file

The basic information in the parameter file is a key associated to a value, possibly with a C++-like comment:

```
key: value // Comment
```

Keys can be organized in sections:

```
[Section1]
key1: value1
key2: value2
...

[Section2]
key1: value3
key2: value4
...
```

Empty line or comment-only line are ignored, any other line will raise an error. The default section is named with an empty string "".

Usage example of util::Parm

Here is an example:


```
int a,b;
QString s1, s2;
std::vector<double> all_d;
util::Parms parms( "view.v", 2 );

// First read simple parameters
parms( "", "a", a );
parms( "Main", "b", b, 0 );
parms( "Main", "string1", s1 );
parms( "Main", "string2", s2 );

// Then read all the keys "double" in section "Values"
parms.all( "Values", "double", all_d);
```

Reading typed parameters

To read user-defined typed as parameter, it is enough to overload the function

```
QTextStream& operator>>( QTextStream&, const T& ).
```

T being the type of the parameter to read.

Key duplicates

If the same key is used many times in the same section, either all the values can be retrieved using the [Parms::all\(\)](#) method, or only the last can be retrieved using the normal [Parms::operator\(\)](#).

Verbosity

The class accepts 5 verbosity levels:

- 0: No output
- 1: Errors only
- 2: Errors and warning
- 3: User information
- 4: Debug information

Debug information output the raw string before evaluation to set up a parameter.

Definition at line 105 of file parms.h.

17.43.2 Constructor & Destructor Documentation

17.43.2.1 util::Parms::Parms (const QString & *parmFile*, int *verboseLevel* = 1)

Constructor of the parameter file.

Definition at line 27 of file parms.cpp.

References verboseLevel().

```

28      : ParmFileName( parmFile )
29      {
30          verboseLevel( vl );
31          init();
32      }
```

17.43.3 Member Function Documentation

17.43.3.1 bool util::Parms::all (const QString & *section*, std::map< QString, QStringList > & *value*)

Retrieve a all parameters with same [section]key.

See also

all(const QString& section, std::map<QString, std::vector<T> >& value)

Definition at line 228 of file parms.cpp.

```

229      {
230          return all<QString>( section, value );
231      }
```

17.43.3.2 bool util::Parms::all (const QString & *section*, std::map< QString, std::vector< QString > > & *value*)

Retrieve a all parameters with same [section]key.

See also

all(const QString& section, std::map<QString, std::vector<T> >& value)

Definition at line 223 of file parms.cpp.

```

224      {
225          return all<QString>( section, value );
226      }
```

17.43.3.3 bool util::Parms::all (const QString & *section*, std::map< QString, std::vector< std::string > > & *value*)

Retrieve a all parameters with same [section]key.

See also

all(const QString& section, std::map<QString, std::vector<T> >& value)

17.43.3.4 `bool util::Parms::all (const QString & section, std::map< QString, std::vector< double > > & value)`

Retrieve a all parameters with same [section].

See also

`all(const QString& section, std::map<QString, std::vector<T> >& value)`

Definition at line 218 of file parms.cpp.

```
219  {  
220      return all<double>( section, value );  
221  }
```

17.43.3.5 `bool util::Parms::all (const QString & section, std::map< QString, std::vector< float > > & value)`

Retrieve a all parameters with same [section].

See also

`all(const QString& section, std::map<QString, std::vector<T> >& value)`

Definition at line 213 of file parms.cpp.

```
214  {  
215      return all<float>( section, value );  
216  }
```

17.43.3.6 `bool util::Parms::all (const QString & section, std::map< QString, std::vector< int > > & value)`

Retrieve a all parameters with same [section].

See also

`all(const QString& section, std::map<QString, std::vector<T> >& value)`

Definition at line 208 of file parms.cpp.

```
209  {  
210      return all<int>( section, value );  
211  }
```

17.43.3.7 **bool util::Parms::all (const QString & *section*, std::map< QString, std::vector< bool > > & *value*)**

Retrieve a all parameters with same [*section*].

See also

all(const QString& *section*, std::map<QString, std::vector<T> >& *value*)

Definition at line 203 of file parms.cpp.

```
204  {
205      return all<bool>( section, value );
206  }
```

17.43.3.8 **template<typename T , typename Container > bool util::Parms::all (const QString & *section*, std::map< QString, Container > & *values*) [inline]**

This operator retrieves all parameters with same [*section*].

Parameters

section Section in which the parameter is looked for

value Variable to set up

value is filled with the different keys found in [*section*] and, for each key, the vector of all the values associated to it. If none, *value* is simply empty. The only error that can arise is a reading error, if one parameter has invalid value. This parameter will simply be ignored, all other parameters being read.

Returns

True if there was no error while converting the different parameters from their string representation.

Definition at line 561 of file parms.h.

References QString::clear(), forall, and QString::size().

```
562  {
563      bool valid = true;
564      CheckExist = false;
565      typedef std::map<QString, QStringList>::value_type value_type;
566      result.clear();
567      int pos = section.size()+1;
568      forall( const value_type& pair, Parameters)
569      {
570          QString sec(pair.first.mid(0, pos-1));
571          if(sec != section)
572              continue;
```

```

573     QString key(pair.first.mid(pos));
574     Container& value = result[key];
575     value.clear();
576     forall( const QString& val, pair.second )
577     {
578         T single_value = T();
579         if( readValue( val, single_value ) )
580         {
581             value.push_back( single_value );
582         }
583         else
584         {
585             if( VerboseLevel > 2 )
586             {
587                 err << "Parms::all:Error reading key [" << section << "]" << key
588                     << " with value " << val << endl;
589             }
590             valid = false;
591         }
592     }
593 }
594 CheckExist = true;
595 return valid;
596 }
```

17.43.3.9 bool util::Parms::all (const QString & *section*, const QString & *key*, QStringList & *value*)

Retrieve a all parameters with same [section]key.

See also

all(const QString& section, const QString& key, std::vector<T>& value)

Definition at line 198 of file parms.cpp.

```

199 {
200     return all<QStringList>( section, key, value );
201 }
```

17.43.3.10 bool util::Parms::all (const QString & *section*, const QString & *key*, std::vector< QString > & *value*)

Retrieve a all parameters with same [section]key.

See also

all(const QString& section, const QString& key, std::vector<T>& value)

Definition at line 193 of file parms.cpp.

```

194 {
195     return all<std::vector<QString> >( section, key, value );
196 }
```

17.43.3.11 **bool util::Parms::all (const QString & *section*, const QString & *key*, std::vector< std::string > & *value*)**

Retrieve a all parameters with same [section]key.

See also

all(const QString& section, const QString& key, std::vector<T>& value)

17.43.3.12 **bool util::Parms::all (const QString & *section*, const QString & *key*, std::vector< double > & *value*)**

Retrieve a all parameters with same [section]key.

See also

all(const QString& section, const QString& key, std::vector<T>& value)

Definition at line 188 of file parms.cpp.

```
189  {
190      return all<std::vector<double> >( section, key, value );
191  }
```

17.43.3.13 **bool util::Parms::all (const QString & *section*, const QString & *key*, std::vector< float > & *value*)**

Retrieve a all parameters with same [section]key.

See also

all(const QString& section, const QString& key, std::vector<T>& value)

Definition at line 183 of file parms.cpp.

```
184  {
185      return all<std::vector<float> >( section, key, value );
186  }
```

17.43.3.14 **bool util::Parms::all (const QString & *section*, const QString & *key*, std::vector< int > & *value*)**

Retrieve a all parameters with same [section]key.

See also

all(const QString& section, const QString& key, std::vector<T>& value)

Definition at line 178 of file parms.cpp.

```
179  {
180      return all<std::vector<int> >( section, key, value );
181  }
```

17.43.3.15 bool util::Parms::all (const QString & *section*, const QString & *key*, std::vector< bool > & *value*)

Retrieve a all parameters with same [section]key.

See also

all(const QString& section, const QString& key, std::vector<T>& value)

Definition at line 173 of file parms.cpp.

```
174  {
175      return all<std::vector<bool> >( section, key, value );
176  }
```

17.43.3.16 template<typename Container > bool util::Parms::all (const QString & *section*, const QString & *key*, Container & *value*) [inline]

This operator retrieves all parameters with same [section]key.

Parameters

section Section in which the parameter is looked for

key Key to look for

value Variable to set up

value is filled with the different values found having same [section]key. If none, *value* is simply empty. The only error that can arise is a reading error, if one parameter has invalid value. This parameter will simply be ignored, all other parameters being read.

Returns

True if there was no error while converting the different parameters from their string representation.

Definition at line 529 of file parms.h.

References forall.

Referenced by util::GraphInset::readParms(), cell_system::CellSystem< TissueClass, RealModel >::readRules(), and cell_system::CellSystem< TissueClass, RealModel >::readSymbols().

```

530 {
531     bool valid = true;
532     typedef typename Container::value_type T;
533     CheckExist = false;
534     QStringList values;
535     if( !extractValues( section, key, values ) )
536         return false;
537     value.clear();
538     std::insert_iterator<Container> it(value, value.end());
539     forall( const QString& val, values )
540     {
541         T single_value = T();
542         if( readValue( val, single_value ) )
543         {
544             *it++ = (const T&)single_value;
545         }
546         else
547         {
548             if( VerboseLevel > 2 )
549             {
550                 err << "Parms::all:Error reading key [" << section << "]" << key
551                     << " with value " << val << "\n" << flush;
552             }
553             valid = false;
554         }
555     }
556     CheckExist = true;
557     return valid;
558 }

```

17.43.3.17 bool util::Parms::isLoaded () const [inline]

Returns true if the parameter object has been correctly loaded.

Definition at line 124 of file parms.h.

```
124 { return loaded; }
```

17.43.3.18 template<typename T > bool util::Parms::operator() (const QString & section, const QString & key, T & value, const T & def) [inline]

Variation on the previous, but if the [section]key is not found, an information message is issued (instead of an error) and value is set up to def.

Definition at line 510 of file parms.h.

```

511 {
512     bool found = true;
513     CheckExist = false;
514     if( !( *this )( section, key, value ) )
515     {
516         found = false;
517         if( VerboseLevel > 2 )
518         {

```



```
519         err << "Parms::operator()::Info key [" << section << "]"
520         << key << " not found, using default value" << endl;
521     }
522     value = def;
523 }
524 CheckExist = true;
525 return found;
526 }
```

17.43.3.19 bool util::Parms::operator() (const QString & *section*, const QString & *key*, QString & *value*)

Retrieve a single parameter.

For string, the value is the whole line with whitespaces and comment stripped before and after the parameter.

See also

[operator\(\)\(const QString& section, const QString& key, T& value \)](#)

Definition at line 168 of file parms.cpp.

```
169 {
170     return operator()<QString>( section, key, value );
171 }
```

17.43.3.20 bool util::Parms::operator() (const QString & *section*, const QString & *key*, std::string & *value*)

Retrieve a single parameter.

For string, the value is the whole line with whitespaces and comment stripped before and after the parameter.

See also

[operator\(\)\(const QString& section, const QString& key, T& value \)](#)

Definition at line 163 of file parms.cpp.

```
164 {
165     return operator()<std::string>( section, key, value );
166 }
```

17.43.3.21 bool util::Parms::operator() (const QString & *section*, const QString & *key*, double & *value*)

Retrieve a single parameter.

See also

[operator\(\)\(const QString& section, const QString& key, T& value \)](#)

Definition at line 158 of file parms.cpp.

```
159  {  
160      return operator()<double>( section, key, value );  
161  }
```

17.43.3.22 bool util::Parms::operator() (const QString & *section*, const QString & *key*, float & *value*)

Retrieve a single parameter.

See also

[operator\(\)\(const QString& section, const QString& key, T& value \)](#)

Definition at line 153 of file parms.cpp.

```
154  {  
155      return operator()<float>( section, key, value );  
156  }
```

17.43.3.23 bool util::Parms::operator() (const QString & *section*, const QString & *key*, int & *value*)

Retrieve a single parameter.

See also

[operator\(\)\(const QString& section, const QString& key, T& value \)](#)

Definition at line 148 of file parms.cpp.

```
149  {  
150      return operator()<int>( section, key, value );  
151  }
```

17.43.3.24 bool util::Parms::operator() (const QString & *section*, const QString & *key*, bool & *value*)

Retrieve a single parameter.

Boolean value must read "true" or "false", case being meaningless.

See also

[operator\(\)\(const QString& section, const QString& key, T& value \)](#)

Definition at line 143 of file parms.cpp.

```
144 {
145     return operator()<bool>( section, key, value );
146 }
```

17.43.3.25 template<typename T > bool util::Parms::operator() (const QString & section, const QString & key, T & value) [inline]

This operator retrieve a single parameter.

Parameters

section Section in which the parameter is looked for

key Key to look for

value Variable to set up if possible

If the [section]key exists, value is set up to the parameter value. Any key placed before the first section is considered in the first section. If the [section]key is multiply defined, only the last one is taken and a warning is issued (see [Verbosity](#)). If the [section]key does not exist, or its content cannot be interpreted as the requested type, then an error is issued.

Returns

True if there was no error while converting the different parameters from their string representation and the [section]key exists.

Definition at line 485 of file parms.h.

```
486 {
487     QStringList values;
488     if( !extractValues( section, key, values ) )
489         return false;
490
491     if( ( values.size() > 1 ) && ( VerboseLevel > 1 ) )
492     {
493         err << "Parms::operator():Warning multiple value for key [" << section << "
494             << key << ", last one used.\n" << flush;
495     }
496
497     if( !readValue( values.back(), value ) )
498     {
499         if( VerboseLevel > 0 )
500         {
501             err << "Parms::operator():Error getting value for key [" << section << "
502                 << key
503                 << " value " << values.back() << "\n" << flush;
```

```
503     }
504     return false;
505 }
506 return true;
507 }
```

17.43.3.26 void util::Parms::verboseLevel (int vl) [inline]

Change the verbosity level.

See also

[Verbosity](#)

Definition at line 119 of file parms.h.

Referenced by Parms().

```
119 { VerboseLevel = ( vl < 0 ) ? 0 : vl; }
```

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

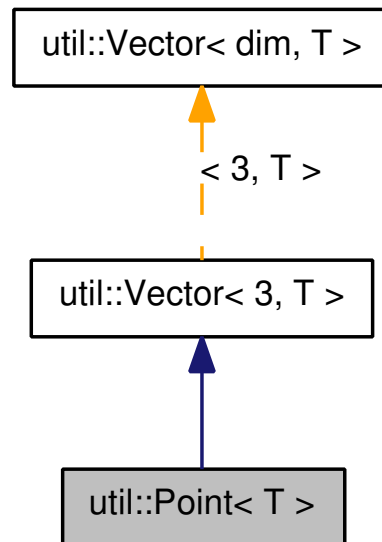
- vvelib/util/[parms.h](#)
- vvelib/util/parms.cpp

17.44 util::Point< T > Class Template Reference

A 3D point/vector class.

```
#include <point.h>
```

Inheritance diagram for util::Point< T >:



Public Member Functions

- `const T * c_data () const`
Return the data as a C-style array.
- `Point< T > cross (const Point< T > &p) const`
- `T distance (const Point< T > &p) const`
- `T distance_sq (const Point< T > &p) const`
- `T length () const`
- `T length_sq () const`
- `void normalise (T l)`
- `void normalise ()`
- `Point & operator*= (const T &p)`
In-place multiplication by a scalar.
- `Point operator+ (const Point &p) const`
- `Point & operator+= (const Point &p)`
- `Point operator- () const`
Vector negation.
- `Point operator- (const Point &p) const`

- **Point** & **operator-=** (const **Point** &p)
- **Point** **operator/** (const T &p) const
Division by a scalar.
- **Point** & **operator/=** (const T &p)
In-place division by a scalar.
- **Point** (const T &x=T(), const T &y=T(), const T &z=T())
- **Point** (const **Vector**< 3, T > &v)
Constructor.
- **Point**< T > **proj** (const **Point**< T > &p) const
Vector projection.
- T **proj_length** (const **Point**< T > &p) const
Vector projection length.
- virtual **~Point** ()
Destructor.

17.44.1 Detailed Description

template<class T> class util::Point< T >

A 3D point/vector class. The **Point** class is a utility class to handle operations on three-dimensional xyz-points and vectors. For some functions, T must be reducible to a floating point type to satisfy the functions used in the math library.

Definition at line 25 of file point.h.

17.44.2 Constructor & Destructor Documentation

17.44.2.1 **template<class T> util::Point< T >::Point (const Vector< 3, T > &v) [inline]**

Constructor.

Definition at line 28 of file point.h.

```
28 : Vector<3,T>( v ) {}
```

17.44.2.2 **template<class T> virtual util::Point< T >::~~Point () [inline, virtual]**

Destructor.

Definition at line 32 of file point.h.

```
32 {}
```

17.44.3 Member Function Documentation

17.44.3.1 `template<class T> const T* util::Point< T >::c_data () const` `[inline]`

Return the data as a C-style array.

Reimplemented from [util::Vector< 3, T >](#).

Definition at line 53 of file point.h.

```
53 {return this->elems;}
```

17.44.3.2 `template<class T> Point& util::Point< T >::operator*= (const T & scalar)` `[inline]`

In-place multiplication by a scalar.

Reimplemented from [util::Vector< 3, T >](#).

Definition at line 46 of file point.h.

```
46 { this->Vector<3,T>::operator*=( p ); return *this; }
```

17.44.3.3 `template<class T> Point util::Point< T >::operator- (void) const` `[inline]`

[Vector](#) negation.

Reimplemented from [util::Vector< 3, T >](#).

Definition at line 51 of file point.h.

```
51 { return this->Vector<3,T>::operator-(); }
```

17.44.3.4 `template<class T> Point util::Point< T >::operator/ (const T & scalar) const` `[inline]`

Division by a scalar.

Reimplemented from [util::Vector< 3, T >](#).

Definition at line 50 of file point.h.

```
50 { return this->Vector<3,T>::operator/( p ); }
```

17.44.3.5 `template<class T> Point& util::Point< T >::operator/= (const T & scalar) [inline]`

In-place division by a scalar.

Reimplemented from [util::Vector< 3, T >](#).

Definition at line 47 of file point.h.

```
47 { this->Vector<3,T>::operator/=( p ); return *this; }
```

17.44.3.6 `template<class T > Point< T > util::Point< T >::proj (const Point< T > & p) const [inline]`

[Vector](#) projection.

Parameters

p The vector projected onto.

Definition at line 83 of file point.h.

```
83                                     {  
84     return ((*this * p) / p.length_sq()) * p;  
85 }
```

17.44.3.7 `template<class T > T util::Point< T >::proj_length (const Point< T > & p) const [inline]`

[Vector](#) projection length.

Parameters

p The vector projected onto.

Definition at line 92 of file point.h.

```
92                                     {  
93     return fabs(*this * p) / p.length();  
94 }
```

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

- [vvelib/util/point.h](#)

17.45 algorithms::TriangleSurface::Position Struct Reference

Structure describing the position of a point on a triangulated surface, relatively to this surface.

```
#include <triangle_growth.h>
```

Public Member Functions

- **operator bool** () const
- **Position** (const [Position](#) ©)
- **Position** (const Point3d &bar, int vv1, int vv2, int vv3, int f)
- **bool serialize** ([storage::VVEStorage](#) &)

Public Attributes

- Point3d **barycentric**
- int **face**
- int **v1**
- int **v2**
- int **v3**

17.45.1 Detailed Description

Structure describing the position of a point on a triangulated surface, relatively to this surface.

Definition at line 69 of file triangle_growth.h.

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

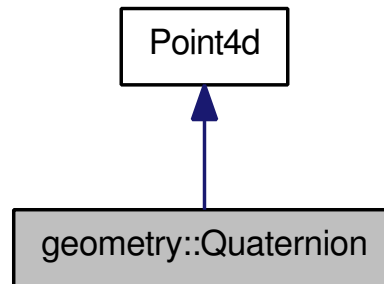
- vvelib/algorithms/[triangle_growth.h](#)
- vvelib/algorithms/triangle_growth.cpp

17.46 geometry::Quaternion Class Reference

Implements the quaternion operations.

```
#include <geometry/quaternion.h>
```

Inheritance diagram for geometry::Quaternion:



Public Member Functions

- `double angle () const`
Returns the angle of the rotation corresponding to this quaternion.
- `Point3d axis () const`
Returns the axis of the rotation corresponding to this quaternion.
- `Quaternion conjugate () const`
Return the conjugate of the current quaternion.
- `Quaternion inverse () const`
Return the quaternion corresponding to the inverse transform.
- `Point3d inverseRotate (const Point3d &v) const`
Apply the inverse of the rotation contained in this quaternion on the vector.
- `Quaternion operator* (const Quaternion &other) const`
Quaternion multiplication.
- `Quaternion & operator*= (double s)`
In-place multiplication of a quaternion by a scalar.
- `Quaternion & operator*= (const Quaternion &other)`
Quaternion in-place multiplication.
- `Quaternion operator+ (const Quaternion &other) const`
Quaternion addition.

- **Quaternion** & **operator+=** (const **Quaternion** &other)
Quaternion in-place addition.
- **Quaternion** **operator/** (double v) const
Division of a quaternion by a real number.
- **Quaternion** & **operator/=** (double v)
Division of a quaternion by a real number.
- **Quaternion** & **operator=** (const **Quaternion** &other)
Assignment operator for quaternions.
- **Quaternion** (const **Matrix3d** &m)
Creates the quaternion corresponding to the rotation matrix m.
- **Quaternion** (const **Point3d** &from, const **Point3d** &to)
Creates the quaternion corresponding to the rotation transforming from into to.
- **Quaternion** (const **Point3d** &axis, double angle)
Creates a Quaternion corresponding to an axis rotation.
- **Quaternion** (const **Quaternion** &other)
Copy constructor.
- **Quaternion** (double x, double y, double z, double w)
Creates a quaternion specified by its components.
- **Quaternion** ()
Default constructor.
- **Point3d** **rotate** (const **Point3d** &v) const
Apply the rotation contained in this quaternion on the vector.
- void **setAxisAngle** (const **Point3d** &axis, double angle)
Set the quaternion to the described rotation.
- void **setMatrix** (**Matrix4d** &m) const
Fill the matrix as argument from the quaternion.
- void **setMatrix** (**Matrix3d** &m) const
Fill the matrix as argument from the quaternion.
- const double & **w** () const
Accessing the real part of the quaternion.

- `double & w ()`

Accessing the real part of the quaternion.

17.46.1 Detailed Description

Implements the quaternion operations.

Definition at line 17 of file quaternion.h.

17.46.2 Constructor & Destructor Documentation

17.46.2.1 `geometry::Quaternion::Quaternion () [inline]`

Default constructor.

Provides an identity quaternion

Definition at line 24 of file quaternion.h.

Referenced by `conjugate()`, `inverse()`, and `Quaternion()`.

```
25         : Point4d(0,0,0,1)
26         {}
```

17.46.2.2 `geometry::Quaternion::Quaternion (double x, double y, double z, double w) [inline]`

Creates a quaternion specified by its components.

Definition at line 31 of file quaternion.h.

```
32         : Point4d(x,y,z,w)
33         {}
```

17.46.2.3 `geometry::Quaternion::Quaternion (const Quaternion & other) [inline]`

Copy constructor.

Definition at line 38 of file quaternion.h.

```
39         : Point4d(other)
40         {}
```

17.46.2.4 Quaternion::Quaternion (const Point3d & axis, double angle)

Creates a [Quaternion](#) corresponding to an axis rotation.

Parameters

axis Axis of the rotation. It needs not be normalized before hand. If it is null, then the [Quaternion](#) will correspond to the identity matrix.

angle Angle of the rotation.

Definition at line 55 of file quaternion.cpp.

```
59      : Point4d()
```

17.46.2.5 Quaternion::Quaternion (const Point3d & from, const Point3d & to)

Creates the quaternion corresponding to the rotation transforming *from* into *to*.

Definition at line 4 of file quaternion.cpp.

References [Quaternion\(\)](#).

```
6 {
7   Quaternion::Quaternion(const Point3d& from, const Point3d& to)
8       : Point4d(0,0,0,1)
9   {
10      const double fromSqNorm = util::normsq(from);
11      const double toSqNorm = util::normsq(to);
12      // Identity Quaternion when one vector is null
13      if ((fromSqNorm < VVE_EPSILON) || (toSqNorm < VVE_EPSILON))
14      {
15          return;
16      }
17      else
18      {
19          Point3d axis = from ^ to;
20          const double axisSqNorm = util::normsq(axis);
21
22          // Aligned vectors, pick any axis, not aligned with from or to
23          if (axisSqNorm < VVE_EPSILON)
24              axis = util::orthogonal(from);
25
26          double angle = std::asin(std::sqrt(axisSqNorm / (fromSqNorm * toSqNorm)));
27
28          if (from*to < 0.0)
29              angle = M_PI-angle;
30      }
```

17.46.2.6 Quaternion::Quaternion (const Matrix3d & m)

Creates the quaternion corresponding to the rotation matrix *m*.

Definition at line 81 of file quaternion.cpp.

```

85  {
86      // Compute one plus the trace of the matrix
87      const double onePlusTrace = 1.0 + m.trace();
88      if(onePlusTrace > VVE_EPSILON)
89      {
90          const double s = std::sqrt(onePlusTrace)*2.0;
91          x() = (m(2,1) - m(1,2))/s;
92          y() = (m(0,2) - m(2,0))/s;
93          z() = (m(1,0) - m(0,1))/s;
94          w() = 0.25*s;
95      }
96      else
97      {
98          // Computation depends on major diagonal term
99          if(m(0,0) > m(1,1) and m(0,0) > m(2,2))
100         {
101             const double s = std::sqrt(1+m(0,0)-m(1,1)-m(2,2))*2.0;
102             x() = 0.25*s;
103             y() = (m(1,0)+m(0,1))/s;
104             z() = (m(2,0)+m(0,2))/s;
105             w() = (m(1,2)-m(2,1))/s;
106         }
107         else if(m(1,1) > m(2,2))
108         {
109             const double s = std::sqrt(1+m(1,1)-m(0,0)-m(2,2))*2.0;
110             x() = (m(0,1)+m(1,0))/s;
111             y() = 0.25*s;
112             z() = (m(1,2)+m(2,1))/s;
113             w() = (m(0,2)-m(2,0))/s;
114         }
115         else
116         {
117             const double s = std::sqrt(1+m(2,2)-m(0,0)-m(1,1))*2.0;
118             x() = (m(0,2)+m(0,0))/s;
119             y() = (m(1,2)+m(2,1))/s;
120             z() = 0.25*s;
121             w() = (m(0,1)-m(1,0))/s;
122         }

```

17.46.3 Member Function Documentation

17.46.3.1 double Quaternion::angle() const

Returns the angle of the rotation corresponding to this quaternion.

Definition at line 174 of file quaternion.cpp.

```

174                                     :-res;
175     }
176
177     double Quaternion::angle() const
178     {

```

17.46.3.2 Point3d Quaternion::axis() const

Returns the axis of the rotation corresponding to this quaternion.

Definition at line 165 of file quaternion.cpp.

```
169 {
170     Point3d res(x(), y(), z());
171     const double sin = res.norm();
172     if(sin > VVE_EPSILON)
```

17.46.3.3 Quaternion geometry::Quaternion::conjugate() const [inline]

Return the conjugate of the current quaternion.

Definition at line 126 of file quaternion.h.

References Quaternion(), w(), util::Vector< dim, T >::x(), util::Vector< dim, T >::y(), and util::Vector< dim, T >::z().

```
126 { return Quaternion(-x(), -y(), -z(), w()); }
```

17.46.3.4 Quaternion geometry::Quaternion::inverse() const [inline]

Return the quaternion corresponding to the inverse transform.

Definition at line 121 of file quaternion.h.

References util::Vector< dim, T >::normsq(), Quaternion(), w(), util::Vector< dim, T >::x(), util::Vector< dim, T >::y(), and util::Vector< dim, T >::z().

```
121 { double n = util::normsq(*this); return Quaternion(-x()/n, -y()/n, -z()/n, w()/n
    ); }
```

17.46.3.5 Point3d Quaternion::inverseRotate (const Point3d & v) const

Apply the inverse of the rotation contained in this quaternion on the vector.

It is identical to calling `this->inverse().rotate()`

Definition at line 204 of file quaternion.cpp.

```
208 {
```

17.46.3.6 Quaternion Quaternion::operator* (const Quaternion & other) const

[Quaternion](#) multiplication.

Definition at line 38 of file quaternion.cpp.

```
42 {
43     double w_ = w()*other.w() - x()*other.x() -
44         y()*other.y() - z()*other.z();
45     double x_ = w()*other.x() + other.w()*x() + y()*other.z() - z()*other.y();
46     double y_ = w()*other.y() + other.w()*y() + z()*other.x() - x()*other.z();
```

17.46.3.7 Quaternion& geometry::Quaternion::operator*= (double s) **[inline]**

In-place multiplication of a quaternion by a scalar.

Definition at line 115 of file quaternion.h.

References `util::Vector< dim, T >::i()`.

```
116     { for(size_t i = 0 ; i < 4 ; ++i) elems[i] *= s; return *this;}
```

17.46.3.8 Quaternion & Quaternion::operator*= (const Quaternion & other)

[Quaternion](#) in-place multiplication.

Definition at line 48 of file quaternion.cpp.

```
52     {
53         Quaternion res = *this * other;
```

17.46.3.9 Quaternion geometry::Quaternion::operator+ (const Quaternion & other) const **[inline]**

[Quaternion](#) addition.

Definition at line 96 of file quaternion.h.

```
97     {
98         Quaternion tmp(*this);
99         tmp += other;
100        return tmp;
101    }
```

17.46.3.10 Quaternion& geometry::Quaternion::operator+= (const Quaternion & other) **[inline]**

[Quaternion](#) in-place addition.

Definition at line 87 of file quaternion.h.

References `util::Vector< dim, T >::i()`.

```
88     {
89         for(size_t i = 0 ; i < 4 ; ++i) elems[i] += other.elems[i];
90        return *this;
91    }
```


**17.46.3.11 Quaternion geometry::Quaternion::operator/ (double *v*) const
[inline]**

Division of a quaternion by a real number.

Definition at line 140 of file quaternion.h.

```

141     {
142         Quaternion tmp(*this);
143         tmp /= v;
144         return tmp;
145     }
```

**17.46.3.12 Quaternion& geometry::Quaternion::operator/= (double *v*)
[inline]**

Division of a quaternion by a real number.

Definition at line 131 of file quaternion.h.

References util::Vector< dim, T >::i().

```

132     {
133         for(size_t i = 0 ; i < 4 ; ++i) elems[i] /= v;
134         return *this;
135     }
```

17.46.3.13 Quaternion & Quaternion::operator= (const Quaternion & *other*)

Assignment operator for quaternions.

Definition at line 32 of file quaternion.cpp.

```

36     {
```

17.46.3.14 Point3d Quaternion::rotate (const Point3d & *v*) const

Apply the rotation contained in this quaternion on the vector.

Definition at line 180 of file quaternion.cpp.

```

180                                     : 2.0*M_PI-a;
181     }
182
183     Point3d Quaternion::rotate(const Point3d& v) const
184     {
185         /*
186         Quaternion q(v.x(), v.y(), v.z(), 0);
187         Quaternion result = *this * q * conjugate();
188         return Point3d(result.x(), result.y(), result.z());
189         */
```

```

190
191     const double xx = 2.01*x()*x();
192     const double yy = 2.01*y()*y();
193     const double zz = 2.01*z()*z();
194     const double xy = 2.01*x()*y();
195     const double xz = 2.01*x()*z();
196     const double yz = 2.01*y()*z();
197
198     const double wx = 2.01*w()*x();
199     const double wy = 2.01*w()*y();
200     const double wz = 2.01*w()*z();
201
202     return Point3d((1.0-yy-zz)*v.x() + (    xy-wz)*v.y() + (    xz+wy)*v.z(),

```

17.46.3.15 void Quaternion::setAxisAngle (const Point3d & axis, double angle)

Set the quaternion to the described rotation.

Parameters

axis Axis of the rotation

angle Angle of the rotation

Definition at line 61 of file quaternion.cpp.

```

65     {
66         const double norm = util::norm(axis);
67         if (norm < VVE_EPSILON)
68         {
69             x() = 0;
70             y() = 0;
71             z() = 0;
72             w() = 1;
73         }
74         else
75         {
76             const double sin_half_angle = std::sin(angle/2.0);
77             x() = sin_half_angle*axis.x()/norm;
78             y() = sin_half_angle*axis.y()/norm;
79             z() = sin_half_angle*axis.z()/norm;

```

17.46.3.16 void geometry::Quaternion::setMatrix (Matrix4d & m) const

Fill the matrix as argument from the quaternion.

Multiplying with this matrix is equivalent to performing a rotation with this quaternion.

17.46.3.17 void Quaternion::setMatrix (Matrix3d & m) const

Fill the matrix as argument from the quaternion.

Multiplying with this matrix is equivalent to performing a rotation with this quaternion.

Definition at line 149 of file quaternion.cpp.

```
153 {
```

17.46.3.18 const double& geometry::Quaternion::w () const [inline]

Accessing the real part of the quaternion.

Definition at line 82 of file quaternion.h.

```
82 { return elems[3]; }
```

17.46.3.19 double& geometry::Quaternion::w () [inline]

Accessing the real part of the quaternion.

Definition at line 78 of file quaternion.h.

Referenced by conjugate(), and inverse().

```
78 { return elems[3]; }
```

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

- vvelib/geometry/[quaternion.h](#)
- vvelib/geometry/quaternion.cpp

17.47 `util::range< Iterator >` Class Template Reference

Class representing a range of values from two iterators.

```
#include <util/range.h>
```

Public Types

- typedef Iterator **const_iterator**
- typedef std::iterator_traits< **iterator** >::difference_type **difference_type**
- typedef Iterator **iterator**

Public Member Functions

- **iterator** `begin` () const
Begin of the range.
- **const_iterator** `cbegin` () const
Begin of the range.
- **const_iterator** `cend` () const
Past-the-End of the range.
- **iterator** `end` () const
Past-the-End of the range.
- template<typename It >
`range & operator=` (const `range`< It > &other)
Assignment operator.
- `range & operator=` (const `range` &other)
Assignment operator.
- template<typename It >
`range` (const `range`< It > ©)
Conversion constructor.
- `range` (const `range` ©)
Copy constructor.
- `range` (const std::pair< **iterator**, **iterator** > &p)
Range from a pair of iterators.
- `range` (const **iterator** &fst, const **iterator** &lst)

Range from two iterators.

- `range ()`
Create an invalid range.
- `void setBegin (const iterator &it)`
Set the start of the range.
- `void setEnd (const iterator &it)`
Set the past-the-end iterator for the range.
- `difference_type size () const`
Return the size of the range.

Public Attributes

- `iterator first`
- `iterator last`

17.47.1 Detailed Description

`template<typename Iterator> class util::range< Iterator >`

Class representing a range of values from two iterators.

Definition at line 21 of file range.h.

17.47.2 Constructor & Destructor Documentation

17.47.2.1 `template<typename Iterator> util::range< Iterator >::range ()`
`[inline]`

Create an invalid range.

Definition at line 30 of file range.h.

```
31         : first()
32         , last()
33         {}
```

17.47.2.2 `template<typename Iterator> util::range< Iterator >::range (const`
`iterator &fst, const iterator &lst) [inline]`

Range from two iterators.

Definition at line 38 of file range.h.

```

40      : first(fst)
41      , last(lst)
42      {}

```

17.47.2.3 `template<typename Iterator> util::range< Iterator >::range (const std::pair< iterator, iterator > &p) [inline, explicit]`

Range from a pair of iterators.

Definition at line 47 of file range.h.

```

48      : first(p.first)
49      , last(p.second)
50      {}

```

17.47.2.4 `template<typename Iterator> util::range< Iterator >::range (const range< Iterator > ©) [inline]`

Copy constructor.

Definition at line 55 of file range.h.

```

56      : first(copy.first)
57      , last(copy.last)
58      {}

```

17.47.2.5 `template<typename Iterator> template<typename It > util::range< Iterator >::range (const range< It > ©) [inline, explicit]`

Conversion constructor.

Definition at line 64 of file range.h.

```

65      : first(copy.first)
66      , last(copy.last)
67      { }

```

17.47.3 Member Function Documentation

17.47.3.1 `template<typename Iterator> iterator util::range< Iterator >::begin () const [inline]`

Begin of the range.

Definition at line 91 of file range.h.

Referenced by vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::deleteCell().

```
91 { return first; }
```

17.47.3.2 template<typename Iterator> const_iterator util::range< Iterator >::cbegin() const [inline]

Begin of the range.

Definition at line 100 of file range.h.

```
100 { return first; }
```

17.47.3.3 template<typename Iterator> const_iterator util::range< Iterator >::cend() const [inline]

Past-the-End of the range.

Definition at line 104 of file range.h.

```
104 { return last; }
```

17.47.3.4 template<typename Iterator> iterator util::range< Iterator >::end() const [inline]

Past-the-End of the range.

Definition at line 95 of file range.h.

Referenced by vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::deleteCell().

```
95 { return last; }
```

17.47.3.5 template<typename Iterator> template<typename It > range& util::range< Iterator >::operator=(const range< It > & other) [inline]

Assignment operator.

Definition at line 138 of file range.h.

```
139     {
140         first = other.first;
141         last = other.last;
142         return *this;
143     }
```

17.47.3.6 `template<typename Iterator> range& util::range< Iterator >::operator= (const range< Iterator > & other) [inline]`

Assignment operator.

Definition at line 127 of file range.h.

```
128     {
129         first = other.first;
130         last = other.last;
131         return *this;
132     }
```

17.47.3.7 `template<typename Iterator> void util::range< Iterator >::setBegin (const iterator & it) [inline]`

Set the start of the range.

Definition at line 109 of file range.h.

```
109 { first = it; }
```

17.47.3.8 `template<typename Iterator> void util::range< Iterator >::setEnd (const iterator & it) [inline]`

Set the past-the-end iterator for the range.

Definition at line 115 of file range.h.

```
115 { last = it; }
```

17.47.3.9 `template<typename Iterator> difference_type util::range< Iterator >::size () const [inline]`

Return the size of the range.

Complexity depends on the iterator.

Definition at line 122 of file range.h.

```
122 { return std::distance(first, last); }
```

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

- [vvelib/util/range.h](#)

17.48 `util::refpair< T, U >` Class Template Reference

Class used to hold references for the `util::tie()` function.

```
#include <util/tie.h>
```

Public Types

- typedef T **first_type**
- typedef U **second_type**

Public Member Functions

- `refpair` & `operator=` (std::pair< first_type, second_type > const &p)
Assign the values of p to the references in this pair.
- `refpair` (refpair const &rp)
Construct a copy.
- `refpair` (T &x, U &y)
Construct a pair of references to x and y.

Public Attributes

- T & `first`
The first member of the pair.
- U & `second`
The second member of the pair.

17.48.1 Detailed Description

```
template<typename T, typename U> class util::refpair< T, U >
```

Class used to hold references for the `util::tie()` function.

Definition at line 24 of file tie.h.

17.48.2 Constructor & Destructor Documentation

17.48.2.1 `template<typename T, typename U> util::refpair< T, U >::refpair`
(T &x, U &y) [`inline`]

Construct a pair of references to x and y.

Definition at line 30 of file tie.h.

```
30 : first(x), second(y) {}
```

17.48.2.2 `template<typename T, typename U> util::refpair< T, U >::refpair (refpair< T, U > const & rp) [inline]`

Construct a copy.

Definition at line 32 of file tie.h.

```
32 : first(rp.first), second(rp.second) {}
```

17.48.3 Member Function Documentation

17.48.3.1 `template<typename T, typename U> refpair& util::refpair< T, U >::operator=(std::pair< first_type, second_type > const & p) [inline]`

Assign the values of p to the references in this pair.

Definition at line 37 of file tie.h.

References util::refpair< T, U >::first, and util::refpair< T, U >::second.

```
38      {
39          first = p.first;
40          second = p.second;
41          return *this;
42      }
```

17.48.4 Member Data Documentation

17.48.4.1 `template<typename T, typename U> T& util::refpair< T, U >::first`

The first member of the pair.

Definition at line 54 of file tie.h.

Referenced by util::refpair< T, U >::operator=().

17.48.4.2 `template<typename T, typename U> U& util::refpair< T, U >::second`

The second member of the pair.

Definition at line 56 of file tie.h.

Referenced by util::refpair< T, U >::operator=().

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

- [vvelib/util/tie.h](#)

17.49 graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator > Struct Template Reference

Type of the result of the search for a vertex in a neighborhood.

```
#include <vvgraph.h>
```

Public Member Functions

- [operator bool \(\)](#)
Convert the object to true if the search is successful.
- [search_result_t](#) (const [search_result_t](#) ©)
Copy constructor.
- [search_result_t](#) (Iterator i, Neighborhood *n, bool ok=true)
Successful constructor.
- [search_result_t](#) ()
Default constructor.

Public Attributes

- bool [found](#)
True if the search was completely successful.
- Iterator [it](#)
Iterator pointing in the edge list.
- Neighborhood * [neighborhood](#)
Neighborhood containing the element.

17.49.1 Detailed Description

```
template<typename VertexContent, typename EdgeContent = _-
EmptyEdgeContent, bool compact = false>template<class Neighborhood,
class Iterator> struct graph::VVGraph< VertexContent, EdgeContent, compact
>::search_result_t< Neighborhood, Iterator >
```

Type of the result of the search for a vertex in a neighborhood. This data structure is made such that all edit or lookup operations can be done from that data structure.

Definition at line 1665 of file vvgraph.h.

17.49.2 Constructor & Destructor Documentation

17.49.2.1 `template<typename VertexContent, typename EdgeContent =
_EmptyEdgeContent, bool compact = false> template<class
Neighborhood, class Iterator > graph::VVGraph< VertexContent,
EdgeContent, compact >::search_result_t< Neighborhood, Iterator
>::search_result_t() [inline]`

Default constructor.

By default, the search is unsuccessful

Definition at line 1672 of file vvgraph.h.

```
1673             : found(false)
1674             , neighborhood(0) {}
```

17.49.2.2 `template<typename VertexContent, typename EdgeContent =
_EmptyEdgeContent, bool compact = false> template<class
Neighborhood, class Iterator > graph::VVGraph< VertexContent,
EdgeContent, compact >::search_result_t< Neighborhood, Iterator
>::search_result_t(Iterator i, Neighborhood * n, bool ok = true)
[inline]`

Successful constructor.

Definition at line 1679 of file vvgraph.h.

```
1682             : found(ok)
1683             , it(i)
1684             , neighborhood(n)
1685             {}
```

17.49.2.3 `template<typename VertexContent, typename EdgeContent =
_EmptyEdgeContent, bool compact = false> template<class
Neighborhood, class Iterator > graph::VVGraph< VertexContent,
EdgeContent, compact >::search_result_t< Neighborhood, Iterator
>::search_result_t(const search_result_t< Neighborhood, Iterator >
& copy) [inline]`

Copy constructor.

Definition at line 1690 of file vvgraph.h.

```
1691             : found(copy.found)
1692             , it(copy.it)
1693             , neighborhood(copy.neighborhood)
1694             {}
```

17.49.3 Member Function Documentation

17.49.3.1 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> template<class Neighborhood, class Iterator > graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::operator bool () [inline]`

Convert the object to true if the search is successful.

Definition at line 1699 of file vvgraph.h.

```
1699 { return found; }
```

17.49.4 Member Data Documentation

17.49.4.1 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> template<class Neighborhood, class Iterator > bool graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::found`

True if the search was completely successful.

Definition at line 1704 of file vvgraph.h.

17.49.4.2 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> template<class Neighborhood, class Iterator > Iterator graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`

Iterator pointing in the edge list.

Definition at line 1708 of file vvgraph.h.

Referenced by `graph::VVGraph< VertexContent, EdgeContent, compact >::arc()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::edge()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::eraseAllEdges()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::eraseEdge()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::findIn()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::flag()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::neighbors()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::nextTo()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::prevTo()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::replace()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::spliceAfter()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::spliceBefore()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::subgraph()`.

**17.49.4.3 template<typename VertexContent, typename EdgeContent =
 _EmptyEdgeContent, bool compact = false> template<class
 Neighborhood , class Iterator > Neighborhood* graph::VVGraph<
 VertexContent, EdgeContent, compact >::search_result_t<
 Neighborhood, Iterator >::neighborhood**

Neighborhood containing the element.

Definition at line 1712 of file vvgraph.h.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact
>::eraseAllEdges(), graph::VVGraph< VertexContent, EdgeContent, com-
pact >::eraseEdge(), graph::VVGraph< VertexContent, EdgeContent, com-
pact >::findIn(), graph::VVGraph< VertexContent, EdgeContent, com-
pact >::flag(), graph::VVGraph< VertexContent, EdgeContent, compact
>::neighbors(), graph::VVGraph< VertexContent, EdgeContent, compact
>::nextTo(), graph::VVGraph< VertexContent, EdgeContent, compact >::prevTo(),
graph::VVGraph< VertexContent, EdgeContent, compact >::spliceAfter(), and
graph::VVGraph< VertexContent, EdgeContent, compact >::spliceBefore().

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

- [vvelib/graph/vvgraph.h](#)

17.50 graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator > Struct Template Reference

Type of the result of the search for a vertex in a neighborhood.

```
#include <vvbigraph.h>
```

Public Member Functions

- [operator bool \(\)](#)
Convert the object to true if the search is successful.
- [search_result_t](#) (const [search_result_t](#) ©)
Copy constructor.
- [search_result_t](#) (Iterator i, Neighborhood *n, bool ok=true)
Successful constructor.
- [search_result_t \(\)](#)
Default constructor.

Public Attributes

- bool [found](#)
True if the search was completely successful.
- Iterator [it](#)
Iterator pointing in container.
- Neighborhood * [neighborhood](#)
Container where the element is found.

17.50.1 Detailed Description

```
template<typename Vertex1Content, typename Vertex2Content, type-
name Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ =
Edge1Content, bool compact = false>template<class Neighborhood, class
Iterator> struct graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood,
Iterator >
```

Type of the result of the search for a vertex in a neighborhood. This data structure is made such that all edit or lookup operations can be done from that data structure.

Definition at line 1625 of file vvbigraph.h.

17.50.2 Constructor & Destructor Documentation

17.50.2.1 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false>template<class Neighborhood, class Iterator > graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::search_result_t() [inline]`

Default constructor.

By default, the search is unsuccessful

Definition at line 1632 of file vvbigraph.h.

```
1633             : found(false)
1634             , neighborhood(0) {}
```

17.50.2.2 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false>template<class Neighborhood, class Iterator > graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::search_result_t(Iterator i, Neighborhood * n, bool ok = true) [inline]`

Successful constructor.

Definition at line 1639 of file vvbigraph.h.

```
1642             : found(ok)
1643             , it(i)
1644             , neighborhood(n)
1645             {}
```

17.50.2.3 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> template<class Neighborhood, class Iterator > graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::search_result_t (const search_result_t< Neighborhood, Iterator > & copy) [inline]`

Copy constructor.

Definition at line 1650 of file vvbigraph.h.

```
1651             : found(copy.found)
1652             , it(copy.it)
1653             , neighborhood(copy.neighborhood)
1654             {}
```

17.50.3 Member Function Documentation

17.50.3.1 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> template<class Neighborhood, class Iterator > graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::operator bool () [inline]`

Convert the object to true if the search is successful.

Definition at line 1659 of file vvbigraph.h.

References `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::found`.

```
1659 { return found; }
```

17.50.4 Member Data Documentation

17.50.4.1 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> template<class Neighborhood, class Iterator > bool graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::found`

True if the search was completely successful.

Definition at line 1664 of file vvbigraph.h.

17.50 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator > Struct
Template Reference **445**
Referenced by graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::operator bool().

17.50.4.2 template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> template<class Neighborhood, class Iterator > Iterator graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it

Iterator pointing in container.

Definition at line 1668 of file vvbigraph.h.

Referenced by graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::edge(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findIn(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::flag(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::prevTo(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::replace(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceBefore().

17.50.4.3 template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> template<class Neighborhood, class Iterator > Neighborhood* graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood

Container where the element is found.

Definition at line 1672 of file vvbigraph.h.

Referenced by graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content,

Edge2Content_, compact >::findIn(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::flag(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::prevTo(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceBefore().

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

- [vvelib/graph/vvbigraph.h](#)

17.51 `util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >` Class Template Reference

Iterate over a container of structure, dereferencing only a member of it.

```
#include <util/member_iterator.h>
```

Public Types

- typedef `Iterator` [base_iterator](#)
Type of the underlying iterator.
- typedef `std::iterator_traits< Iterator >::difference_type` [difference_type](#)
Type of the difference between two iterators.
- typedef `std::iterator_traits< Iterator >::iterator_category` [iterator_category](#)
Category of the iterator.
- typedef `Pointer` [pointer](#)
Type of a pointer on the values.
- typedef `Reference` [reference](#)
Type of a reference on the values.
- typedef `T` [value_type](#)
Type of the value iterated on.

Public Member Functions

- [base_iterator](#) `base () const`
Direct access to the base iterator.
- [SelectMemberIterator](#) (`const SelectMemberIterator &other`)
Copy constructor.
- [SelectMemberIterator](#) (`const base_iterator &i`)
Conversion from the base iterator.
- [SelectMemberIterator](#) ()
Default constructor.

1 - Forward/input iterator methods

- `bool operator!= (const SelectMemberIterator &other) const`
- `const reference operator* () const`
Constant dereference operator.
- `reference operator* ()`
Dereference operator.
- `SelectMemberIterator & operator++ (int)`
Postfix increment operator.
- `SelectMemberIterator & operator++ ()`
Prefix increment operator.
- `const pointer operator-> () const`
Pointer-like constant arrow operator.
- `pointer operator-> ()`
Pointer-like arrow operator.
- `SelectMemberIterator & operator= (const SelectMemberIterator ©)`
Assignment operator.
- `bool operator== (const SelectMemberIterator &other) const`

3 - Random access methods

- `SelectMemberIterator & operator+= (difference_type n)`
In-place random increment operator.
- `SelectMemberIterator & operator-= (difference_type n)`
In-place random decrement operator.
- `bool operator< (const SelectMemberIterator &other) const`
- `bool operator<= (const SelectMemberIterator &other) const`
- `bool operator> (const SelectMemberIterator &other) const`
- `bool operator>= (const SelectMemberIterator &other) const`

2 - Bidirectional iterator methods

- `SelectMemberIterator & operator-- (int)`
Postfix decrement operator.
- `SelectMemberIterator & operator-- ()`
Prefix decrement operator.

Protected Attributes

- `base_iterator it`
Underlying iterator.

Friends

Functions for random access iterators

- [SelectMemberIterator](#) **operator+** ([difference_type](#) n, const [SelectMemberIterator](#) &it)
- [SelectMemberIterator](#) **operator+** (const [SelectMemberIterator](#) &it, [difference_type](#) n)
- [difference_type](#) **operator-** (const [SelectMemberIterator](#) &last, const [SelectMemberIterator](#) &first)

Distance between two iterators.

- [SelectMemberIterator](#) **operator-** ([difference_type](#) n, const [SelectMemberIterator](#) &it)
- [SelectMemberIterator](#) **operator-** (const [SelectMemberIterator](#) &it, [difference_type](#) n)

17.51.1 Detailed Description

template<class Iterator, class T, T std::iterator_traits< Iterator >::value_type::* member, class Reference = T&, class Pointer = T*> class util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >

Iterate over a container of structure, dereferencing only a member of it. This iterator is used when iterating over a container of a complexe structure. It allows for dereferencing only a member of that structure. A typical example is iterating over a map but dereferencing either the key or the value:

```
std::map<int,double> my_map;
typedef std::map<int,double>::iterator map_iterator;
typedef std::map<int,double>::value_type map_value;
util::SelectMemberIterator<map_iterator,const int, &map_value::first> key_iterat
    or;
util::SelectMemberIterator<map_iterator,int,&map_value::second> data_iterator;

    util::SelectMemberIterator<map_iterator,int,&map_value::second,const int&, const int*>
        data_const_iterator;
// Fill in my_map
for(key_iterator it = my_map.begin() ; it != my_map.end() ; ++it)
{
    cout << *it << endl; // Output the keys !
}
for(data_iterator it = my_map.begin() ; it != my_map.end() ; ++it)
{
    *it += 5;
    cout << *it << endl; // Output the data !
}
for(data_const_iterator it = my_map.begin() ; it != my_map.end() ; ++it)
{
    // It is not possible to modify the value pointed
    cout << *it << endl; // Output the data !
}
```

Definition at line 51 of file member_iterator.h.

17.51.2 Member Typedef Documentation

17.51.2.1 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
typedef Iterator util::SelectMemberIterator< Iterator, T, member,
Reference, Pointer >::base_iterator`

Type of the underlying iterator.

Definition at line 56 of file member_iterator.h.

17.51.2.2 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer
= T*> typedef std::iterator_traits<Iterator>::difference_type
util::SelectMemberIterator< Iterator, T, member, Reference, Pointer
>::difference_type`

Type of the difference between two iterators.

Definition at line 68 of file member_iterator.h.

17.51.2.3 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer =
T*> typedef std::iterator_traits<Iterator>::iterator_category
util::SelectMemberIterator< Iterator, T, member, Reference, Pointer
>::iterator_category`

Category of the iterator.

Definition at line 60 of file member_iterator.h.

17.51.2.4 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
typedef Pointer util::SelectMemberIterator< Iterator, T, member,
Reference, Pointer >::pointer`

Type of a pointer on the values.

Definition at line 76 of file member_iterator.h.

17.51.2.5 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
typedef Reference util::SelectMemberIterator< Iterator, T, member,
Reference, Pointer >::reference`

Type of a reference on the values.

Definition at line 72 of file member_iterator.h.

17.51.2.6 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer =
T*> typedef T util::SelectMemberIterator< Iterator, T, member,
Reference, Pointer >::value_type`

Type of the value iterated on.

Definition at line 64 of file member_iterator.h.

17.51.3 Constructor & Destructor Documentation

17.51.3.1 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
util::SelectMemberIterator< Iterator, T, member, Reference, Pointer
>::SelectMemberIterator () [inline]`

Default constructor.

Definition at line 81 of file member_iterator.h.

```
81 {}
```

17.51.3.2 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
util::SelectMemberIterator< Iterator, T, member, Reference, Pointer
>::SelectMemberIterator (const base_iterator & i) [inline]`

Conversion from the base iterator.

Definition at line 86 of file member_iterator.h.

```
87     : it(i)  
88     { }
```

17.51.3.3 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
util::SelectMemberIterator< Iterator, T, member, Reference, Pointer
>::SelectMemberIterator (const SelectMemberIterator< Iterator, T,
member, Reference, Pointer > & other) [inline]`

Copy constructor.

Definition at line 93 of file member_iterator.h.

```
94     : it(other.it)  
95     {}
```

17.51.4 Member Function Documentation

17.51.4.1 `template<class Iterator, class T, T std::iterator_traits< Iterator >::value_type::* member, class Reference = T&, class Pointer = T*> base_iterator util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::base () const [inline]`

Direct access to the base iterator.

Definition at line 211 of file member_iterator.h.

References `util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::it`.

Referenced by `graph::VVGraph< VertexContent, EdgeContent, compact >::clear()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::clear()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::erase()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::erase()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::eraseAllEdges()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::eraseEdge()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge()`.

```
211 { return it; }
```

17.51.4.2 `template<class Iterator, class T, T std::iterator_traits< Iterator >::value_type::* member, class Reference = T&, class Pointer = T*> const reference util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::operator* () const [inline]`

Constant dereference operator.

Definition at line 136 of file member_iterator.h.

```
136 { return (*it).*member; }
```

17.51.4.3 `template<class Iterator, class T, T std::iterator_traits< Iterator >::value_type::* member, class Reference = T&, class Pointer = T*> reference util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::operator* () [inline]`

Dereference operator.

Definition at line 132 of file member_iterator.h.

```
132 { return (*it).*member; }
```

17.51.4.4 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
SelectMemberIterator& util::SelectMemberIterator< Iterator, T,
member, Reference, Pointer >::operator++ (int) [inline]`

Postfix increment operator.

Definition at line 127 of file member_iterator.h.

References util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::it.

```
127 { SelectMemberIterator tmp(*this); ++it; return tmp; }
```

17.51.4.5 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
SelectMemberIterator& util::SelectMemberIterator< Iterator, T,
member, Reference, Pointer >::operator++ () [inline]`

Prefix increment operator.

Definition at line 123 of file member_iterator.h.

References util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::it.

```
123 { ++it; return *this; }
```

17.51.4.6 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer =
T*> SelectMemberIterator& util::SelectMemberIterator< Iterator,
T, member, Reference, Pointer >::operator+= (difference_type n)
[inline]`

In-place random increment operator.

Definition at line 165 of file member_iterator.h.

References util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::it.

```
166 { it += n; return *this; }
```

17.51.4.7 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
SelectMemberIterator& util::SelectMemberIterator< Iterator, T,
member, Reference, Pointer >::operator-- (int) [inline]`

Postfix decrement operator.

Definition at line 157 of file member_iterator.h.

References util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::it.

```
157 { SelectMemberIterator tmp(*this); --it; return tmp; }
```

17.51.4.8 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
SelectMemberIterator& util::SelectMemberIterator< Iterator, T,
member, Reference, Pointer >::operator-- () [inline]`

Prefix decrement operator.

Definition at line 153 of file member_iterator.h.

References util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::it.

```
153 { --it; return *this; }
```

17.51.4.9 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer =
T*> SelectMemberIterator& util::SelectMemberIterator< Iterator,
T, member, Reference, Pointer >::operator-= (difference_type n)
[inline]`

In-place random decrement operator.

Definition at line 171 of file member_iterator.h.

References util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::it.

```
172 { it -= n; return *this; }
```

17.51.4.10 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer =
T*> const pointer util::SelectMemberIterator< Iterator, T, member,
Reference, Pointer >::operator-> () const [inline]`

Pointer-like constant arrow operator.

Definition at line 145 of file member_iterator.h.

```
145 { return &((*it).member); }
```

17.51.4.11 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer =
T*> pointer util::SelectMemberIterator< Iterator, T, member,
Reference, Pointer >::operator-> () [inline]`

Pointer-like arrow operator.

Definition at line 141 of file member_iterator.h.

```
141 { return &((*it).*member); }
```

17.51.4.12 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer
= T*> SelectMemberIterator& util::SelectMemberIterator<
Iterator, T, member, Reference, Pointer >::operator= (const
SelectMemberIterator< Iterator, T, member, Reference, Pointer >
& copy) [inline]`

Assignment operator.

Definition at line 113 of file member_iterator.h.

References util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::it.

```
114 { it = copy.it; return *this; }
```

17.51.5 Friends And Related Function Documentation

17.51.5.1 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
difference_type operator- (const SelectMemberIterator< Iterator, T,
member, Reference, Pointer > & last, const SelectMemberIterator<
Iterator, T, member, Reference, Pointer > & first) [friend]`

Distance between two iterators.

Definition at line 204 of file member_iterator.h.

```
205 { return last.it - first.it; }
```

17.51.6 Member Data Documentation

17.51.6.1 `template<class Iterator, class T, T std::iterator_traits< Iterator
>::value_type::* member, class Reference = T&, class Pointer = T*>
base_iterator util::SelectMemberIterator< Iterator, T, member,
Reference, Pointer >::it [protected]`

Underlying iterator.

Definition at line 217 of file member_iterator.h.

Referenced by util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::base(), util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::operator++(), util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::operator+=(), util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::operator--(), util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::operator=(), and util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::operator=().

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

- vvelib/util/[member_iterator.h](#)

17.52 util::set_vector< T, Equal, Alloc > Struct Template Reference

Define a class using (small) vectors as unordered sets.

```
#include <set_vector.h>
```

Inherits std::vector< T, Alloc >.

Public Types

- typedef base::allocator_type **allocator_type**
- typedef std::vector< T, Alloc > **base**
- typedef base::const_iterator **const_iterator**
- typedef base::const_pointer **const_pointer**
- typedef base::const_reference **const_reference**
- typedef base::const_reverse_iterator **const_reverse_iterator**
- typedef base::difference_type **difference_type**
- typedef base::iterator **iterator**
- typedef Equal **key_equal**
- typedef base::value_type **key_type**
- typedef base::pointer **pointer**
- typedef base::reference **reference**
- typedef base::reverse_iterator **reverse_iterator**
- typedef base::size_type **size_type**
- typedef base::value_type **value_type**

Public Member Functions

- int **count** (const T &v) const
- iterator **erase** (iterator it)
- int **erase** (const T &v)
- const_iterator **find** (const T &v) const
- iterator **find** (const T &v)
- std::pair< iterator, bool > **insert** (const T &v, bool test_presence=true)
- [set_vector](#) & **operator=** (const [set_vector](#) &other)
- **set_vector** (const base ©)
- **set_vector** (const [set_vector](#) ©)

17.52.1 Detailed Description

```
template<typename T, typename Equal = std::equal_to<T>, typename Alloc =
std::allocator<T>> struct util::set_vector< T, Equal, Alloc >
```

Define a class using (small) vectors as unordered sets.

Definition at line 36 of file set_vector.h.

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

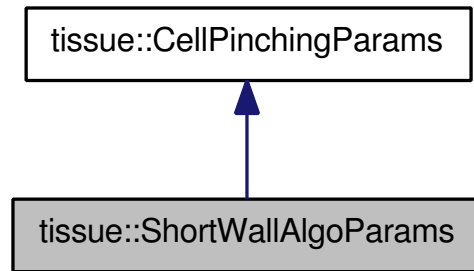
- vvelib/util/set_vector.h

17.53 tissue::ShortWallAlgoParams Struct Reference

Parameters for the shortest wall algorithm.

```
#include <tissue.h>
```

Inheritance diagram for tissue::ShortWallAlgoParams:



Public Member Functions

- [ShortWallAlgoParams](#) (const [ShortWallAlgoParams](#) ©)
Copy constructor.
- [ShortWallAlgoParams](#) (double cwm=0, bool scwm=true, double cws=0.1, double d=0.1)
Default and initialization constructor.

Public Attributes

- double [cellWallMin](#)
Minimum size of a wall.
- double [cellWallSample](#)
Size of sampling to find the minimum wall size.
- double [dx](#)
Precision required to consider a point in a segment.
- bool [strictCellWallMin](#)
If true, the minimum size of a wall is strictly enforce, otherwise, it will not imply a change of division wall.

17.53.1 Detailed Description

Parameters for the shortest wall algorithm.

Definition at line 180 of file tissue.h.

17.53.2 Constructor & Destructor Documentation

17.53.2.1 `tissue::ShortWallAlgoParams::ShortWallAlgoParams (double cwm = 0, bool scwm = true, double cws = 0.1, double d = 0.1) [inline]`

Default and initialization constructor.

Definition at line 185 of file tissue.h.

```
186      : CellPinchingParams ()
187      , cellWallMin (cwm)
188      , strictCellWallMin (scwm)
189      , cellWallSample (cws)
190      , dx (d)
191      { }
```

17.53.2.2 `tissue::ShortWallAlgoParams::ShortWallAlgoParams (const ShortWallAlgoParams & copy) [inline]`

Copy constructor.

Definition at line 196 of file tissue.h.

```
197      : CellPinchingParams (copy)
198      , cellWallMin (copy.cellWallMin)
199      , strictCellWallMin (copy.strictCellWallMin)
200      , cellWallSample (copy.cellWallSample)
201      , dx (copy.dx)
202      { }
```

17.53.3 Member Data Documentation

17.53.3.1 `double tissue::ShortWallAlgoParams::cellWallMin`

Minimum size of a wall.

Definition at line 207 of file tissue.h.

Referenced by `tissue::findDivisionPoints()`.

17.53.3.2 `double tissue::ShortWallAlgoParams::cellWallSample`

Size of sampling to find the minimum wall size.

Definition at line 216 of file tissue.h.

Referenced by tissue::findDivisionPoints().

17.53.3.3 double tissue::ShortWallAlgoParams::dx

Precision required to consider a point in a segment.

Definition at line 220 of file tissue.h.

Referenced by tissue::findDivisionPoints().

17.53.3.4 bool tissue::ShortWallAlgoParams::strictCellWallMin

If true, the minimum size of a wall is strictly enforce, otherwise, it will not imply a change of division wall.

Definition at line 212 of file tissue.h.

Referenced by tissue::findDivisionPoints().

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

- vvelib/algorithms/[tissue.h](#)

17.54 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content, compact >::single_neighborhood_t< VertexSrcContent, VertexTgtContent, EdgeSrcContent > Struct Template Reference

Type of the neighborhood of a vertex.

```
#include <vvbigraph.h>
```

Inherits graph::__vvbigraph_VertexCache< VVBIGRAPH_COMPACT_VERTEX, VVBIGraph, single_neighborhood_t< VertexSrcContent, VertexTgtContent, EdgeSrcContent >, graph::Vertex< VertexSrcContent > >.

Public Types

- typedef neighbor_t< VertexSrcContent, VertexTgtContent, EdgeSrcContent > **_neighbor_t**
- typedef __vvbigraph_VertexCache< VVBIGRAPH_COMPACT_VERTEX, VVBIGraph, single_neighborhood_t, graph::Vertex< VertexSrcContent > > **base**
- typedef graph::Vertex< VertexSrcContent > **vertex_src_t**
- typedef graph::Vertex< VertexTgtContent > **vertex_tgt_t**

Public Member Functions

- bool **operator==** (const single_neighborhood_t &other) const
Equality of two neighborhood.
- single_neighborhood_t (const single_neighborhood_t ©)

Public Attributes

- _neighbor_t::edge_list_t **edges**
List of the outgoing edges.
- const vertex_tgt_t * **flagged**
Flagged neighbor.
- __vvbigraph_set< compact, vertex_tgt_t >::type **in_edges**
Set of the sources of the incoming edges.

17.54.1 Detailed Description

```
template<typename Vertex1Content, typename Vertex2Content, type-
name Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_
= Edge1Content, bool compact = false>template<typename VertexSrc-
Content, typename VertexTgtContent, typename EdgeSrcContent> struct
graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content,
Edge2Content_, compact >::single_neighborhood_t< VertexSrcContent,
VertexTgtContent, EdgeSrcContent >
```

Type of the neighborhood of a vertex.

Definition at line 529 of file vvbigraph.h.

17.54.2 Member Function Documentation

17.54.2.1 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
template<typename VertexSrcContent , typename VertexTgtContent
, typename EdgeSrcContent > bool graph::VVBIGraph<
Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_,
compact >::single_neighborhood_t< VertexSrcContent,
VertexTgtContent, EdgeSrcContent >::operator==(const
single_neighborhood_t< VertexSrcContent, VertexTgtContent,
EdgeSrcContent > & other) const [inline]`

Equality of two neighborhood.

Only the outgoing edges are compared!

Definition at line 585 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content,
Edge2Content_, compact >::single_neighborhood_t< VertexSrcContent, VertexTgt-
Content, EdgeSrcContent >::edges`.

```
586                                     {
587                                     return edges == other.edges;
588                                     }
```

17.54.3 Member Data Documentation

17.54.3.1 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> template<typename VertexSrcContent , typename VertexTgtContent , typename EdgeSrcContent > _neighbor_t::edge_list_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::single_neighborhood_t< VertexSrcContent, VertexTgtContent, EdgeSrcContent >::edges`

List of the outgoing edges.

Definition at line 541 of file `vvbigraph.h`.

Referenced by `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::single_neighborhood_t< VertexSrcContent, VertexTgtContent, EdgeSrcContent >::operator==()`.

17.54.3.2 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> template<typename VertexSrcContent , typename VertexTgtContent , typename EdgeSrcContent > const vertex_tgt_t* graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::single_neighborhood_t< VertexSrcContent, VertexTgtContent, EdgeSrcContent >::flagged`

Flagged neighbor.

Definition at line 578 of file `vvbigraph.h`.

17.54.3.3 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> template<typename VertexSrcContent , typename VertexTgtContent , typename EdgeSrcContent > __vvbigraph_set<compact,vertex_tgt_t>::type graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::single_neighborhood_t< VertexSrcContent, VertexTgtContent, EdgeSrcContent >::in_edges`

Set of the sources of the incoming edges.

Definition at line 573 of file `vvbigraph.h`.

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

- `vvelib/graph/vvbigraph.h`

17.55 graph::VVGraph< VertexContent, EdgeContent, compact >::single_neighborhood_t Struct Reference

Type of the neighborhood of a vertex.

```
#include <vvgraph.h>
```

Inherits graph::__vvgraph_VertexCache< VVGRAPH_COMPACT_VERTEX, VVGraph, single_neighborhood_t >.

Public Types

- typedef __vvgraph_VertexCache< VVGRAPH_COMPACT_VERTEX, VVGraph, single_neighborhood_t > base

Public Member Functions

- bool operator==(const single_neighborhood_t &other) const
Equality of two neighborhood.
- single_neighborhood_t (const single_neighborhood_t ©)

Public Attributes

- edge_list_t edges
List of the outgoing edges.
- const vertex_t * flagged
Iterator toward the flagged neighbor.
- in_edges_t in_edges
Set of the sources of the incoming edges.

17.55.1 Detailed Description

```
template<typename VertexContent, typename EdgeContent = _-  
EmptyEdgeContent, bool compact = false> struct graph::VVGraph< Ver-  
texContent, EdgeContent, compact >::single_neighborhood_t
```

Type of the neighborhood of a vertex.

Definition at line 508 of file vvgraph.h.

17.55.2 Member Function Documentation

17.55.2.1 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> bool graph::VVGraph< VertexContent, EdgeContent, compact >::single_neighborhood_t::operator==(const single_neighborhood_t & other) const [inline]`

Equality of two neighborhood.

Only the outgoing edges are compared!

Definition at line 556 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::single_neighborhood_t::edges`.

```
557         {
558             return edges == other.edges;
559         }
```

17.55.3 Member Data Documentation

17.55.3.1 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> edge_list_t graph::VVGraph< VertexContent, EdgeContent, compact >::single_neighborhood_t::edges`

List of the outgoing edges.

Definition at line 539 of file vvgraph.h.

Referenced by `graph::VVGraph< VertexContent, EdgeContent, compact >::single_neighborhood_t::operator==()`.

17.55.3.2 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> const vertex_t* graph::VVGraph< VertexContent, EdgeContent, compact >::single_neighborhood_t::flagged`

Iterator toward the flagged neighbor.

Definition at line 549 of file vvgraph.h.

17.55.3.3 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> in_edges_t graph::VVGraph< VertexContent, EdgeContent, compact >::single_neighborhood_t::in_edges`

Set of the sources of the incoming edges.

Definition at line 544 of file vvgraph.h.

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

- vvelib/graph/[vvgraph.h](#)

17.56 solver::Solver< nb_vars, identifier > Class Template Reference

Implement a set of solvers for ODE on a graph.

```
#include <algorithms/solver.h>
```

Public Types

- typedef `_EdgeInternals< nb_vars >` [EdgeInternals](#)
Type of the internals stored on the edges for some algorithms.
- typedef `util::Matrix< nb_vars, nb_vars, double >` [Mat](#)
Type of a matrix used to store interaction on the edges.
- typedef `tag_holder< nb_vars, identifier >` [tag_t](#)
Tag identifying this specific solver type.
- typedef `util::Vector< nb_vars, double >` [Vec](#)
Type of a vector used to hold the data.
- typedef `_VertexInternals< nb_vars >` [VertexInternals](#)
Type of the internals stored on the vertices for some algorithms.

Public Member Functions

- template<typename VertexContent , typename EdgeContent , bool compact, typename Model >
void [FindPartials](#) (const [graph::Vertex](#)< VertexContent > &v,
[graph::VVGraph](#)< VertexContent, EdgeContent, compact > &S, double
[dt](#), int newtstep, [Model](#) &model)
Find the partial derivative by successive approximation on each neighbors.
- void [initialize](#) (bool update_dt=true)
Initialize the solver.
- template<typename VertexContent , typename EdgeContent , bool compact, typename Model >
void [operator\(\)](#) ([graph::VVGraph](#)< VertexContent, EdgeContent, compact >
&S, [Model](#) &model)
Invoke the solver, using the currently selected method.
- void [readParms](#) ([util::Parms](#) &parms, const [QString](#) §ion, bool selectAlgorithm=true)
Read the parameters needed for the different algorithms..

- template<typename VertexContent , typename EdgeContent , bool compact, typename Model >
void [solveAdaptiveCrankNicholson](#) (graph::VVGraph< VertexContent, EdgeContent, compact > &S, [Model](#) &model)

Invoke the solver, forcing the use of the adaptative Crank-Nicholson method.

- template<typename VertexContent , typename EdgeContent , bool compact, typename Model >
void [solveAdaptiveEuler](#) (graph::VVGraph< VertexContent, EdgeContent, compact > &S, [Model](#) &model)

Invoke the solver, forcing the use of the adaptative forward euler method.

- template<typename VertexContent , typename EdgeContent , bool compact, typename Model >
void [solveAdaptiveRungeKutta](#) (graph::VVGraph< VertexContent, EdgeContent, compact > &S, [Model](#) &model)

Invoke the solver, forcing the use of the adaptative order 4 runge-kutta method.

- template<typename VertexContent , typename EdgeContent , bool compact, typename Model >
void [solveEuler](#) (graph::VVGraph< VertexContent, EdgeContent, compact > &S, [Model](#) &model)

Invoke the solver, forcing the use of the forward euler method.

- template<typename VertexContent , typename EdgeContent , bool compact, typename Model >
void [solveFixedpoint](#) (graph::VVGraph< VertexContent, EdgeContent, compact > &S, [Model](#) &model)

Invoke the solver, forcing the use of the fixed point method.

- template<typename VertexContent , typename EdgeContent , bool compact, typename Model >
void [solveMidpoint](#) (graph::VVGraph< VertexContent, EdgeContent, compact > &S, [Model](#) &model)

Invoke the solver, forcing the use of the mid point method.

- [Solver](#) (util::Parms &parms, const [QString](#) §ion, bool selectAlgorithm=true)

Constructor reading the parameters instantly.

- [Solver](#) ()

Default constructor.

- template<typename VertexContent , typename EdgeContent , bool compact, typename Model >
void [solveRungeKutta](#) (graph::VVGraph< VertexContent, EdgeContent, compact > &S, [Model](#) &model)

Invoke the solver, forcing the use of the order 4 runge-kutta method.

Public Attributes

- bool [initialized](#)

True if the solver is initialized.

Parameters for Adaptative Euler

- double [AEulerHighTol](#)
high water mark
- double [AEulerIncDt](#)
Dt increment.
- double [AEulerLowTol](#)
low water mark
- double [AEulerResDt](#)
Restart Dt decrement.
- double [AEulerResTol](#)
restart tolerance
- [ToleranceType](#) [AEulerTolType](#)
Type of the tolerance.

Adaptive Runge-Kutta parms

- double [ARungeHighTol](#)
high water mark
- double [ARungeIncDt](#)
Dt increment/decrement.
- double [ARungeLowTol](#)
low water mark
- double [ARungeResDt](#)
restart Dt decrement
- double [ARungeResTol](#)
restart tolerance
- [ToleranceType](#) [ARungeTolType](#)
Tolerance type.

Conjugate gradient params (used by Crank Nicholson)

- double [ConjGradMaxSteps](#)
Max steps for conjugate gradient (multiple of N).
- double [ConjGradTol](#)
Tolerance for conjugate gradient.

- [ToleranceType ConjGradTolType](#)
Tolerance type.
- bool [ConstNbPartials](#)
Set if partials to neighbors are constant (diffusion only).

Crank Nicholson params

- double [CRAvgCPU](#)
Weight of current dt in eff average.
- double [CRIncDt](#)
Increment for timestep based on efficiency (multiple of current size).
- double [CRMinCPU](#)
Minimum CPU, below this always increases timestep.
- double [CResDt](#)
Decrement for timestep if unsucc.
- [SolvingMethod current_method](#)
Method currently used (i.e. it is set only after initialization).
- double [dt](#)
Current dt.
- [SolvingMethod method](#)
name Parameters
- int [PrintStats](#)
Level of debugging output.
- int [step](#)
Current computation step.

Misc general parms

- double [Dx](#)
Delta for numerical diff.
- bool [PrintMatrix](#)
Print Matrix (Conj-Grad).

Different Dt

- double [EulerDt](#)

Timestep for Forward Euler solver.

- double [FixedPointDt](#)
*Timestep for **FixedPoint** iteration solver.*
- double [InitialDt](#)
Timestep for adaptive solvers.
- double [MaxDt](#)
Maximum Dt for adapative solvers.
- double [MidPointDt](#)
Timestep for MultiPoint iteration solver.
- double [MinDt](#)
Minimum Dt for adapative solvers.
- double [RungeKuttaDt](#)
Timestep for Runge-Kutta solver.

Fixed Point iteration parms

- double [FixedPointMaxSteps](#)
Max steps for fixed point iteration.
- double [FixedPointTol](#)
Tolerance for fixed point iteration.
- [ToleranceType](#) [FixedPointTolType](#)
Type of tolerance.

Newton Tolerance (used by Crank Nicholson)

- int [NewtMaxSteps](#)
Max steps for Newton's method.
- double [NewtTol](#)
Tolerance for Newton's method.
- [ToleranceType](#) [NewtTolType](#)
Tolerance type.

Previous efficiency and dt for Cranck-Nicholson

- double [pdt](#)
Previous dt.
- double [peff](#)
Previous efficiency.

17.56.1 Detailed Description

template<size_t nb_vars, typename identifier = default_id_t> class solver::Solver< nb_vars, identifier >

Implement a set of solvers for ODE on a graph. The solver class implements 7 ODE algorithms:

1. Forward Euler
2. Adaptative forward Euler
3. Fixed point
4. Mid point
5. Runge-Kutta order 4
6. Adaptative Runge-Kutta order 4
7. Adaptative Crank-Nicholson

Definition at line 396 of file solver.h.

17.56.2 Member Typedef Documentation

17.56.2.1 template<size_t nb_vars, typename identifier = default_id_t> typedef _EdgeInternals<nb_vars> solver::Solver< nb_vars, identifier >::EdgeInternals

Type of the internals stored on the edges for some algorithms.

Definition at line 405 of file solver.h.

17.56.2.2 template<size_t nb_vars, typename identifier = default_id_t> typedef util::Matrix<nb_vars,nb_vars,double> solver::Solver< nb_vars, identifier >::Mat

Type of a matrix used to store interaction on the edges.

Definition at line 401 of file solver.h.

17.56.2.3 template<size_t nb_vars, typename identifier = default_id_t> typedef tag_holder<nb_vars,identifier> solver::Solver< nb_vars, identifier >::tag_t

Tag identifying this specific solver type.

Definition at line 408 of file solver.h.

17.56.2.4 `template<size_t nb_vars, typename identifier = default_id_t> typedef
util::Vector<nb_vars, double> solver::Solver< nb_vars, identifier
>::Vec`

Type of a vector used to hold the data.

Definition at line 399 of file solver.h.

17.56.2.5 `template<size_t nb_vars, typename identifier = default_id_t> typedef
_VertexInternals<nb_vars> solver::Solver< nb_vars, identifier
>::VertexInternals`

Type of the internals stored on the vertices for some algorithms.

Definition at line 403 of file solver.h.

17.56.3 Constructor & Destructor Documentation

17.56.3.1 `template<size_t nb_vars, typename identifier = default_id_t>
solver::Solver< nb_vars, identifier >::Solver () [inline]`

Default constructor.

Definition at line 549 of file solver.h.

```
550         : initialized(false)
551         {}
```

17.56.3.2 `template<size_t nb_vars, typename identifier = default_id_t>
solver::Solver< nb_vars, identifier >::Solver (util::Parms & parms,
const QString & section, bool selectAlgorithm = true) [inline]`

Constructor reading the parameters instantly.

Definition at line 554 of file solver.h.

References solver::Solver< nb_vars, identifier >::initialize(), and solver::Solver< nb_vars, identifier >::readParms().

```
555         : initialized(false)
556         {
557             readParms(parms, section, selectAlgorithm);
558             initialize();
559         }
```


17.56.4 Member Function Documentation

17.56.4.1 `template<size_t nb_vars, typename identifier = default_id_t>
template<typename VertexContent, typename EdgeContent, bool
compact, typename Model > void solver::Solver< nb_vars, identifier
>::FindPartials (const graph::Vertex< VertexContent > & v,
graph::VVGraph< VertexContent, EdgeContent, compact > & S,
double dt, int newtstep, Model & model) [inline]`

Find the partial derivative by successive approximation on each neighbors.

Definition at line 1115 of file solver.h.

References `solver::Solver< nb_vars, identifier >::ConstNbPartials`, `solver::Solver< nb_vars, identifier >::Dx`, `forall`, `solver::Solver< nb_vars, identifier >::initialized`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::neighbors()`.

Referenced by `solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson()`.

```

1117     {
1118         if(!initialized)
1119         {
1120             cerr << "Error, trying to use uninitialized solver" << endl;
1121             return;
1122         }
1123         typedef graph::Vertex<VertexContent> vertex;
1124         // Save old vars and deltas
1125         VertexInternals& interns = model.vertexInternals(v, tag_t());
1126         model.updateDerivatives(v, tag_t());
1127         Vec ovars = model.values(v, tag_t());
1128         Vec odels = model.derivatives(v, tag_t());
1129
1130         // Find partials for current node
1131         for(size_t i = 0; i < nb_vars; i++)
1132         {
1133             Vec& vars = model.values(v, tag_t());
1134             vars = ovars;
1135             vars[i] += Dx;
1136             model.updateDerivatives(v, tag_t());
1137             Vec dels = model.derivatives(v, tag_t());
1138             Vec partials = (dels - odels)/Dx * dt/2.0;
1139             partials[i] -= 1.0;
1140             interns.AT[i] = partials;
1141         }
1142         interns.AA = transpose(interns.AT);
1143         model.values(v, tag_t()) = ovars;
1144
1145         // If only a diffusive interaction with neighbors, do not re-calc partials
1146         (constant)
1147         if(newtstep == 0 || !ConstNbPartials)
1148         {
1149             // Find partials for neighbors
1150             forall(const vertex& n, S.neighbors(v))
1151             {
1152                 // Save velocity and position and deltas
1153                 Vec ovars = model.values(n, tag_t());
1154                 EdgeInternals &einterns = model.edgeInternals(v,n,S, tag_t());
1155
1156                 // Find partials for current node
1157                 for(size_t i = 0; i < nb_vars; i++)

```

```

1157         {
1158             Vec &vars = model.values(n, tag_t());
1159             vars = ovars;
1160             vars[i] += Dx;
1161             model.updateDerivatives(v, tag_t());
1162             Vec dels = model.derivatives(v, tag_t());
1163             Vec partials = (dels - odelts)/Dx * dt/2.0;
1164             einterns.AT[i] = partials;
1165         }
1166         model.edgeInternals(n,v,S,tag_t()).AA = transpose(einterns.AT);
1167         model.values(n, tag_t()) = ovars;
1168     }
1169 }
1170 model.derivatives(v, tag_t()) = odelts;
1171 }

```

17.56.4.2 `template<size_t nb_vars, typename identifier = default_id_t> void solver::Solver< nb_vars, identifier >::initialize (bool update_dt = true) [inline]`

Initialize the solver.

Call this method after changing the solver type.

Definition at line 637 of file solver.h.

References `solver::AdaptiveCrankNicholson`, `solver::AdaptiveEuler`, `solver::AdaptiveRungeKutta`, `solver::Solver< nb_vars, identifier >::current_method`, `solver::Solver< nb_vars, identifier >::dt`, `solver::Euler`, `solver::Solver< nb_vars, identifier >::EulerDt`, `solver::Fixedpoint`, `solver::Solver< nb_vars, identifier >::FixedPointDt`, `solver::Solver< nb_vars, identifier >::InitialDt`, `solver::Solver< nb_vars, identifier >::initialized`, `solver::Solver< nb_vars, identifier >::method`, `solver::Midpoint`, `solver::Solver< nb_vars, identifier >::MidPointDt`, `solver::Solver< nb_vars, identifier >::pdt`, `solver::Solver< nb_vars, identifier >::peff`, `solver::RungeKutta`, `solver::Solver< nb_vars, identifier >::RungeKuttaDt`, and `solver::Solver< nb_vars, identifier >::step`.

Referenced by `solver::Solver< nb_vars, identifier >::readParms()`, and `solver::Solver< nb_vars, identifier >::Solver()`.

```

638     {
639         if(update_dt or !initialized)
640         {
641             peff = 0;
642             pdt = 0;
643             step = 0;
644             dt = 0;
645         }
646         current_method = method;
647         switch(method)
648         {
649             case Euler:
650                 dt = EulerDt;
651                 break;
652             case AdaptiveEuler:
653                 if(update_dt)
654

```

```

655         dt = EulerDt;
656         break;
657     case Fixedpoint:
658         dt = FixedPointDt;
659         break;
660     case Midpoint:
661         dt = MidPointDt;
662         break;
663     case RungeKutta:
664         dt = RungeKuttaDt;
665         break;
666     case AdaptiveRungeKutta:
667     case AdaptiveCrankNicholson:
668         if(update_dt or !initialized)
669             dt = InitialDt;
670         break;
671     }
672     initialized = true;
673 }

```

17.56.4.3 `template<size_t nb_vars, typename identifier = default_id_t>
template<typename VertexContent, typename EdgeContent, bool
compact, typename Model > void solver::Solver< nb_vars, identifier
>::operator() (graph::VVGraph< VertexContent, EdgeContent,
compact > &S, Model &model) [inline]`

Invoke the solver, using the currently selected method.

Definition at line 1460 of file solver.h.

References solver::AdaptiveCrankNicholson, solver::AdaptiveEuler, solver::AdaptiveRungeKutta, solver::Solver< nb_vars, identifier >::current_method, solver::Euler, solver::Fixedpoint, solver::Midpoint, solver::RungeKutta, solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson(), solver::Solver< nb_vars, identifier >::solveAdaptiveEuler(), solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta(), solver::Solver< nb_vars, identifier >::solveEuler(), solver::Solver< nb_vars, identifier >::solveFixedpoint(), solver::Solver< nb_vars, identifier >::solveMidpoint(), and solver::Solver< nb_vars, identifier >::solveRungeKutta().

```

1461     {
1462         switch(current_method)
1463         {
1464             case Euler:
1465                 solveEuler(S, model);
1466                 break;
1467             case AdaptiveEuler:
1468                 solveAdaptiveEuler(S, model);
1469                 break;
1470             case Fixedpoint:
1471                 solveFixedpoint(S, model);
1472                 break;
1473             case Midpoint:
1474                 solveMidpoint(S, model);
1475                 break;
1476             case RungeKutta:

```

```

1477         solveRungeKutta(S, model);
1478         break;
1479     case AdaptiveRungeKutta:
1480         solveAdaptiveRungeKutta(S, model);
1481         break;
1482     case AdaptiveCrankNicholson:
1483         solveAdaptiveCrankNicholson(S, model);
1484         break;
1485     }
1486 }
```

17.56.4.4 `template<size_t nb_vars, typename identifier = default_id_t> void solver::Solver< nb_vars, identifier >::readParms (util::Parms & parms, const QString & section, bool selectAlgorithm = true) [inline]`

Read the parameters needed for the different algorithms,.

Parameters

parms Parameter object pointing to the parameter file

section Section including the parameters for the solver

selectAlgorithm If true, the algorithm will be read from the parameter file.

Definition at line 568 of file solver.h.

References solver::Solver< nb_vars, identifier >::AEulerHighTol, solver::Solver< nb_vars, identifier >::AEulerIncDt, solver::Solver< nb_vars, identifier >::AEulerLowTol, solver::Solver< nb_vars, identifier >::AEulerResDt, solver::Solver< nb_vars, identifier >::AEulerResTol, solver::Solver< nb_vars, identifier >::AEulerTolType, solver::Solver< nb_vars, identifier >::ARungeHighTol, solver::Solver< nb_vars, identifier >::ARungeIncDt, solver::Solver< nb_vars, identifier >::ARungeLowTol, solver::Solver< nb_vars, identifier >::ARungeResDt, solver::Solver< nb_vars, identifier >::ARungeResTol, solver::Solver< nb_vars, identifier >::ARungeTolType, solver::Solver< nb_vars, identifier >::ConjGradMaxSteps, solver::Solver< nb_vars, identifier >::ConjGradTol, solver::Solver< nb_vars, identifier >::ConjGradTolType, solver::Solver< nb_vars, identifier >::ConstNbPartials, solver::Solver< nb_vars, identifier >::CRAvgCPU, solver::Solver< nb_vars, identifier >::CRIncDt, solver::Solver< nb_vars, identifier >::CRMinCPU, solver::Solver< nb_vars, identifier >::CRResDt, solver::Solver< nb_vars, identifier >::Dx, solver::Solver< nb_vars, identifier >::EulerDt, solver::Solver< nb_vars, identifier >::FixedPointDt, solver::Solver< nb_vars, identifier >::FixedPointMaxSteps, solver::Solver< nb_vars, identifier >::FixedPointTol, solver::Solver< nb_vars, identifier >::FixedPointTolType, solver::Solver< nb_vars, identifier >::InitialDt, solver::Solver< nb_vars, identifier >::initialize(), solver::Solver< nb_vars, identifier >::MaxDt, solver::Solver< nb_vars, identifier >::method, solver::Solver< nb_vars, identifier >::MidPointDt, solver::Solver< nb_vars, identifier >::NewtMaxSteps, solver::Solver< nb_vars, identifier >::NewtTol, solver::Solver< nb_vars, identifier >::NewtTolType, solver::Solver< nb_vars, identifier >::PrintMatrix, solver::Solver< nb_vars, identifier >::PrintStats, and solver::Solver< nb_vars, identifier >::RungeKuttaDt.

Referenced by solver::Solver< nb_vars, identifier >::Solver().

```

569     {
570         if(selectAlgorithm)
571         {
572             parms(section, "Solver", method);
573             cout << "Solver selected: " << method << endl;
574         }
575
576         // Level of debugging output
577         parms(section, "PrintStats", PrintStats);
578
579         // Timesteps for the different algorithms
580         parms(section, "EulerDt", EulerDt);
581         parms(section, "FixedPointDt", FixedPointDt);
582         parms(section, "MidPointDt", MidPointDt);
583         parms(section, "RungeKuttaDt", RungeKuttaDt);
584         parms(section, "InitialDt", InitialDt);
585         parms(section, "MaxDt", MaxDt);
586
587         // Parameters for Adaptative Euler
588         parms(section, "AEulerTolType", AEulerTolType);
589         parms(section, "AEulerIncDt", AEulerIncDt);
590         parms(section, "AEulerResDt", AEulerResDt);
591         parms(section, "AEulerResTol", AEulerResTol);
592         parms(section, "AEulerLowTol", AEulerLowTol);
593         parms(section, "AEulerHighTol", AEulerHighTol);
594
595         // Fixed Point iteration parms
596         parms(section, "FixedPointMaxSteps", FixedPointMaxSteps);
597         parms(section, "FixedPointTol", FixedPointTol);
598         parms(section, "FixedPointTolType", FixedPointTolType);
599
600         // Adaptive Runge-Kutta parms
601         parms(section, "ARungeIncDt", ARungeIncDt);
602         parms(section, "ARungeResDt", ARungeResDt);
603         parms(section, "ARungeTolType", ARungeTolType);
604         parms(section, "ARungeResTol", ARungeResTol);
605         parms(section, "ARungeLowTol", ARungeLowTol);
606         parms(section, "ARungeHighTol", ARungeHighTol);
607
608         // Crank Nicholson params
609         parms(section, "CRIncDt", CRIncDt);
610         parms(section, "CRResDt", CRResDt);
611         parms(section, "CRAvgCPU", CRAvgCPU);
612         parms(section, "CRMinCPU", CRMinCPU);
613
614         // Newton Tolerance (used by Crank Nicholson)
615         parms(section, "NewtTol", NewtTol);
616         parms(section, "NewtTolType", NewtTolType);
617         parms(section, "NewtMaxSteps", NewtMaxSteps);
618
619         // Conjugate gradient params (used by Crank Nicholson)
620         parms(section, "ConjGradTol", ConjGradTol);
621         parms(section, "ConjGradTolType", ConjGradTolType);
622         parms(section, "ConjGradMaxSteps", ConjGradMaxSteps);
623         parms(section, "ConstNbPartials", ConstNbPartials);
624
625         // Misc general parms
626         parms(section, "Dx", Dx);
627         parms(section, "PrintMatrix", PrintMatrix);
628

```

```

629         initialize(false);
630     }

```

17.56.4.5 `template<size_t nb_vars, typename identifier = default_id_t>
template<typename VertexContent, typename EdgeContent,
bool compact, typename Model> void solver::Solver< nb_vars,
identifier >::solveAdaptiveCrankNicholson (graph::VVGraph<
VertexContent, EdgeContent, compact> & S, Model & model)
[inline]`

Invoke the solver, forcing the use of the adaptative Crank-Nicholson method.

Definition at line 1179 of file solver.h.

References `solver::Solver< nb_vars, identifier >::ConjGradMaxSteps`,
`solver::Solver< nb_vars, identifier >::ConjGradTol`, `solver::Solver< nb_vars, identifier >::ConjGradTolType`,
`solver::Solver< nb_vars, identifier >::ConstNbPartials`, `solver::Solver< nb_vars, identifier >::CRAvgCPU`,
`solver::Solver< nb_vars, identifier >::CRIncDt`, `solver::Solver< nb_vars, identifier >::CRMinCPU`,
`solver::Solver< nb_vars, identifier >::CRResDt`, `solver::Solver< nb_vars, identifier >::dt`,
`solver::Solver< nb_vars, identifier >::FindPartials()`, `forall`, `solver::Solver< nb_vars, identifier >::initialized`,
`solver::MAX_COMPONENT`, `solver::Solver< nb_vars, identifier >::MaxDt`, `solver::MEAN_COMPONENT`,
`graph::VVGraph< VertexContent, EdgeContent, compact>::neighbors()`, `solver::Solver< nb_vars, identifier >::NewtMaxSteps`,
`solver::Solver< nb_vars, identifier >::NewtTol`, `solver::Solver< nb_vars, identifier >::NewtTolType`,
`solver::Solver< nb_vars, identifier >::pdt`, `solver::Solver< nb_vars, identifier >::peff`,
`solver::Solver< nb_vars, identifier >::PrintMatrix`, `solver::Solver< nb_vars, identifier >::PrintStats`,
`graph::VVGraph< VertexContent, EdgeContent, compact>::size()`, `solver::Solver< nb_vars, identifier >::step`,
and `util::Vector< dim, T>::x()`.

Referenced by `solver::Solver< nb_vars, identifier >::operator()()`.

```

1180     {
1181         if(!initialized)
1182         {
1183             cerr << "Error, trying to use uninitialized solver" << endl;
1184             return;
1185         }
1186         typedef graph::Vertex<VertexContent> vertex;
1187         int conjGradMaxSteps = int (ConjGradMaxSteps*S.size());
1188         int newthalf = (int) (NewtMaxSteps/2.0);
1189         int cghalf = (int) (conjGradMaxSteps/2.0);
1190
1191         int cgsteps = 0;
1192         int newtsteps = 0;
1193         long cpu = 0;
1194
1195         double rr = 0;
1196         newtsteps = 0;
1197
1198         // Back up data values in case of restart
1199         forall(const vertex& v, S)
1200         {

```

```

1201         model.vertexInternals(v, tag_t()).x0 = model.values(v, tag_t());
1202     }
1203
1204     // Newton iteration
1205     bool restart = false;
1206     while(true)
1207     {
1208         if(restart)
1209         {
1210             cgsteps = newtsteps = 0;
1211             cpu = 1;
1212             dt *= CRResDt;
1213             restart = false;
1214             // Restore data
1215             forall(const vertex& v, S)
1216             {
1217                 model.values(v, tag_t()) = model.vertexInternals(v, tag_t()).x0;
1218             }
1219         }
1220         if(newtsteps == 0)
1221         {
1222             // Find t0 values
1223             forall(const vertex& v, S)
1224             {
1225                 model.updateDerivatives(v, tag_t());
1226                 VertexInternals &interns = model.vertexInternals(v, tag_t());
1227                 interns.t0 = model.derivatives(v, tag_t()) * dt/2.0;
1228                 interns.t0 += interns.x0;
1229             }
1230         }
1231
1232         // Initialize matrix A and vector b in Ax=b
1233         forall(const vertex& v, S)
1234         {
1235             FindPartials(v, S, dt, newtsteps, model);
1236             VertexInternals &interns = model.vertexInternals(v, tag_t());
1237             interns.b = (model.values(v, tag_t())
1238                         - model.derivatives(v, tag_t())*dt/2.0
1239                         - interns.t0);
1240         }
1241
1242         newtsteps++;
1243
1244         // Calculate cpu, add penalty for too many steps to avoid restart
1245         long newtcpu = 1;
1246         int newtover = newtsteps - newthalf;
1247         if(newtover > 1)
1248         {
1249             newtcpu += newtover;
1250             if(PrintStats >= 3)
1251                 cout << "Newton step:" << newtsteps << " penalized by:" << newtover <
1252         < endl;
1253         }
1254         if(!ConstNbPartials)
1255             newtcpu *= nb_vars;
1256         cpu += newtcpu;
1257
1258         // Print matrix if required
1259         if(PrintMatrix)
1260         {
1261             cout << "Bi-conjugate Gradient Matrix" << endl;
1262             //PrintMatrix();
1263         }
1264     }

```

```

1262     }
1263
1264     // Find first residual: Ax - b (note x = 0)
1265     rr = 0;
1266     double orr = 0;
1267     forall(const vertex& v, S)
1268     {
1269         VertexInternals &interns = model.vertexInternals(v, tag_t());
1270         interns.x = 0.0;
1271         interns.r1 = interns.r2 = -interns.b;
1272         interns.p1 = -interns.r1;
1273         interns.p2 = -interns.r2;
1274         rr += interns.r2 * interns.r1;
1275     }
1276
1277     // Print out initial residual
1278     if(PrintStats >= 3)
1279         cout << "Bi-conjugate gradient initial residual:" << fabs(rr)/S.size()
1280         << " step size:" << dt << endl;
1281
1282     // Main conj-grad iteration loop
1283     cgsteps = 0;
1284     double cgerr = 0;
1285     while(true)
1286     {
1287         cgsteps++;
1288
1289         // Calculate cpu, add penalty for too many steps to avoid restart
1290         int cgcpu = 1;
1291         int cgover = cgsteps - cghalf;
1292         if(cgover > 1)
1293         {
1294             cgcpu += cgover;
1295             if(PrintStats >= 3)
1296                 cout << "Bi-conjugate gradient step:" << cgsteps << " penalized by:
1297                 " << cgover << endl;
1298         }
1299         cpu += cgcpu;
1300
1301         // Find AApl, ATp2, and p2AApl
1302         double p2AApl = 0;
1303         forall(const vertex& v, S)
1304         {
1305             VertexInternals &v_interns = model.vertexInternals(v, tag_t());
1306             v_interns.AApl = v_interns.AA * v_interns.p1;
1307             v_interns.ATp2 = v_interns.AT * v_interns.p2;
1308             forall(const vertex& n, S.neighbors(v))
1309             {
1310                 VertexInternals &n_interns = model.vertexInternals(n, tag_t());
1311                 EdgeInternals &e_interns = model.edgeInternals(v, n, S, tag_t());
1312                 v_interns.AApl += e_interns.AA * n_interns.p1;
1313                 v_interns.ATp2 += e_interns.AT * n_interns.p2;
1314             }
1315             p2AApl += v_interns.p2 * v_interns.AApl;
1316         }
1317
1318         // Find coefficients for basis vectors (alpha)
1319         double alpha = rr / p2AApl;
1320
1321         // Find next solution, residual, error, and new rr
1322         orr = rr;
1323         rr = 0;

```



```

1322         cgerr = 0;
1323         forall(const vertex& v, S)
1324         {
1325             VertexInternals &interns = model.vertexInternals(v, tag_t());
1326             // Improve solution vector
1327             interns.x += alpha * interns.p1;
1328
1329             // Next residual
1330             interns.r1 += alpha * interns.AAp1;
1331             interns.r2 += alpha * interns.ATp2;
1332             double res = interns.r2 * interns.r1;
1333             rr += res;
1334             res = fabs(res);
1335             switch(ConjGradTolType)
1336             {
1337                 case MEAN_COMPONENT:
1338                     cgerr += res;
1339                     break;
1340                 case MAX_COMPONENT:
1341                 default:
1342                     if(res > cgerr)
1343                         cgerr = res;
1344             }
1345         }
1346         if(ConjGradTolType == MEAN_COMPONENT)
1347             cgerr /= S.size();
1348
1349         // Calculate next vector
1350         double beta = rr/orr;
1351         forall(const vertex& v, S)
1352         {
1353             VertexInternals &interns = model.vertexInternals(v, tag_t());
1354             interns.p1 = beta * interns.p1 - interns.r1;
1355             interns.p2 = beta * interns.p2 - interns.r2;
1356         }
1357
1358         // Print Stats
1359         if(PrintStats >= 3)
1360             cout << "Bi-conj grad step:" << cgsteps << " Residual:" << cgerr << endl;
1361
1362         // Check if we are done
1363         if(cgerr < ConjGradTol)
1364             break;
1365
1366         // Check if we need to restart
1367         if(cgsteps >= conjGradMaxSteps || isnan(cgerr) || !finite(cgerr))
1368         {
1369             if(PrintStats >= 1)
1370             {
1371                 if(cgsteps >= conjGradMaxSteps)
1372                     cout << "Restarting step:" << step << ", Conj-Grad too many iterations at stepsize:" << dt << endl;
1373                 else
1374                     cout << "Restarting step:" << step << ", Conj-Grad residual out of bounds" << endl;
1375             }
1376             restart = true;
1377             break;
1378         }
1379     }
1380

```

```

1381         double newterr = 0.0;
1382         if(cgsteps > 0 && !restart)
1383         {
1384             // Update model values
1385             forall(const vertex& v, S)
1386             {
1387                 VertexInternals &interns = model.vertexInternals(v, tag_t());
1388                 Vec &vars = model.values(v, tag_t());
1389                 vars += interns.x;
1390                 double corr = norm(interns.x);
1391                 switch(NewtTolType)
1392                 {
1393                     case MEAN_COMPONENT:
1394                         newterr += corr;
1395                         break;
1396                     case MAX_COMPONENT:
1397                     default:
1398                         if(corr > newterr)
1399                             newterr = corr;
1400                 }
1401             }
1402             if(NewtTolType == MEAN_COMPONENT)
1403                 newterr /= S.size();
1404
1405             if(PrintStats >= 2)
1406             {
1407                 cout << "Newton step:" << newtsteps << " Corr Size:" << newterr << "
1408 ";
1409                 cout << "Bi-Conj steps:" << cgsteps << " Residual:" << cgerr << endl;
1410             }
1411             if(newterr < NewtTol && !restart)
1412                 break;
1413         }
1414         if(newtsteps >= NewtMaxSteps || isnan(newterr) || !finite(newterr))
1415         {
1416             if(PrintStats >= 1)
1417             {
1418                 if(newtsteps >= NewtMaxSteps)
1419                     cout << "Restarting:" << step << ", Newton iterations exceeded at s
1420 tepsize:" << dt << endl;
1421                 else
1422                     cout << "Restarting:" << step << ", Newton correction out of bounds
1423 " << endl;
1424             }
1425             restart = true;
1426         }
1427         // Adaptive stepsize based on cpu use
1428         double eff = dt/double(cpu);
1429         double inc = dt * CRIncDt;
1430         if(cpu > CRMinCPU)
1431         {
1432             if(peff > eff)
1433             {
1434                 if(dt >= pdt)
1435                     inc *= -1;
1436             }
1437             else
1438             {

```

```

1439         if(dt < pdt)
1440             inc *= -1;
1441     }
1442 }
1443
1444     peff = (peff * (1.0 - CRAvgCPU) + CRAvgCPU * eff);
1445     pdt = dt;
1446
1447     if(PrintStats >= 2)
1448         cout << "Stepsize:" << dt << ", current increment:" << inc << endl;
1449
1450     dt += inc;
1451
1452     if(dt > MaxDt)
1453         dt = MaxDt;
1454 }
```

17.56.4.6 `template<size_t nb_vars, typename identifier = default_id_t>`
`template<typename VertexContent, typename EdgeContent,`
`bool compact, typename Model > void solver::Solver< nb_vars,`
`identifier >::solveAdaptiveEuler (graph::VVGraph< VertexContent,`
`EdgeContent, compact > & S, Model & model) [inline]`

Invoke the solver, forcing the use of the adaptative forward euler method.

Definition at line 706 of file solver.h.

References solver::Solver< nb_vars, identifier >::AEulerHighTol, solver::Solver< nb_vars, identifier >::AEulerIncDt, solver::Solver< nb_vars, identifier >::AEulerLowTol, solver::Solver< nb_vars, identifier >::AEulerResDt, solver::Solver< nb_vars, identifier >::AEulerResTol, solver::Solver< nb_vars, identifier >::AEulerTolType, solver::Solver< nb_vars, identifier >::dt, forall, solver::Solver< nb_vars, identifier >::initialized, solver::MAX_COMPONENT, solver::Solver< nb_vars, identifier >::MaxDt, solver::MEAN_COMPONENT, solver::Solver< nb_vars, identifier >::MinDt, solver::Solver< nb_vars, identifier >::PrintStats, graph::VVGraph< VertexContent, EdgeContent, compact >::size(), and solver::Solver< nb_vars, identifier >::step.

Referenced by solver::Solver< nb_vars, identifier >::operator()().

```

707     {
708         if(!initialized)
709         {
710             cerr << "Error, trying to use uninitialized solver" << endl;
711             return;
712         }
713         typedef graph::Vertex<VertexContent> vertex;
714         double err = 0;
715         forall(const vertex& v, S)
716         {
717             model.updateDerivatives(v, tag_t());
718         }
719         forall(const vertex& v, S)
720         {
721             VertexInternals& interns = model.vertexInternals(v, tag_t());
722             Vec& vars = model.values(v, tag_t());
```

```

723     const Vec& derivs = model.derivatives(v, tag_t());
724     double derr = norm(derivs - interns.pDerivs);
725     switch(AEulerTolType)
726     {
727         case MEAN_COMPONENT:
728             err += derr;
729             break;
730         case MAX_COMPONENT:
731         default:
732             if(err < derr)
733                 err = derr;
734     }
735     vars += dt * derivs;
736     interns.ppDerivs = interns.pDerivs;
737     interns.pDerivs = derivs;
738 }
739 if(AEulerTolType == MEAN_COMPONENT)
740     err /= S.size();
741
742 if(step)
743 {
744     if(err > AEulerResTol && dt > MinDt)
745     { // Backup a step and restart
746         forall( vertex v, S )
747         {
748             VertexInternals& interns = model.vertexInternals(v, tag_t());
749             Vec& vars = model.values(v, tag_t());
750             const Vec& derivs = model.derivatives(v, tag_t());
751             vars -= dt * (1.0 - AEulerResDt) * derivs;
752             interns.pDerivs = interns.ppDerivs;
753         }
754         dt *= AEulerResDt;
755         if(PrintStats >= 1)
756             cout << "Euler restart Err:" << err << " Timestep:" << dt << endl;
757     }
758     else if(err > AEulerHighTol) // Over high water, reduce step
759         dt -= dt * AEulerIncDt;
760     else if(err < AEulerLowTol) // Under high water, increase step
761         dt += dt * AEulerIncDt;
762
763     // Clip to max/min
764     dt = min(MaxDt, max(MinDt, dt));
765 }
766 step++;
767 }

```

17.56.4.7 `template<size_t nb_vars, typename identifier = default_id_t>`
`template<typename VertexContent, typename EdgeContent, bool`
`compact, typename Model > void solver::Solver< nb_vars, identifier`
`>::solveAdaptiveRungeKutta (graph::VVGraph< VertexContent,`
`EdgeContent, compact > & S, Model & model) [inline]`

Invoke the solver, forcing the use of the adaptative order 4 runge-kutta method.

Definition at line 964 of file solver.h.

References `solver::Solver< nb_vars, identifier >::ARungeHighTol`, `solver::Solver< nb_vars, identifier >::ARungeIncDt`, `solver::Solver< nb_vars, identifier >::ARungeLowTol`, `solver::Solver< nb_vars, identifier >::ARungeResDt`,

solver::Solver< nb_vars, identifier >::ARungeResTol, solver::Solver< nb_vars, identifier >::ARungeTolType, solver::Solver< nb_vars, identifier >::dt, forall, solver::Solver< nb_vars, identifier >::initialized, solver::MAX_COMPONENT, solver::Solver< nb_vars, identifier >::MaxDt, solver::MEAN_COMPONENT, solver::Solver< nb_vars, identifier >::MinDt, solver::Solver< nb_vars, identifier >::PrintStats, graph::VVGraph< VertexContent, EdgeContent, compact >::size(), and solver::Solver< nb_vars, identifier >::step.

Referenced by solver::Solver< nb_vars, identifier >::operator().

```

965     {
966         if(!initialized)
967         {
968             cerr << "Error, trying to use uninitialized solver" << endl;
969             return;
970         }
971         typedef graph::Vertex<VertexContent> vertex;
972         double err = 0.0;
973         for(int rstep = 1; rstep <= 3; rstep++)
974         {
975             double ldt = dt;
976             if(rstep != 1) // 1st step at full dt
977                 ldt *= .5;
978
979             // Save original data values and deltas at start of step - k1
980             if(rstep != 2) // only compute k1 on 1st and 3rd steps
981             {
982                 forall(const vertex& v, S)
983                 {
984                     VertexInternals& interns = model.vertexInternals(v, tag_t());
985                     if(rstep == 1) // 1st time through save original vars
986                         interns.pVars = model.values(v, tag_t());
987                     model.updateDerivatives(v, tag_t());
988                     interns.k1 = model.derivatives(v, tag_t());
989                 }
990             }
991
992             // Find deltas at first trial midpoint - k2
993             forall(const vertex& v, S)
994             {
995                 VertexInternals& interns = model.vertexInternals(v, tag_t());
996                 Vec &vars = model.values(v, tag_t());
997                 if(rstep != 3) // Steps 1 & 2 start from beginning of timestep
998                     vars = interns.pVars;
999                 else
1000                     vars = interns.mVars;
1001                 vars += ldt * .5 * interns.k1;
1002             }
1003             forall(const vertex& v, S)
1004             {
1005                 model.updateDerivatives(v, tag_t());
1006                 VertexInternals& interns = model.vertexInternals(v, tag_t());
1007                 interns.k2 = model.derivatives(v, tag_t());
1008             }
1009
1010             // Find deltas at second trial midpoint - k3
1011             forall(const vertex& v, S)
1012             {
1013                 VertexInternals& interns = model.vertexInternals(v, tag_t());
1014                 Vec &vars = model.values(v, tag_t());
1015                 if(rstep != 3) // Steps 1 & 2 start from beginning of timestep

```

```

1016         vars = interns.pVars;
1017     else
1018         vars = interns.mVars;
1019     vars += ldt * .5 * interns.k2;
1020 }
1021 forall(const vertex& v, S)
1022 {
1023     model.updateDerivatives(v, tag_t());
1024     VertexInternals& interns = model.vertexInternals(v, tag_t());
1025     interns.k3 = model.derivatives(v, tag_t());
1026 }
1027
1028 // Find deltas at trial endpoint (based on second midpoint estimate) - k3

1029 forall(const vertex& v, S)
1030 {
1031     VertexInternals& interns = model.vertexInternals(v, tag_t());
1032     Vec &vars = model.values(v, tag_t());
1033     if(rstep != 3) // Steps 1 & 2 start from beginning of timestep
1034         vars = interns.pVars;
1035     else
1036         vars = interns.mVars;
1037     vars += ldt * interns.k3;
1038 }
1039 forall(const vertex& v, S)
1040 {
1041     model.updateDerivatives(v, tag_t());
1042     VertexInternals& interns = model.vertexInternals(v, tag_t());
1043     interns.k4 = model.derivatives(v, tag_t());
1044 }
1045
1046 // Update concentrations based on Runge-Kutta weighted average
1047 forall(const vertex& v, S)
1048 {
1049     VertexInternals& interns = model.vertexInternals(v, tag_t());
1050     Vec &vars = model.values(v, tag_t());
1051     if(rstep != 3) // Steps 1 & 2 update from beginning of timestep
1052         vars = interns.pVars;
1053     else
1054         vars = interns.mVars;
1055
1056     vars += ldt * ((interns.k1 + interns.k4)/6.0 + (interns.k2 + interns.k3
1057 )/3.0);
1058     if(rstep == 1)
1059     {
1060         interns.fVars = vars;
1061     }
1062     else if(rstep == 2)
1063     {
1064         interns.mVars = vars;
1065     }
1066     else // Calc truncation err
1067     {
1068         double derr = norm(vars - interns.fVars)/norm(interns.fVars);
1069         switch(ARungeTolType)
1070         {
1071             case MEAN_COMPONENT:
1072                 err += derr;
1073                 break;
1074             case MAX_COMPONENT:
1075                 default:
1076                     if(derr > err)

```

```

1076         err = derr;
1077     }
1078 }
1079 }
1080 }
1081 if(ARungeTolType == MEAN_COMPONENT)
1082     err /= S.size();
1083
1084 // Check truncation error (diif of midpoints)
1085 if(step != 0)
1086 {
1087     if(err > ARungeResTol && dt > MinDt) // Backup a step and restart
1088     {
1089         forall(const vertex& v, S)
1090         {
1091             VertexInternals &interns = model.vertexInternals(v, tag_t());
1092             Vec &vars = model.values(v, tag_t());
1093             vars = interns.pVars;
1094         }
1095         dt *= ARungeResDt;
1096         if(PrintStats >= 1)
1097             cout << "Runge-Kutta restart Err:" << err << " Timestep:" << dt << endl;
1098     }
1099     else if(err > ARungeHighTol) // Over high water, reduce step
1100         dt -= dt * ARungeIncDt;
1101     else if(err < ARungeLowTol) // Under high water, increase step
1102         dt += dt * ARungeIncDt;
1103
1104     // Clip to max/min
1105     dt = min(MaxDt, max(MinDt, dt));
1106 }
1107 step++;
1108 }

```

17.56.4.8 `template<size_t nb_vars, typename identifier = default_id_t>`
`template<typename VertexContent, typename EdgeContent, bool`
`compact, typename Model > void solver::Solver< nb_vars, identifier`
`>::solveEuler (graph::VVGraph< VertexContent, EdgeContent,`
`compact > &S, Model &model) [inline]`

Invoke the solver, forcing the use of the forward euler method.

Definition at line 679 of file solver.h.

References solver::Solver< nb_vars, identifier >::dt, forall, solver::Solver< nb_vars, identifier >::initialized, and solver::Solver< nb_vars, identifier >::step.

Referenced by solver::Solver< nb_vars, identifier >::operator()().

```

680     {
681         if(!initialized)
682         {
683             cerr << "Error, trying to use uninitialized solver" << endl;
684             return;
685         }
686         typedef graph::Vertex<VertexContent> vertex;
687         typedef graph::VVGraph<VertexContent, EdgeContent, compact> vvgraph;

```

```

688     forall(const vertex& v, S)
689     {
690         model.updateDerivatives(v, tag_t());
691     }
692     forall(const vertex& v, S)
693     {
694         Vec& vars = model.values(v, tag_t());
695         const Vec& derivs = model.derivatives(v, tag_t());
696         vars += dt * derivs;
697     }
698     step++;
699 }

```

17.56.4.9 `template<size_t nb_vars, typename identifier = default_id_t>
template<typename VertexContent, typename EdgeContent,
bool compact, typename Model> void solver::Solver<nb_vars,
identifier>::solveFixedpoint(graph::VVGraph<VertexContent,
EdgeContent, compact> &S, Model &model) [inline]`

Invoke the solver, forcing the use of the fixed point method.

Definition at line 773 of file solver.h.

References `solver::Solver<nb_vars, identifier>::dt`, `solver::Solver<nb_vars, identifier>::FixedPointMaxSteps`, `solver::Solver<nb_vars, identifier>::FixedPointTolType`, `forall`, `solver::Solver<nb_vars, identifier>::initialized`, `solver::MAX_COMPONENT`, `solver::MEAN_COMPONENT`, `solver::Solver<nb_vars, identifier>::PrintStats`, `graph::VVGraph<VertexContent, EdgeContent, compact>::size()`, and `solver::Solver<nb_vars, identifier>::step`.

Referenced by `solver::Solver<nb_vars, identifier>::operator()()`.

```

774     {
775         if(!initialized)
776         {
777             cerr << "Error, trying to use uninitialized solver" << endl;
778             return;
779         }
780         typedef graph::Vertex<VertexContent> vertex;
781         // Find initial deltas
782         forall(const vertex& v, S)
783         {
784             model.updateDerivatives(v, tag_t());
785         }
786         forall(const vertex& v, S)
787         {
788             VertexInternals& interns = model.vertexInternals(v, tag_t());
789             // save original vars
790             interns.pDerivs = model.derivatives(v, tag_t());
791             Vec& vars = model.values(v, tag_t());
792             interns.pVars = vars;
793             // Find first estimate
794             vars += dt * interns.pDerivs;
795         }
796
797         // Loop to improve approximation
798         double fxerr = 0;

```



```

799     int fxsteps = 0;
800     while(true)
801     {
802         fxsteps++;
803         fxerr = 0;
804         forall(const vertex& v, S)
805         {
806             model.updateDerivatives(v, tag_t());
807         }
808
809         forall(const vertex& v, S)
810         {
811             VertexInternals& interns = model.vertexInternals(v, tag_t());
812             Vec derivs = model.derivatives(v, tag_t());
813             Vec& vars = model.values(v, tag_t());
814             vars = interns.pVars + dt * (derivs + interns.pDerivs)/2.0;
815             double err = norm(derivs - interns.pDerivs);
816             switch(FixedPointTolType)
817             {
818                 case MEAN_COMPONENT:
819                     fxerr += err;
820                     break;
821                 case MAX_COMPONENT:
822                 default:
823                     if(err > fxerr)
824                         fxerr = err;
825             }
826             interns.pDerivs = derivs;
827         }
828         if(FixedPointTolType == MEAN_COMPONENT)
829             fxerr /= S.size();
830
831         if(fxerr < FixedPointTol || fxsteps >= FixedPointMaxSteps)
832             break;
833     }
834     if(PrintStats >= 2)
835         cout << "Fixed point steps:" << fxsteps << ", Err:" << fxerr << endl;
836     step++;
837 }

```

17.56.4.10 `template<size_t nb_vars, typename identifier = default_id_t>
template<typename VertexContent, typename EdgeContent,
bool compact, typename Model> void solver::Solver< nb_vars,
identifier >::solveMidpoint (graph::VVGrahp< VertexContent,
EdgeContent, compact > & S, Model & model) [inline]`

Invoke the solver, forcing the use of the mid point method.

Definition at line 843 of file solver.h.

References `solver::Solver< nb_vars, identifier >::dt`, `forall`, `solver::Solver< nb_vars, identifier >::initialized`, and `solver::Solver< nb_vars, identifier >::step`.

Referenced by `solver::Solver< nb_vars, identifier >::operator()()`.

```

844     {
845         if(!initialized)
846         {

```

```

847         cerr << "Error, trying to use uninitialized solver" << endl;
848         return;
849     }
850     typedef graph::Vertex<VertexContent> vertex;
851     // Save original data values and deltas at start of step - k1
852     forall( vertex v, S)
853     {
854         VertexInternals& interns = model.vertexInternals(v, tag_t());
855         interns.pVars = model.values(v, tag_t());
856         model.updateDerivatives(v, tag_t());
857         interns.k1 = model.derivatives(v, tag_t());
858     }
859
860     // Find deltas at trial midpoint - k2
861     forall(const vertex& v, S)
862     {
863         VertexInternals& interns = model.vertexInternals(v, tag_t());
864         Vec &vars = model.values(v, tag_t());
865         vars += dt * 0.5 * interns.k1;
866     }
867
868     forall(const vertex& v, S)
869     {
870         VertexInternals& interns = model.vertexInternals(v, tag_t());
871         model.updateDerivatives(v, tag_t());
872         interns.k2 = model.derivatives(v, tag_t());
873     }
874
875     // Update concentrations based on midpoint deltas
876     forall(const vertex& v, S)
877     {
878         VertexInternals& interns = model.vertexInternals(v, tag_t());
879         Vec &vars = model.values(v, tag_t());
880         vars = interns.pVars + dt * interns.k2;
881     }
882     step++;
883 }

```

17.56.4.11 `template<size_t nb_vars, typename identifier = default_id_t>
template<typename VertexContent, typename EdgeContent,
bool compact, typename Model > void solver::Solver< nb_vars,
identifier >::solveRungeKutta (graph::VVGraph< VertexContent,
EdgeContent, compact > &S, Model &model) [inline]`

Invoke the solver, forcing the use of the order 4 runge-kutta method.

Definition at line 889 of file solver.h.

References `solver::Solver< nb_vars, identifier >::dt`, `forall`, `solver::Solver< nb_vars, identifier >::initialized`, and `solver::Solver< nb_vars, identifier >::step`.

Referenced by `solver::Solver< nb_vars, identifier >::operator()()`.

```

890     {
891         if(!initialized)
892         {
893             cerr << "Error, trying to use uninitialized solver" << endl;
894             return;

```

```

895     }
896     typedef graph::Vertex<VertexContent> vertex;
897     // Save original data values and deltas at start of step - k1
898     forall(const vertex& v, S)
899     {
900         VertexInternals& interns = model.vertexInternals(v, tag_t());
901         interns.pVars = model.values(v, tag_t());
902         model.updateDerivatives(v, tag_t());
903         interns.k1 = model.derivatives(v, tag_t());
904     }
905
906     // Find deltas at first trial midpoint - k2
907     forall(const vertex& v, S)
908     {
909         VertexInternals& interns = model.vertexInternals(v, tag_t());
910         Vec &vars = model.values(v, tag_t());
911         vars = interns.pVars + dt * .5 * interns.k1;
912     }
913     forall(const vertex& v, S)
914     {
915         VertexInternals& interns = model.vertexInternals(v, tag_t());
916         model.updateDerivatives(v, tag_t());
917         interns.k2 = model.derivatives(v, tag_t());
918     }
919
920     // Find deltas at second trial midpoint - k3
921     forall(const vertex& v, S)
922     {
923         VertexInternals& interns = model.vertexInternals(v, tag_t());
924         Vec &vars = model.values(v, tag_t());
925         vars = interns.pVars + dt * .5 * interns.k2;
926     }
927     forall(const vertex& v, S)
928     {
929         VertexInternals& interns = model.vertexInternals(v, tag_t());
930         model.updateDerivatives(v, tag_t());
931         interns.k3 = model.derivatives(v, tag_t());
932     }
933
934     // Find deltas at trial endpoint (based on second midpoint estimate) - k3
935     forall(const vertex& v, S)
936     {
937         VertexInternals& interns = model.vertexInternals(v, tag_t());
938         Vec &vars = model.values(v, tag_t());
939         vars = interns.pVars + dt * interns.k3;
940     }
941     forall(const vertex& v, S)
942     {
943         VertexInternals& interns = model.vertexInternals(v, tag_t());
944         model.updateDerivatives(v, tag_t());
945         interns.k4 = model.derivatives(v, tag_t());
946     }
947
948     // Update concentrations based on Runge-Kutta weighted average
949     forall(const vertex& v, S)
950     {
951         VertexInternals& interns = model.vertexInternals(v, tag_t());
952         Vec &vars = model.values(v, tag_t());
953         vars = interns.pVars;
954         vars += dt * ((interns.k1+interns.k4)/6.0 + (interns.k2 + interns.k3)/3.0
955     );
956     }

```

```

956         step++;
957     }

```

17.56.5 Member Data Documentation

17.56.5.1 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::AEulerHighTol`

high water mark

Definition at line 458 of file solver.h.

Referenced by `solver::Solver< nb_vars, identifier >::readParms()`, and `solver::Solver< nb_vars, identifier >::solveAdaptiveEuler()`.

17.56.5.2 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::AEulerIncDt`

Dt increment.

Definition at line 450 of file solver.h.

Referenced by `solver::Solver< nb_vars, identifier >::readParms()`, and `solver::Solver< nb_vars, identifier >::solveAdaptiveEuler()`.

17.56.5.3 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::AEulerLowTol`

low water mark

Definition at line 456 of file solver.h.

Referenced by `solver::Solver< nb_vars, identifier >::readParms()`, and `solver::Solver< nb_vars, identifier >::solveAdaptiveEuler()`.

17.56.5.4 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::AEulerResDt`

Restart Dt decrement.

Definition at line 452 of file solver.h.

Referenced by `solver::Solver< nb_vars, identifier >::readParms()`, and `solver::Solver< nb_vars, identifier >::solveAdaptiveEuler()`.

17.56.5.5 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::AEulerResTol`

restart tolerance

Definition at line 454 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveEuler().

17.56.5.6 template<size_t nb_vars, typename identifier = default_id_t> ToleranceType solver::Solver< nb_vars, identifier >::AEulerTolType

Type of the tolerance.

Definition at line 448 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveEuler().

17.56.5.7 template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::ARungeHighTol

high water mark

Definition at line 486 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta().

17.56.5.8 template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::ARungeIncDt

Dt increment/decrement.

Definition at line 476 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta().

17.56.5.9 template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::ARungeLowTol

low water mark

Definition at line 484 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta().

17.56.5.10 template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::ARungeResDt

restart Dt decrement

Definition at line 478 of file solver.h.

Referenced by `solver::Solver< nb_vars, identifier >::readParms()`, and `solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta()`.

17.56.5.11 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::ARungeResTol`

restart tolerance

Definition at line 482 of file `solver.h`.

Referenced by `solver::Solver< nb_vars, identifier >::readParms()`, and `solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta()`.

17.56.5.12 `template<size_t nb_vars, typename identifier = default_id_t> ToleranceType solver::Solver< nb_vars, identifier >::ARungeTolType`

Tolerance type.

Definition at line 480 of file `solver.h`.

Referenced by `solver::Solver< nb_vars, identifier >::readParms()`, and `solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta()`.

17.56.5.13 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::ConjGradMaxSteps`

Max steps for conjugate gradient (multiple of N).

Definition at line 521 of file `solver.h`.

Referenced by `solver::Solver< nb_vars, identifier >::readParms()`, and `solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson()`.

17.56.5.14 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::ConjGradTol`

Tolerance for conjugate gradient.

Definition at line 517 of file `solver.h`.

Referenced by `solver::Solver< nb_vars, identifier >::readParms()`, and `solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson()`.

17.56.5.15 `template<size_t nb_vars, typename identifier = default_id_t> ToleranceType solver::Solver< nb_vars, identifier >::ConjGradTolType`

Tolerance type.

Definition at line 519 of file `solver.h`.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

17.56.5.16 **template<size_t nb_vars, typename identifier = default_id_t> bool solver::Solver< nb_vars, identifier >::ConstNbPartials**

Set if partials to neighbors are constant (diffusion only).

Definition at line 523 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::FindPartials(), solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

17.56.5.17 **template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::CRAvgCPU**

Weight of current dt in eff average.

Definition at line 497 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

17.56.5.18 **template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::CRIncDt**

Increment for timestep based on efficiency (multiple of current size).

Definition at line 493 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

17.56.5.19 **template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::CRMinCPU**

Minimun CPU, below this alway increases timestep.

Definition at line 499 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

17.56.5.20 **template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::CRResDt**

Decrement for timestep if unsucc.

Definition at line 495 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

17.56.5.21 `template<size_t nb_vars, typename identifier = default_id_t>
SolvingMethod solver::Solver< nb_vars, identifier
>::current_method`

Method currently used (i.e. it is set only after initialization).

Definition at line 416 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::initialize(), and solver::Solver< nb_vars, identifier >::operator()().

17.56.5.22 `template<size_t nb_vars, typename identifier = default_id_t>
double solver::Solver< nb_vars, identifier >::dt`

Current dt.

Definition at line 418 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::initialize(), solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson(), solver::Solver< nb_vars, identifier >::solveAdaptiveEuler(), solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta(), solver::Solver< nb_vars, identifier >::solveEuler(), solver::Solver< nb_vars, identifier >::solveFixedpoint(), solver::Solver< nb_vars, identifier >::solveMidpoint(), and solver::Solver< nb_vars, identifier >::solveRungeKutta().

17.56.5.23 `template<size_t nb_vars, typename identifier = default_id_t>
double solver::Solver< nb_vars, identifier >::Dx`

Delta for numerical diff.

Definition at line 530 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::FindPartials(), and solver::Solver< nb_vars, identifier >::readParms().

17.56.5.24 `template<size_t nb_vars, typename identifier = default_id_t>
double solver::Solver< nb_vars, identifier >::EulerDt`

Timestep for Forward Euler solver.

Definition at line 429 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::initialize(), and solver::Solver< nb_vars, identifier >::readParms().

17.56.5.25 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::FixedPointDt`

Timestep for **FixedPoint** iteration solver.

Definition at line 431 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::initialize(), and solver::Solver< nb_vars, identifier >::readParms().

17.56.5.26 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::FixedPointMaxSteps`

Max steps for fixed point iteration.

Definition at line 465 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveFixedpoint().

17.56.5.27 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::FixedPointTol`

Tolerance for fixed point iteration.

Definition at line 467 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms().

17.56.5.28 `template<size_t nb_vars, typename identifier = default_id_t> ToleranceType solver::Solver< nb_vars, identifier >::FixedPointTolType`

Type of tolerance.

Definition at line 469 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveFixedpoint().

17.56.5.29 `template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::InitialDt`

Timestep for adaptive solvers.

Definition at line 437 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::initialize(), and solver::Solver< nb_vars, identifier >::readParms().

17.56.5.30 **template<size_t nb_vars, typename identifier = default_id_t> bool solver::Solver< nb_vars, identifier >::initialized**

True if the solver is initialized.

Definition at line 546 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::FindPartials(), solver::Solver< nb_vars, identifier >::initialize(), solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson(), solver::Solver< nb_vars, identifier >::solveAdaptiveEuler(), solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta(), solver::Solver< nb_vars, identifier >::solveEuler(), solver::Solver< nb_vars, identifier >::solveFixedpoint(), solver::Solver< nb_vars, identifier >::solveMidpoint(), and solver::Solver< nb_vars, identifier >::solveRungeKutta().

17.56.5.31 **template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::MaxDt**

Maximum Dt for adaptive solvers.

Definition at line 441 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParams(), solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson(), solver::Solver< nb_vars, identifier >::solveAdaptiveEuler(), and solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta().

17.56.5.32 **template<size_t nb_vars, typename identifier = default_id_t> SolvingMethod solver::Solver< nb_vars, identifier >::method**

name Parameters

Method read in the parameter file

Definition at line 414 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::initialize(), and solver::Solver< nb_vars, identifier >::readParams().

17.56.5.33 **template<size_t nb_vars, typename identifier = default_id_t> double solver::Solver< nb_vars, identifier >::MidPointDt**

Timestep for MultiPoint iteration solver.

Definition at line 433 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::initialize(), and solver::Solver< nb_vars, identifier >::readParams().

**17.56.5.34 template<size_t nb_vars, typename identifier = default_id_t>
 double solver::Solver< nb_vars, identifier >::MinDt**

Minimum Dt for adaptive solvers.

Definition at line 439 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::solveAdaptiveEuler(), and solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta().

**17.56.5.35 template<size_t nb_vars, typename identifier = default_id_t> int
 solver::Solver< nb_vars, identifier >::NewtMaxSteps**

Max steps for Newton's method.

Definition at line 510 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

**17.56.5.36 template<size_t nb_vars, typename identifier = default_id_t>
 double solver::Solver< nb_vars, identifier >::NewtTol**

Tolerance for Newton's method.

Definition at line 506 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

**17.56.5.37 template<size_t nb_vars, typename identifier = default_id_t>
 ToleranceType solver::Solver< nb_vars, identifier >::NewtTolType**

Tolerance type.

Definition at line 508 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

**17.56.5.38 template<size_t nb_vars, typename identifier = default_id_t>
 double solver::Solver< nb_vars, identifier >::pdt**

Previous dt.

Definition at line 541 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::initialize(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

**17.56.5.39 template<size_t nb_vars, typename identifier = default_id_t>
double solver::Solver< nb_vars, identifier >::peff**

Previous efficiency.

Definition at line 539 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::initialize(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

**17.56.5.40 template<size_t nb_vars, typename identifier = default_id_t> bool
solver::Solver< nb_vars, identifier >::PrintMatrix**

Print Matrix (Conj-Grad).

Definition at line 532 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), and solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson().

**17.56.5.41 template<size_t nb_vars, typename identifier = default_id_t> int
solver::Solver< nb_vars, identifier >::PrintStats**

Level of debugging output.

Definition at line 423 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::readParms(), solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson(), solver::Solver< nb_vars, identifier >::solveAdaptiveEuler(), solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta(), and solver::Solver< nb_vars, identifier >::solveFixedpoint().

**17.56.5.42 template<size_t nb_vars, typename identifier = default_id_t>
double solver::Solver< nb_vars, identifier >::RungeKuttaDt**

Timestep for Runge-Kutta solver.

Definition at line 435 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::initialize(), and solver::Solver< nb_vars, identifier >::readParms().

**17.56.5.43 template<size_t nb_vars, typename identifier = default_id_t> int
solver::Solver< nb_vars, identifier >::step**

Current computation step.

Definition at line 420 of file solver.h.

Referenced by solver::Solver< nb_vars, identifier >::initialize(), solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson(), solver::Solver< nb_

vars, identifier >::solveAdaptiveEuler(), solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta(), solver::Solver< nb_vars, identifier >::solveEuler(), solver::Solver< nb_vars, identifier >::solveFixedpoint(), solver::Solver< nb_vars, identifier >::solveMidpoint(), and solver::Solver< nb_vars, identifier >::solveRungeKutta().

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

- [vvelib/algorithms/solver.h](#)

17.57 `complex_factory::SquareFiller`< `Complex`, `Model` > Struct Template Reference

For internal use only!

```
#include <complex_grid.h>
```

Public Types

- typedef `Complex::cell_graph` `cell_graph`

Public Member Functions

- `IMPORT_COMPLEX_VERTICES` (`Complex`)
- void `initGridVertex` (`size_t` i, `size_t` j, const `cell` &v, `cell_graph` &)
- `SquareFiller` (const [Point3d](#) &bl, const [Point3d](#) &sr, const [Point3d](#) &su, `Complex` &T, [Model](#) &model)

Public Attributes

- [Point3d](#) `bottom_left`
- [Point3d](#) `delta_right`
- [Point3d](#) `delta_up`
- [Model](#) & `model`
- [Point3d](#) `shift_right`
- [Point3d](#) `shift_up`
- `Complex` & `T`

17.57.1 Detailed Description

```
template<typename    Complex,    class    Model>    struct    complex_
factory::SquareFiller< Complex, Model >
```

For internal use only! Compute the position of each point

Definition at line 38 of file `complex_grid.h`.

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

- `vvelib/factory/complex_grid.h`

17.58 util::Tensor< T > Class Template Reference

A growth tensor class.

```
#include <util/tensor.h>
```

Public Member Functions

- const T & [angle_theta](#) ()
Return the s axis scale.
- [util::Point](#)< T > [grow](#) ([util::Point](#)< T > p)
Transform a [Point](#).
- [Tensor](#)< T > & [operator=](#) ([Tensor](#)< T > &tensor)
- const T & [scale_s](#) ()
Return the s axis scale.
- const T & [scale_t](#) ()
Return the s axis scale.
- void [set](#) (const T &s, const T &t, const T &theta)
set new values.
- [Tensor](#) (const T &s=T(), const T &t=T(), const T &theta=T())
Constructor.
- [~Tensor](#) ()
Destructor.

17.58.1 Detailed Description

```
template<class T> class util::Tensor< T >
```

A growth tensor class. The [Tensor](#) class is a utility to handle transformations to a [Point](#) using a growth tensor. For some functions, T must be reducible to a floating point type to satisfy functions from the math library.

Definition at line 23 of file tensor.h.

17.58.2 Constructor & Destructor Documentation

17.58.2.1 `template<class T> util::Tensor< T >::Tensor (const T & s = T (),
const T & t = T (), const T & theta = T ()) [inline]`

Constructor.

Parameters

- s* scale of first axis
- t* scale of second axis
- theta* angle of rotation

Definition at line 30 of file tensor.h.

```

32                                     :
33     s(t), t(s), theta(theta), a(), b(), c(), d()
34     {
        calculateMatrix();

```

17.58.2.2 template<class T> util::Tensor< T>::~~Tensor() [inline]

Destructor.

Definition at line 37 of file tensor.h.

```

38 {}

```

17.58.3 Member Function Documentation**17.58.3.1 template<class T> const T& util::Tensor< T>::angle_theta() [inline]**

Return the s axis scale.

Definition at line 46 of file tensor.h.

```

47 {return theta;}

```

17.58.3.2 template<class T> util::Point<T> util::Tensor< T>::grow(util::Point< T> p) [inline]

Transform a [Point](#).

Parameters

- p* The point to transform. Only x and y of the point are considered.

Definition at line 66 of file tensor.h.

```

67                                     {
68     return util::Point<T>(a * p.x() + b * p.y(), c * p.x() + d * p.y());

```


**17.58.3.3 template<class T > const T& util::Tensor< T >::scale_s ()
 [inline]**

Return the s axis scale.

Definition at line 40 of file tensor.h.

```
41 {return s;}
```

**17.58.3.4 template<class T > const T& util::Tensor< T >::scale_t ()
 [inline]**

Return the s axis scale.

Definition at line 43 of file tensor.h.

```
44 {return t;}
```

**17.58.3.5 template<class T > void util::Tensor< T >::set (const T & s, const T
 & t, const T & theta) [inline]**

set new values.

Parameters

s the s scale

t the t scale

theta the angle

Definition at line 54 of file tensor.h.

```
55                                     {  
56     this->s = s;  
57     this->t = t;  
58     this->theta = theta;  
59     calculateMatrix();
```

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

- vvelib/util/[tensor.h](#)

17.59 util::Texture1D Class Reference

A utility class for one-dimensional textures.

```
#include <util/texture.h>
```

Public Member Functions

- void [bind](#) ()
Bind the texture.
- const [Texture1D](#) & [operator=](#) (const [Texture1D](#) &texture)
Assignment operator.
- [Texture1D](#) (std::string filename)
Constructor initialising from an image file.
- [Texture1D](#) (const [Texture1D](#) &texture)
Copy Constructor.
- [Texture1D](#) ()
Default constructor.
- [~Texture1D](#) ()
Destructor.

Static Public Member Functions

- static void [blend](#) ()
Sets the blending mode to blend.
- static void [clamp](#) (bool enable=true)
Enables texture coordinate clamping.
- static void [decal](#) ()
Sets the blending mode to decal.
- static void [filter](#) (bool enable=true)
Enables filtering.
- static void [modulate](#) ()
Sets the blending mode to modulate.
- static void [replace](#) ()
Sets the blending mode to replace.

17.59.1 Detailed Description

A utility class for one-dimensional textures.

Definition at line 18 of file texture.h.

17.59.2 Constructor & Destructor Documentation

17.59.2.1 util::Texture1D::Texture1D ()

Default constructor.

Definition at line 7 of file texture.cpp.

```
8             :
9   data(0),
10  size(0),
11  tex_name(0)
12  {}
```

17.59.2.2 util::Texture1D::Texture1D (const Texture1D & *texture*)

Copy Constructor.

Definition at line 14 of file texture.cpp.

References clamp(), filter(), and modulate().

```
15             :
16   data(0),
17   size(0),
18   tex_name(0)
19 {
20   if (!texture.data) return;
21
22   size = texture.size;
23   data = new GLubyte[size * 4];
24
25   for (unsigned int i = 0; i < size; i++)
26     data[i] = texture.data[i];
27
28   glEnable(GL_TEXTURE_1D);
29
30   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
31
32   glGenTextures(1, &tex_name);
33   glBindTexture(GL_TEXTURE_1D, tex_name);
34   glTexImage1D(GL_TEXTURE_1D, 0, GL_RGBA, size, 0, GL_RGBA, GL_UNSIGNED_BYTE, data);
35
36   clamp();
37   filter();
38   modulate();
39 }
```

17.59.2.3 util::Texture1D::Texture1D (std::string filename)

Constructor initialising from an image file.

Parameters

filename The file containing the image. The image must be in a format supported by the Qt image filters.

Definition at line 44 of file texture.cpp.

References clamp(), filter(), and modulate().

```

45                                     :
46     data(0),
47     size(0),
48     tex_name(0)
49 {
50     QImage image(filename.c_str());
51     if (!image.width()) {
52         std::cerr << "Error: Texture initialised by "
53                 << filename << " is empty." << std::endl;
54         return;
55     }
56
57     size = image.width();
58
59     data = new GLubyte[size * 4];
60     for (unsigned int x = 0; x < size; x++) {
61         data[x * 4 + 0] = (GLubyte)qRed(image.pixel(x, 0));
62         data[x * 4 + 1] = (GLubyte)qGreen(image.pixel(x, 0));
63         data[x * 4 + 2] = (GLubyte)qBlue(image.pixel(x, 0));
64         data[x * 4 + 3] = (GLubyte)qAlpha(image.pixel(x, 0));
65     }
66
67     glEnable(GL_TEXTURE_1D);
68
69     glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
70
71     glGenTextures(1, &tex_name);
72     glBindTexture(GL_TEXTURE_1D, tex_name);
73     glTexImage1D(GL_TEXTURE_1D, 0, GL_RGBA, size, 0, GL_RGBA, GL_UNSIGNED_BYTE, data);
74
75     clamp();
76     filter();
77     modulate();
78 }

```

17.59.2.4 util::Texture1D::~~Texture1D ()

Destructor.

Definition at line 110 of file texture.cpp.

```

110                                     {
111     glDeleteTextures(1, &tex_name);
112     delete[] data;
113 }

```

17.59.3 Member Function Documentation

17.59.3.1 void util::Texture1D::bind ()

Bind the texture.

Definition at line 116 of file texture.cpp.

```
116         {
117     glBindTexture(GL_TEXTURE_1D, tex_name);
118 }
```

17.59.3.2 void util::Texture1D::blend () [static]

Sets the blending mode to blend.

Definition at line 155 of file texture.cpp.

```
155         {
156     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_BLEND);
157 }
```

17.59.3.3 void util::Texture1D::clamp (bool *enable* = true) [static]

Enables texture coordinate clamping.

Parameters

enable If true enable clamping, if false repeat.

Definition at line 123 of file texture.cpp.

Referenced by operator=(), and Texture1D().

```
123         {
124     if (enable)
125         glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_CLAMP);
126     else
127         glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_REPEAT);
128 }
```

17.59.3.4 void util::Texture1D::decals () [static]

Sets the blending mode to decal.

Definition at line 150 of file texture.cpp.

```
150         {
151     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
152 }
```

17.59.3.5 void util::Texture1D::filter (bool *enable* = true) [static]

Enables filtering.

Parameters

enable If true, filtering is enabled, if false it is disabled/

Definition at line 133 of file texture.cpp.

Referenced by operator=(), and Texture1D().

```

133                                     {
134     if (enable) {
135         glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
136         glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
137     }
138     else {
139         glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
140         glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
141     }
142 }
```

17.59.3.6 void util::Texture1D::modulate () [static]

Sets the blending mode to modulate.

Definition at line 145 of file texture.cpp.

Referenced by operator=(), and Texture1D().

```

145                                     {
146     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
147 }
```

17.59.3.7 const util::Texture1D & util::Texture1D::operator= (const Texture1D & *texture*)

Assignment operator.

Definition at line 80 of file texture.cpp.

References clamp(), filter(), and modulate().

```

80                                     {
81     glDeleteTextures(1, &tex_name);
82     delete[] data;
83     data = 0;
84     size = 0;
85
86     if (!texture.data) return *this;
87
88     size = texture.size;
```

```
89  data = new GLubyte[size * 4];
90
91  for (unsigned int i = 0; i < size; i++)
92      data[i] = texture.data[i];
93
94  glEnable(GL_TEXTURE_1D);
95
96  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
97
98  glGenTextures(1, &tex_name);
99  glBindTexture(GL_TEXTURE_1D, tex_name);
100  glTexImage1D(GL_TEXTURE_1D, 0, GL_RGBA, size, 0, GL_RGBA, GL_UNSIGNED_BYTE, data);
101
102  clamp();
103  filter();
104  modulate();
105
106  return *this;
107 }
```

17.59.3.8 void util::Texture1D::replace () [static]

Sets the blending mode to replace.

Definition at line 160 of file texture.cpp.

```
160  {
161      glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
162  }
```

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

- vvelib/util/[texture.h](#)
- vvelib/util/texture.cpp

17.60 util::Texture2D Class Reference

A utility class for two-dimensional textures.

```
#include <util/texture.h>
```

Public Member Functions

- void [bind](#) ()
Bind the texture.
- const [Texture2D](#) & [operator=](#) (const [Texture2D](#) &texture)
Assignment operator.
- [Texture2D](#) (std::string filename)
Constructor initialising from an image file.
- [Texture2D](#) (const [Texture2D](#) &texture)
Copy constructor.
- [Texture2D](#) ()
Default constructor.
- [~Texture2D](#) ()
Destructor.

Static Public Member Functions

- static void [blend](#) ()
Sets the blending mode to blend.
- static void [clamp](#) (bool enable=true)
Enables texture coordinate clamping.
- static void [decal](#) ()
Sets the blending mode to decal.
- static void [filter](#) (bool enable=true)
Enables filtering.
- static void [modulate](#) ()
Sets the blending mode to modulate.
- static void [replace](#) ()
Sets the blending mode to replace.

17.60.1 Detailed Description

A utility class for two-dimensional textures.

Definition at line 51 of file texture.h.

17.60.2 Constructor & Destructor Documentation

17.60.2.1 util::Texture2D::Texture2D ()

Default constructor.

Definition at line 165 of file texture.cpp.

```
166                                     :
167   data(0),
168   size(0),
169   width(0),
170   height(0),
171   tex_name(0)
172 {}
```

17.60.2.2 util::Texture2D::Texture2D (const Texture2D & *texture*)

Copy constructor.

Definition at line 174 of file texture.cpp.

References clamp(), filter(), and modulate().

```
175                                     :
176   data(0),
177   size(0),
178   tex_name(0)
179 {
180   if (!texture.data) return;
181
182   size = texture.size;
183   width = texture.width;
184   height = texture.height;
185   data = new GLubyte[size * 4];
186
187   for (unsigned int i = 0; i < size * 4; i++)
188     data[i] = texture.data[i];
189
190   glEnable(GL_TEXTURE_2D);
191
192   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
193
194   glGenTextures(1, &tex_name);
195   glBindTexture(GL_TEXTURE_2D, tex_name);
196   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_
197     BYTE, data);
198
199   clamp();
200   filter();
```

```

200     modulate();
    }

```

17.60.2.3 util::Texture2D::Texture2D (std::string filename)

Constructor initialising from an image file.

Parameters

filename The file containing the image. The image must be in a format supported by the Qt image filters.

Definition at line 206 of file texture.cpp.

References clamp(), filter(), and modulate().

```

207                                     :
208     data(0),
209     size(0),
210     tex_name(0)
211 {
212     QImage image(filename.c_str());
213     if (!image.width()) {
214         std::cerr << "Error: Texture initialised by "
215                     << filename << " is empty." << std::endl;
216         return;
217     }
218
219     width = image.width();
220     height = image.height();
221     size = width * height;
222
223     data = new GLubyte[size * 4];
224     for (unsigned int x = 0; x < width; x++) {
225         for (unsigned int y = 0; y < height; y++) {
226             data[(x + y * width) * 4 + 0] = (GLubyte)qRed(image.pixel(x, y));
227             data[(x + y * width) * 4 + 1] = (GLubyte)qGreen(image.pixel(x, y));
228             data[(x + y * width) * 4 + 2] = (GLubyte)qBlue(image.pixel(x, y));
229             data[(x + y * width) * 4 + 3] = (GLubyte)qAlpha(image.pixel(x, y));
230         }
231     }
232
233     glEnable(GL_TEXTURE_2D);
234
235     glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
236
237     glGenTextures(1, &tex_name);
238     glBindTexture(GL_TEXTURE_2D, tex_name);
239     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_
        BYTE, data);
240
241     clamp();
242     filter();
243     modulate();
    }

```

17.60.2.4 util::Texture2D::~~Texture2D ()

Destructor.

Definition at line 278 of file texture.cpp.

```
278         {
279     glDeleteTextures(1, &tex_name);
280     delete[] data;
281 }
```

17.60.3 Member Function Documentation

17.60.3.1 void util::Texture2D::bind ()

Bind the texture.

Definition at line 284 of file texture.cpp.

```
284         {
285     glBindTexture(GL_TEXTURE_2D, tex_name);
286 }
```

17.60.3.2 void util::Texture2D::blend () [static]

Sets the blending mode to blend.

Definition at line 327 of file texture.cpp.

```
327         {
328     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_BLEND);
329 }
```

17.60.3.3 void util::Texture2D::clamp (bool *enable* = true) [static]

Enables texture coordinate clamping.

Parameters

enable If true enable clamping, if false repeat.

Definition at line 291 of file texture.cpp.

Referenced by operator=(), and Texture2D().

```
291         {
292     if (enable) {
293         glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
294         glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
295     }
```

```
296     else {
297         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
298         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
299     }
300 }
```

17.60.3.4 void util::Texture2D::decal () [static]

Sets the blending mode to decal.

Definition at line 322 of file texture.cpp.

```
322     {
323         glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
324     }
```

17.60.3.5 void util::Texture2D::filter (bool *enable* = true) [static]

Enables filtering.

Parameters

enable If true, filtering is enabled, if false it is disabled/

Definition at line 305 of file texture.cpp.

Referenced by operator=(), and Texture2D().

```
305     {
306         if (enable) {
307             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
308             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
309         }
310         else {
311             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
312             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
313         }
314     }
```

17.60.3.6 void util::Texture2D::modulate () [static]

Sets the blending mode to modulate.

Definition at line 317 of file texture.cpp.

Referenced by operator=(), and Texture2D().

```
317     {
318         glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
319     }
```

17.60.3.7 const util::Texture2D & util::Texture2D::operator= (const Texture2D & texture)

Assignment operator.

Definition at line 246 of file texture.cpp.

References clamp(), filter(), and modulate().

```
246                                     {
247     glDeleteTextures(1, &tex_name);
248     delete[] data;
249     data = 0;
250     size = 0;
251     width = 0;
252     height = 0;
253
254     if (!texture.data) return *this;
255
256     size = texture.size;
257     data = new GLubyte[size * 4];
258
259     for (unsigned int i = 0; i < size * 4; i++)
260         data[i] = texture.data[i];
261
262     glEnable(GL_TEXTURE_2D);
263
264     glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
265
266     glGenTextures(1, &tex_name);
267     glBindTexture(GL_TEXTURE_2D, tex_name);
268     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_
        BYTE, data);
269
270     clamp();
271     filter();
272     modulate();
273
274     return *this;
275 }
```

17.60.3.8 void util::Texture2D::replace () [static]

Sets the blending mode to replace.

Definition at line 332 of file texture.cpp.

```
332                                     {
333     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
334 }
```

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

- vvelib/util/[texture.h](#)
- vvelib/util/texture.cpp

17.61 parallel::ThreadEval Class Reference

Base class for function evaluated in a thread.

```
#include <algorithms/parallel.h>
```

Public Member Functions

- virtual void **operator()** () const =0

17.61.1 Detailed Description

Base class for function evaluated in a thread.

Definition at line 25 of file parallel.h.

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

- vvelib/algorithms/[parallel.h](#)

17.62 **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass** > **Class Template Reference** 521

~~17.62 **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass** > **Class Template Reference**~~

Class handling the development and representation of a cell tissue.

```
#include <algorithms/tissue.h>
```

Inheritance diagram for **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass** >:



Public Types

- typedef [VVComplex::division_data](#) **division_data**
Type of the division data used for the complex.
- typedef [VVComplex::division_result_t](#) **division_result_t**
- typedef [Model](#) **model_t**
- typedef [vvcomplex::VVComplex](#)< [MANDATORY_COMPLEX_TEMPLATE_ARGS](#), [leaf_class](#) > **VVComplex**

Public Member Functions

- [division_result_t](#) **divideCell** (const [cell](#) &to_divide, const [cell](#) &cell_kept)
Divide a cell.
- [division_result_t](#) **divideCell** (const [cell](#) &to_divide)
Divide a cell.
- template<typename CellContainer >
[division_result_t](#) **divideCell** (const [cell](#) &to_divide, const CellContainer &kept_cells)
Divide a cell.
- template<typename AlgoParameter >
[division_result_t](#) **divideCell** (const [cell](#) &to_divide, const AlgoParameter ¶ms, const [cell](#) &cell_kept)
Divide a cell.
- template<typename AlgoParameter , typename CellContainer >
[division_result_t](#) **divideCell** (const [cell](#) &c, const AlgoParameter ¶ms, const CellContainer &to_keep)

Divide a cell of the tissue.

- `division_result_t divideCell` (const `cell` &c, const `division_data` &ddata, const `cell` &to_keep)

Divide a cell with the cells to keep defined by any kind of container.

- `template<typename CellContainer > division_result_t divideCell` (const `cell` &c, const `division_data` &ddata, const `CellContainer` &to_keep)

Divide a cell with the cells to keep defined by any kind of container.

- `division_result_t divideCell` (const `cell` &c, const `division_data` &ddata)
- `void drawCell` (const `cell` &c, double value)

Draw a single cell.

- `void drawCell` (const `cell` &c, `util::Palette::Color` cell_color, `util::Palette::Color` center_color)

Draw a single cell.

- `void drawCellContour` (const `cell` &c)

Draw just the contour of the cells.

- `void drawSimplifiedCell` (const `cell` &c)

Draw the cell as a simple colored polygon.

- `void drawWalledCell` (const `cell` &c, double value)

Draw a cell with walls, specifying the value.

- `void drawWalledCell` (const `cell` &c, `util::Palette::Color` cell_color, `util::Palette::Color` center_color)

Draw a single cell with differentiated walls.

- `CellPinchingParams getCellPinchingParams` ()

Returns the cell pinching parameter object.

- `ClosestMidAlgoParams getClosestMidAlgoParams` ()

Returns the closest mid algorithm parameter object.

- `ClosestWallAlgoParams getCloseWallAlgoParams` ()

Returns the closest wall algorithm parameter object.

- `ShortWallAlgoParams getShortWallAlgoParams` ()

Returns the shortest wall algorithm parameter object.

- `IMPORT_COMPLEX_TYPES` (`VVComplex`)

- `void postDraw` ()

Restore the OpenGL context after the drawing.

- void `preDraw` ()
Setup the OpenGL context to draw the cells.
- void `readParms` (`util::Parms` &parms, const `QString` §ion)
Read the parameters for the algorithmic part of the object.
- void `readViewParms` (`util::Parms` &parms, const `QString` §ion)
Read the parameters for the drawing of a cell.
- typedef `RESOLVE_LEAF_CLASS` (`LeafClass`, `Tissue`) `leaf_class`
- void `setCellPinchingParams` (`CellPinchingParams` ¶ms)
Set the pinching parameters to be used.
- `Tissue` (const `Tissue` ©)
Copy constructor.
- `Tissue` (const `util::Palette` &pal, `Model` *mod)
Constructor setting up both the palette and the model.
- `Tissue` (`Model` *mod)
Constructor with just the model.
- `util::Palette::Color` `valueCenterColor` (double value)
Get the center color corresponding to a given value.
- `util::Palette::Color` `valueColor` (double value)
Get the color corresponding to a given value.

Public Attributes

- double `blending`
Blending for the exterior of the cell.
- `CELL_DIVISION_ALGORITHM` `cellDivAlg`
Default algorithm to use.
- double `cellMaxPinch`
Maximum cell pinching.
- double `cellPinch`
Cell pinching ratio.
- double `cellWallCorner`
Size of the cell corners.

- double [cellWallMin](#)
Minimum size of a cell wall.
- double [cellWallSample](#)
Number of samples to find the shortest wall.
- double [cellWallWidth](#)
Width of the "thin" cell wall.
- double [center_blending](#)
Blending for the center of the cell.
- int [colorBegin](#)
Color of the high values for the cell.
- double [colorCenter](#)
Increase in color of the center ($0 < \text{colorCenter} < 1$).
- int [colorEnd](#)
Color of the low values for the cell.
- int [contourColor](#)
Color of the contour of the cell.
- bool [drawBorders](#)
Draw thick corners if true, otherwise draw `cellWallWidth` pixel borders.
- bool [drawInsides](#)
The inside of the cell is drawn depending on the value is true, otherwise the end color is used.
- const [util::Palette](#) * [palette](#)
Palette to use for the colors.
- double [sampleDx](#)
precision required to consider a point on a segment.
- bool [strictCellWallMin](#)
If true, the `cellWallMin` property is strictly enforced.

Protected Member Functions

- std::map< [junction](#), double > [calcQuads](#) (const [cell](#) &c)
Compute parameters for the width of the corners.

```
template<typename Model, typename CellContent, typename JunctionContent,
typename WallContent = graph::_EmptyEdgeContent, typename CellEdge-
Content = graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_-
EmptyEdgeContent, bool compact = false, typename LeafClass = template_-
utils::this_class> class tissue::Tissue< Model, CellContent, JunctionContent,
WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent,
compact, LeafClass >
```

Class handling the development and representation of a cell tissue.

Parameters

VertexContent **Data** structure for the vertex content

TissueEdgeContent **Data** structure for the edge content of the tissue graph

CellEdgeContent **Data** structure for the edge content of the cell graph

Model Type of your model

Definition at line 527 of file tissue.h.

17.62.2 Member Typedef Documentation

17.62.2.1 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent
 = graph::_EmptyEdgeContent, bool compact = false,
 typename LeafClass = template_utils::this_class> typedef
 VVComplex::division_data tissue::Tissue< Model,
 CellContent, JunctionContent, WallContent, CellEdgeContent,
 CellJunctionContent, JunctionCellContent, compact, LeafClass
 >::division_data`

Type of the division data used for the complex.

Reimplemented from `vvcomplex::VVComplex< MANDATORY_COMPLEX_-
 TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_-
 COMPLEX_TEMPLATE_ARGS >>>.`

Definition at line 534 of file tissue.h.

17.62.3 Constructor & Destructor Documentation

17.62.3.1 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> tissue::Tissue< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::Tissue (Model * mod) [inline]`

Constructor with just the model.

Definition at line 624 of file tissue.h.

```

625      : VVComplex(mod)
626      , cellDivAlg(SHORT_WALL)
627      , cellPinch(0)
628      , cellMaxPinch(0)
629      , cellWallMin(0)
630      , strictCellWallMin(true)
631      , cellWallSample(100)
632      , sampleDx(0.1)
633      , palette(NULL)
634      , colorBegin(0)
635      , colorEnd(1)
636      , colorCenter(0.2)
637      , contourColor(3)
638      , cellWallCorner(0.15)
639      , drawInsides(true)
640      , drawBorders(true)
641      , blending(1)
642      , center_blending(1)
643      , cellWallWidth(0.2)
644      { }
```

17.62.3.2 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> tissue::Tissue< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::Tissue (const util::Palette & pal, Model * mod) [inline]`

Constructor setting up both the palette and the model.

Definition at line 649 of file tissue.h.

```

650      : VVComplex(mod)
```

17.62 tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference

527

```

651     , cellDivAlg(SHORT_WALL)
652     , cellPinch(0)
653     , cellMaxPinch(0)
654     , cellWallMin(0)
655     , strictCellWallMin(true)
656     , cellWallSample(100)
657     , sampleDx(0.1)
658     , palette(&pal)
659     , colorBegin(0)
660     , colorEnd(1)
661     , colorCenter(0.2)
662     , contourColor(3)
663     , cellWallCorner(0.15)
664     , drawInsides(true)
665     , drawBorders(true)
666     , blending(1)
667     , center_blending(1)
668     , cellWallWidth(0.2)
669     { }
```

17.62.3.3 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> tissue::Tissue< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::Tissue (const Tissue< Model, CellContent, JunctionContent,
WallContent, CellEdgeContent, CellJunctionContent,
JunctionCellContent, compact, LeafClass > ©) [inline]`

Copy constructor.

Definition at line 674 of file tissue.h.

```

675     : VVComplex(copy)
676     , cellDivAlg(copy.cellDivAlg)
677     , cellPinch(copy.cellPinch)
678     , cellMaxPinch(copy.cellMaxPinch)
679     , cellWallMin(copy.cellWallMin)
680     , strictCellWallMin(copy.strictCellWallMin)
681     , cellWallSample(copy.cellWallSample)
682     , sampleDx(copy.sampleDx)
683     , palette(copy.palette)
684     , colorBegin(copy.colorBegin)
685     , colorEnd(copy.colorEnd)
686     , colorCenter(copy.colorCenter)
687     , contourColor(copy.contourColor)
688     , cellWallCorner(copy.cellWallCorner)
689     , drawInsides(copy.drawInsides)
690     , drawBorders(copy.drawBorders)
691     , blending(copy.blending)
692     , center_blending(copy.center_blending)
693     , cellWallWidth(copy.cellWallWidth)
694     { }
```

17.62.4 Member Function Documentation

17.62.4.1 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> std::map<junction,double>
tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent,
compact, LeafClass >::calcQuads (const cell & c) [inline,
protected]`

Compute parameters for the width of the corners.

Warning

For internal use only

Definition at line 1305 of file tissue.h.

References forall, vvcomplex::VVComplex< MANDATORY_COMPLEX_-
TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_-
COMPLEX_TEMPLATE_ARGS >)>::model, graph::VVBIGraph< Ver-
tex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact
>::neighbors(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content,
Edge2Content_, compact >::nextTo(), graph::VVBIGraph< Vertex1Content,
Vertex2Content, Edge1Content, Edge2Content_, compact >::prevTo(), and
vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_-
ARGS >)>::S.

Referenced by tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass
>::drawCell(), and tissue::Tissue< Model, CellContent, JunctionContent, WallCon-
tent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass
>::drawWalledCell().

```

1306     {
1307         std::map<junction,double> quadsWidth;
1308         forall(const junction& k, this->S.neighbors(c))
1309         {
1310             const junction& l = this->S.nextTo(c, k);
1311             const junction& j = this->S.prevTo(c, k);
1312
1313             // Get vectors towards neighbors
1314             Point3d kpos = this->model->position(k);
1315             Point3d kj = this->model->position(j) - kpos;
1316             Point3d kl = this->model->position(l) - kpos;
1317             Point3d kc = this->model->position(c) - kpos;
1318
1319             // Info for quads
1320             double lsz = norm(kc ^ util::normalized(kl)); // norm(kc - kl * (kc * kl)

```

```

);
1321         if(lsz > 0)
1322             lsz = norm(kc)/lsz;
1323         double jsz = norm(kc ^ util::normalized(kj)); // norm(kc - kj * (kc * kj)
);
1324         if(jsz > 0)
1325             jsz = norm(kc)/jsz;
1326         quadsWidth[k] = max(lsz, jsz);
1327     }
1328     return quadsWidth;
1329 }
1330 }
```

17.62.4.2 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> division_result_t
tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent,
compact, LeafClass >::divideCell (const cell & to_divide, const cell &
cell_kept) [inline]`

Divide a cell.

Convenience function where the algorithm used is the default one and one cell is enough to setup the data of the new cells.

Definition at line 931 of file tissue.h.

References `vvcomplex::FindCenter()`.

```

933     {
934         std::vector<cell> k;
935         k.push_back(cell_kept);
936         division_result_t res = this->divideCell(to_divide, k);
937         if(res)
938         {
939             FindCenter(res.cl, static_cast<leaf_class*>(*this));
940             FindCenter(res.cr, static_cast<leaf_class*>(*this));
941         }
942         return res;
943     }
```

17.62.4.3 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> division_result_t
tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent,
compact, LeafClass >::divideCell (const cell & to_divide) [inline]`

Divide a cell.

Convenience function where the algorithm used is the default one and no context is required to setup the data of the new cells.

Definition at line 913 of file tissue.h.

References `vvcomplex::FindCenter()`.

```

914     {
915         std::vector<cell> k;
916         division_result_t res = this->divideCell(to_divide, k);
917         if(res)
918         {
919             FindCenter(res.cl, static_cast<leaf_class>(&(*this)));
920             FindCenter(res.cr, static_cast<leaf_class>(&(*this)));
921         }
922         return res;
923     }

```

17.62.4.4 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> template<typename
CellContainer > division_result_t tissue::Tissue< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::divideCell (const cell & to_divide, const CellContainer &
kept_cells) [inline]`

Divide a cell.

Convenience function where the algorithm used is the default one.

Definition at line 890 of file tissue.h.

References `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellDivAlg`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent,`

17.62 tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference **531**
 CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass
 >::getClosestMidAlgoParams(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getCloseWallAlgoParams(), and tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getShortWallAlgoParams().

```

892     {
893         switch(cellDivAlg)
894         {
895             case CLOSEST_MID:
896                 return divideCell(to_divide, getClosestMidAlgoParams(), kept_cells);
897             case SHORT_WALL:
898                 return divideCell(to_divide, getShortWallAlgoParams(), kept_cells);
899             case CLOSEST_WALL:
900                 return divideCell(to_divide, getCloseWallAlgoParams(), kept_cells);
901             default:
902                 util::err << "Error, unknown algorithm specified ... doing nothing." <<
endl;
903         }
904         return division_result_t();
905     }

```

17.62.4.5 **template<typename Model , typename CellContent , typename JunctionContent , typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> template<typename AlgoParameter > division_result_t tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell (const cell & to_divide, const AlgoParameter & params, const cell & cell_kept) [inline]**

Divide a cell.

Convenience function where a single cell is to be kept

Reimplemented from [vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS\(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >>>](#).

Definition at line 871 of file tissue.h.

References [vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell\(\)](#), and [vvcomplex::FindCenter\(\)](#).

```

874     {
875         division_result_t res = VVComplex::divideCell(to_divide, params, cell_kept)
;
876         if(res)

```

```

877     {
878         FindCenter(res.cl, static_cast<leaf_class*>(*this));
879         FindCenter(res.cr, static_cast<leaf_class*>(*this));
880     }
881     return res;
882 }

```

17.62.4.6 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> template<typename
AlgoParameter , typename CellContainer > division_result_t
tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent,
compact, LeafClass >::divideCell (const cell & c, const
AlgoParameter & params, const CellContainer & to_keep)
[inline]`

Divide a cell of the tissue.

Parameters

c Cell to divide

params Parameter of the algorithm to use. The type of the params will decide which algorithm to use.

to_keep Set of cells to keep to update the data in the new cells

See also

[updateFromOld](#)

Reimplemented from `vvcomplex::VVComplex< MANDATORY_COMPLEX_-
TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_-
COMPLEX_TEMPLATE_ARGS >)>`.

Definition at line 852 of file tissue.h.

References `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell()`, and `vvcomplex::FindCenter()`.

```

855     {
856         division_result_t res = VVComplex::divideCell(c, params, to_keep);
857         if(res)
858         {
859             FindCenter(res.cl, static_cast<leaf_class*>(*this));
860             FindCenter(res.cr, static_cast<leaf_class*>(*this));
861         }
862         return res;
863     }

```

17.62 `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >` **Class Template Reference** 533

17.62.4.7 `template<typename Model , typename CellContent , typename JunctionContent , typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> division_result_t tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell (const cell & c, const division_data & ddata, const cell & to_keep) [inline]`

Divide a cell with the cells to keep defined by any kind of container.

Reimplemented from `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>`.

Definition at line 827 of file `tissue.h`.

References `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell()`, and `vvcomplex::FindCenter()`.

```

830     {
831         division_result_t res = VVComplex::divideCell(c, ddata, to_keep);
832         if(res)
833         {
834             FindCenter(res.cl, static_cast<leaf_class*>(*this));
835             FindCenter(res.cr, static_cast<leaf_class*>(*this));
836         }
837         return res;
838     }

```

17.62.4.8 `template<typename Model , typename CellContent , typename JunctionContent , typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> template<typename CellContainer > division_result_t tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell (const cell & c, const division_data & ddata, const CellContainer & to_keep) [inline]`

Divide a cell with the cells to keep defined by any kind of container.

Reimplemented from `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>`.

Definition at line 811 of file tissue.h.

References `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell()`, and `vvcomplex::FindCenter()`.

```

814      {
815          division_result_t res = VVComplex::divideCell(c, ddata, to_keep);
816          if(res)
817          {
818              FindCenter(res.cl, static_cast<leaf_class*>(*this));
819              FindCenter(res.cr, static_cast<leaf_class*>(*this));
820          }
821          return res;
822      }

```

17.62.4.9 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> void tissue::Tissue< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::drawCell (const cell & c, double value) [inline]`

Draw a single cell.

The value is used to setup the color (using `colorBegin`, `colorEnd`) if the interior of the cell is drawn.

If `center_color` is specified, then it is used instead of the usual modification of the color of the cell color.

Definition at line 1260 of file tissue.h.

References `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawInsides`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::model`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::valueCenterColor()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::valueColor()`.

```

1261      {
1262          Point3d cpos = this->model->position(c);
1263
1264          if(!drawInsides)

```

17.62 tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference 535

```

1265     {
1266         value = 0;
1267     }
1268
1269     // Draw cell faces
1270     glNormal3dv(this->model->normal(c).c_data());
1271
1272     drawCell(c, valueColor(value), valueCenterColor(value));
1273 }
```

17.62.4.10 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> void tissue::Tissue< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::drawCell (const cell & c, util::Palette::Color cell_color,
util::Palette::Color center_color) [inline]`

Draw a single cell.

The colors used for the cell are specified by `cell_color` and `center_color`.

Definition at line 1116 of file `tissue.h`.

References `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::blending`, `util::Vector< dim, T >::c_data()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::calcQuads()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellWallCorner`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellWallWidth`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::center_blending`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::colorEnd`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::contourColor`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawBorders`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawInsides`, `forall`, `util::Palette::getColor()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::model`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact`

>::nextTo(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::palette, graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::prevTo(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::S, and util::Palette::useColor().

Referenced by tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell().

```

1118     {
1119         Point3d cpos = this->model->position(c);
1120
1121         if(!drawInsides)
1122         {
1123             cell_color = palette->getColor(colorEnd, blending);
1124             center_color = palette->getColor(colorEnd, center_blending);
1125         }
1126
1127         // Draw cell faces
1128         glNormal3dv(this->model->normal(c).c_data());
1129
1130         glPolygonMode(GL_FRONT, GL_FILL);
1131         glPolygonOffset(3.0, 1.0);
1132         glBegin(GL_TRIANGLE_FAN);
1133         glColor4fv(center_color.c_data());
1134         glVertex3dv(cpos.c_data());
1135         glColor4fv(cell_color.c_data());
1136         junction fv(0);
1137         forall(const junction& k, this->S.neighbors(c))
1138         {
1139             if(fv.isNull())
1140                 fv = k;
1141             glVertex3dv(this->model->position(k).c_data());
1142         }
1143         glVertex3dv(this->model->position(fv).c_data());
1144         glEnd();
1145
1146         const std::map<junction,double>& quadsWidth = calcQuads(c);
1147
1148         if(drawBorders)
1149         {
1150             palette->useColor(contourColor);
1151             glPolygonOffset(-1.0, -1.0);
1152             forall(const junction& k, this->S.neighbors(c))
1153             {
1154                 // Get vectors towards neighbors and normalize
1155                 const junction& j = this->S.prevTo(c, k);
1156                 const junction& l = this->S.nextTo(c, k);
1157
1158                 const Point3d& jpos = this->model->position(j);
1159                 const Point3d& kpos = this->model->position(k);
1160                 const Point3d& lpos = this->model->position(l);
1161
1162                 Point3d kj = jpos - kpos;
1163                 Point3d kl = lpos - kpos;
1164                 Point3d kc = cpos - kpos;
1165                 Point3d jc = cpos - jpos;
1166                 Point3d kjn = kj, kln = kl, kcn = kc, jcn = jc;
1167                 kjn /= norm(kjn);

```

**17.62 tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent, compact,
LeafClass > Class Template Reference**

537

```

1168         kln /= norm(kln);
1169         kcn /= norm(kcn);
1170         jcn /= norm(jcn);
1171
1172         // Corner size along cell wall, clip to wall length
1173         double kjl = norm(kj);
1174         if(kjl > cellWallCorner)
1175             kjl = cellWallCorner;
1176         double kll = norm(kl);
1177         if(kll > cellWallCorner)
1178             kll = cellWallCorner;
1179
1180         double kcl = .75 * cellWallCorner * (kjn * kcn + kln * kcn)/2.0;
1181
1182         // Width of quads for walls
1183         double kcqw = quadsWidth.find(k)->second * cellWallWidth;
1184         double jcqw = quadsWidth.find(j)->second * cellWallWidth;
1185
1186         // Adjust corner centers if required
1187         bool ccvx; // convex
1188         ccvx = ((kll*kln - kcl*kcn)^(kjl*kjn - kcl*kcn)) * this->model->normal(
k) > 0;
1189         if(!ccvx || kcl < kcqw)
1190             kcl = kcqw;
1191
1192         Point3d p;
1193
1194         // Draw corners
1195         glBegin(GL_TRIANGLE_FAN);
1196         glVertex3dv(kpos.c_data());
1197         p = kpos + kll * kln;
1198         glVertex3dv(p.c_data());
1199         p = kpos + kcl * kcn;
1200         glVertex3dv(p.c_data());
1201         p = kpos + kjl * kjn;
1202         glVertex3dv(p.c_data());
1203         glEnd();
1204
1205         // Draw cell walls
1206         glBegin(GL_QUADS);
1207         glVertex3dv(jpos.c_data());
1208         glVertex3dv(kpos.c_data());
1209         p = kpos + kcqw * kcn;
1210         glVertex3dv(p.c_data());
1211         p = jpos + jcqw * jcn;
1212         glVertex3dv(p.c_data());
1213         glEnd();
1214     }
1215 }
1216 }
```

```

17.62.4.11 template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> void tissue::Tissue< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::drawCellContour (const cell & c)  [inline]

```

Draw just the contour of the cells.

This function was extracted as, due to a bug in latest NVidia drivers, alternating between LINE and FILL to draw polygon crashes text rendering.

Definition at line 1225 of file tissue.h.

References tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::contourColor, forall, vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::model, graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::palette, vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::S, and util::Palette::useColor().

```

1226     {
1227         palette->useColor(contourColor);
1228         glPolygonMode(GL_FRONT, GL_LINE);
1229         glPolygonOffset(1.0, 5.0);
1230         glBegin(GL_POLYGON);
1231         forall(const junction& n, this->S.neighbors(c))
1232         {
1233             glVertex3dv(this->model->position(n).c_data());
1234         }
1235         glEnd();
1236     }

```


17.62 **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**, **compact**,
LeafClass > **Class Template Reference** 539

17.62.4.12 **template**<**typename Model** , **typename CellContent** ,
typename JunctionContent , **typename WallContent** =
graph::_EmptyEdgeContent, **typename CellEdgeContent** =
graph::_EmptyEdgeContent, **typename CellJunctionContent** =
graph::_EmptyEdgeContent, **typename JunctionCellContent** =
graph::_EmptyEdgeContent, **bool compact** = **false**, **typename**
LeafClass = **template_utils::this_class**> **void tissue::Tissue**< **Model**,
CellContent, **JunctionContent**, **WallContent**, **CellEdgeContent**,
CellJunctionContent, **JunctionCellContent**, **compact**, **LeafClass**
>::drawSimplifiedCell (**const cell & c**) [**inline**]

Draw the cell as a simple colored polygon.

Useful for quick drawing of the shape, like for selection system.

Definition at line 1243 of file tissue.h.

References forall, vvcomplex::VVComplex< MANDATORY_COMPLEX_
TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_
COMPLEX_TEMPLATE_ARGS >)>::model, graph::VVBigraph< Vertex1Content,
Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors(), and
vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_
ARGS >)>::S.

```
1244     {
1245         glBegin(GL_POLYGON);
1246         forall(const junction& n, this->S.neighbors(c))
1247         {
1248             glVertex3dv(this->model->position(n).c_data());
1249         }
1250         glEnd();
1251     }
```

17.62.4.13 **template**<**typename Model** , **typename CellContent** ,
typename JunctionContent , **typename WallContent** =
graph::_EmptyEdgeContent, **typename CellEdgeContent** =
graph::_EmptyEdgeContent, **typename CellJunctionContent** =
graph::_EmptyEdgeContent, **typename JunctionCellContent** =
graph::_EmptyEdgeContent, **bool compact** = **false**, **typename**
LeafClass = **template_utils::this_class**> **void tissue::Tissue**< **Model**,
CellContent, **JunctionContent**, **WallContent**, **CellEdgeContent**,
CellJunctionContent, **JunctionCellContent**, **compact**, **LeafClass**
>::drawWalledCell (**const cell & c**, **double value**) [**inline**]

Draw a cell with walls, specifying the value.

Definition at line 1094 of file tissue.h.

References tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass
>::drawInsides, tissue::Tissue< Model, CellContent, JunctionContent, Wall-
Content, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact,

LeafClass >::drawWalledCell(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::model, tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::valueCenterColor(), and tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::valueColor().

```

1096     {
1097         Point3d cpos = this->model->position(c);
1098
1099         if(!drawInsides)
1100         {
1101             value = 0;
1102         }
1103
1104         // Draw cell faces
1105         glNormal3dv(this->model->normal(c).c_data());
1106
1107         drawWalledCell(c, valueColor(value), valueCenterColor(value));
1108     }

```

17.62.4.14 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> void tissue::Tissue< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::drawWalledCell (const cell & c, util::Palette::Color cell_color,
util::Palette::Color center_color) [inline]`

Draw a single cell with differentiated walls.

The colors used for the cell are specified by `cell_color` and `center_color`. The color used by the wall depend on the value on the wall.

This method requires two new methods:

```
util::Palette::Color cellWallColor(const vertex& cell, const vertex& wall_junction)
```

```
double cellWallOrder(const vertex& cell, const vertex& wall_junction)
```

The wall considered is the one from `wall_junction` to the next vertex in the neighborhood of cell in the VVGraph. For the order, the value has to be between 0 and 1. A value of 1 will put the representation of the wall on top.

Definition at line 983 of file `tissue.h`.

References `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::blending`, `util::Vector< dim, T >::c_data()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent,`

17.62 tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact,

LeafClass > Class Template Reference

541

JunctionCellContent, compact, LeafClass >::calcQuads(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellWallCorner, tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellWallWidth, tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::center_blending, tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::colorEnd, tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawBorders, tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawInsides, forall, util::Palette::getColor(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::model, graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors(), graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::palette, graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::prevTo(), and vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::S.

Referenced by tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawWalledCell().

```

986     {
987         Point3d cpos = this->model->position(c);
988
989         if(!drawInsides)
990         {
991             cell_color = palette->getColor(colorEnd, blending);
992             center_color = palette->getColor(colorEnd, center_blending);
993         }
994
995         // Draw cell faces
996         glNormal3dv(this->model->normal(c).c_data());
997
998         glPolygonMode(GL_FRONT, GL_FILL);
999         glPolygonOffset(3.0, 1.0);
1000         glBegin(GL_TRIANGLE_FAN);
1001         glColor4fv(center_color.c_data());
1002         glVertex3dv(cpos.c_data());
1003         glColor4fv(cell_color.c_data());
1004         junction fv(0);
1005         forall(const junction& k, this->S.neighbors(c))
1006         {
1007             if(fv.isNull())
1008                 fv = k;
1009             glVertex3dv(this->model->position(k).c_data());
1010         }
1011         glVertex3dv(this->model->position(fv).c_data());

```

```

1012     glEnd();
1013
1014     const std::map<junction,double>& quadsWidth = calcQuads(c);
1015
1016     glPolygonOffset(-1.0, -1.0);
1017     forall(const junction& k, this->S.neighbors(c))
1018     {
1019         const junction& l = this->S.nextTo(c, k);
1020         const junction& j = this->S.prevTo(c, k);
1021         // Get vectors towards neighbors and normalize
1022         Point3d jpos = this->model->position(j);
1023         Point3d kpos = this->model->position(k);
1024
1025         if(drawBorders)
1026         {
1027             util::Palette::Color col = this->model->cellWallColor(c, j);
1028             double j_order = 2*this->model->cellWallOrder(c, j);
1029             double l_order = 2*this->model->cellWallOrder(c, l);
1030             double order = std::min(j_order, l_order);
1031             Point3d lpos = this->model->position(l);
1032             Point3d kj = jpos - kpos;
1033             Point3d kl = lpos - kpos;
1034             Point3d kc = cpos - kpos;
1035             Point3d jc = cpos - jpos;
1036             Point3d kjn = kj, kln=kl, kcn = kc, jcn = jc;
1037             kjn /= norm(kjn);
1038             kln /= norm(kln);
1039             kcn /= norm(kcn);
1040             jcn /= norm(jcn);
1041
1042             // Corner size along cell wall, clip to wall length
1043             double kjl = norm(kj);
1044             if(kjl > cellWallCorner)
1045                 kjl = cellWallCorner;
1046             double kll = norm(kl);
1047             if(kll > cellWallCorner)
1048                 kll = cellWallCorner;
1049
1050             double kcl = .75 * cellWallCorner * exp((kjn * kcn + kln * kcn)/2.0);
1051
1052             // Width of quads for walls
1053             double kcqw = quadsWidth.find(k)->second * cellWallWidth;
1054             double jcqw = quadsWidth.find(j)->second * cellWallWidth;
1055
1056             // Adjust corner centers if required
1057             bool ccvx; // convex
1058             ccvx = ((kll*kln - kcl*kcn)^(kjl*kjn - kcl*kcn)) * this->model->normal(
k) > 0;
1059             if(!ccvx || kcl < kcqw)
1060                 kcl = kcqw;
1061
1062             Point3d p;
1063
1064             // Draw corners
1065             glPolygonOffset((1.0-order)+1, -3.0);
1066             glColor4fv(col.c_data());
1067             glBegin(GL_TRIANGLE_FAN);
1068             glVertex3dv(kpos.c_data());
1069             p = kpos + kll * kln;
1070             glVertex3dv(p.c_data());
1071             p = kpos + kcl * kcn;
1072             glVertex3dv(p.c_data());

```

17.62 tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference

543

```

1073         p = kpos + kjl * kjn;
1074         glVertex3dv(p.c_data());
1075         glEnd();
1076
1077         // Draw cell walls
1078         glPolygonOffset((1.0-j_order)+1, -3.0);
1079         glBegin(GL_QUADS);
1080         glVertex3dv(jpos.c_data());
1081         glVertex3dv(kpos.c_data());
1082         p = kpos + kcqw * kcn;
1083         glVertex3dv(p.c_data());
1084         p = jpos + jcqw * jcn;
1085         glVertex3dv(p.c_data());
1086         glEnd();
1087     }
1088 }
1089 }
```

17.62.4.15 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> CellPinchingParams
tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent,
compact, LeafClass >::getCellPinchingParams () [inline]`

Returns the cell pinching parameter object.

Definition at line 760 of file tissue.h.

References `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellMaxPinch`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellPinch`.

```

761     {
762         return CellPinchingParams(cellPinch, cellMaxPinch);
763     }
```

17.62.4.16 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> ClosestMidAlgoParams
 tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
 CellEdgeContent, CellJunctionContent, JunctionCellContent,
 compact, LeafClass >::getClosestMidAlgoParams () [inline]`

Returns the closest mid algorithm parameter object.

Definition at line 778 of file tissue.h.

References `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellWallMin`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::setCellPinchingParams()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::strictCellWallMin`.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell()`.

```

779     {
780         ClosestMidAlgoParams p(cellWallMin, strictCellWallMin);
781         setCellPinchingParams(p);
782         return p;
783     }
```

17.62.4.17 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> ClosestWallAlgoParams
 tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
 CellEdgeContent, CellJunctionContent, JunctionCellContent,
 compact, LeafClass >::getCloseWallAlgoParams () [inline]`

Returns the closest wall algorithm parameter object.

Definition at line 788 of file tissue.h.

References `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellWallMin`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass`

17.62 tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference 545

~~>::setCellPinchingParams()~~, and ~~tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::strictCellWallMin.~~

Referenced by ~~tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell()~~.

```

789     {
790         ClosestWallAlgoParams p(cellWallMin, strictCellWallMin);
791         setCellPinchingParams(p);
792         return p;
793     }

```

17.62.4.18 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> ShortWallAlgoParams
tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent,
compact, LeafClass >::getShortWallAlgoParams () [inline]`

Returns the shortest wall algorithm parameter object.

Definition at line 768 of file tissue.h.

References ~~tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellWallMin~~, ~~tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellWallSample~~, ~~tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::sampleDx~~, ~~tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::setCellPinchingParams()~~, and ~~tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::strictCellWallMin.~~

Referenced by ~~tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell()~~.

```

769     {
770         ShortWallAlgoParams p(cellWallMin, strictCellWallMin, cellWallSample,  
sampleDx);
771         setCellPinchingParams(p);
772         return p;
773     }

```

17.62.4.19 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> void tissue::Tissue< Model,
 CellContent, JunctionContent, WallContent, CellEdgeContent,
 CellJunctionContent, JunctionCellContent, compact, LeafClass
 >::postDraw () [inline]`

Restore the OpenGL context after the drawing.

Definition at line 958 of file tissue.h.

```

959     {
960         glDisable (GL_POLYGON_OFFSET_FILL);
961         glDisable (GL_POLYGON_OFFSET_LINE);
962     }
```

17.62.4.20 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> void tissue::Tissue< Model,
 CellContent, JunctionContent, WallContent, CellEdgeContent,
 CellJunctionContent, JunctionCellContent, compact, LeafClass
 >::preDraw () [inline]`

Setup the OpenGL context to draw the cells.

Definition at line 948 of file tissue.h.

```

949     {
950         glEnable (GL_POLYGON_OFFSET_FILL);
951         glEnable (GL_POLYGON_OFFSET_LINE);
952         glLineWidth (1.0);
953     }
```


17.62 **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass** > **Class Template Reference** 547

17.62.4.21 **template**<typename **Model** , typename **CellContent** ,
 typename **JunctionContent** , typename **WallContent** =
graph::_EmptyEdgeContent, typename **CellEdgeContent** =
graph::_EmptyEdgeContent, typename **CellJunctionContent** =
graph::_EmptyEdgeContent, typename **JunctionCellContent** =
graph::_EmptyEdgeContent, bool **compact** = false, typename
LeafClass = **template_utils::this_class**> void **tissue::Tissue**< **Model**,
CellContent, **JunctionContent**, **WallContent**, **CellEdgeContent**,
CellJunctionContent, **JunctionCellContent**, **compact**, **LeafClass**
 >::**readParms** (**util::Parms** & *parms*, const **QString** & *section*)
[inline]

Read the parameters for the algorithmic part of the object.

Parameters

parms Parameter object to read from
section Section containing the parameters

Definition at line 702 of file **tissue.h**.

References **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass** >::**cellDivAlg**, **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass** >::**cellMaxPinch**, **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass** >::**cellPinch**, **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass** >::**cellWallMin**, **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass** >::**cellWallSample**, **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass** >::**sampleDx**, **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass** >::**strictCellWallMin**, and **QString::toLower**().

```

703     {
704         QString algo;
705         parms(section, "DivisionAlgorithm", algo);
706         algo = algo.toLower();
707         if( algo == "closestmid" )
708             cellDivAlg = CLOSEST_MID;
709         else if(algo == "shortwall")
710             cellDivAlg = SHORT_WALL;
711         else if(algo == "closestwall")
712             cellDivAlg = CLOSEST_WALL;
713         else
714         {
715             util::err << "Invalid division algorithm, should be ClosestMid, ShortWall
or ClosestWall. Choosing ShortWall by default." << endl;
716             cellDivAlg = SHORT_WALL;
717         }

```

```

718     parms(section, "CellPinch", cellPinch);
719     parms(section, "CellMaxPinch", cellMaxPinch);
720     parms(section, "CellWallMin", cellWallMin);
721     parms(section, "StrictCellWallMin", strictCellWallMin);
722     parms(section, "CellWallSample", cellWallSample);
723     parms(section, "CellSampleDx", sampleDx);
724 }

```

17.62.4.22 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph:: EmptyEdgeContent, typename CellEdgeContent =
graph:: EmptyEdgeContent, typename CellJunctionContent =
graph:: EmptyEdgeContent, typename JunctionCellContent =
graph:: EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> void tissue::Tissue< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::readViewParms (util::Parms & parms, const QString & section)
[inline]`

Read the parameters for the drawing of a cell.

Parameters

parms Parameter object to read from

section Section containing the parameters

Definition at line 732 of file tissue.h.

References `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::blending`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellWallCorner`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellWallWidth`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::center_blending`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::colorBegin`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::colorCenter`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::colorEnd`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::contourColor`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawBorders`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawInsides`.

17.62 tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference

549

```

733     {
734         parms(section, "CellColorBegin", colorBegin);
735         parms(section, "CellColorEnd", colorEnd);
736         parms(section, "CellColorCenter", colorCenter);
737         parms(section, "CellContourColor", contourColor);
738         parms(section, "CellWallCorner", cellWallCorner);
739         parms(section, "DrawInsideCells", drawInsides);
740         parms(section, "DrawCellBorders", drawBorders);
741         parms(section, "CellBlending", blending);
742         parms(section, "CellCenterBlending", center_blending);
743         parms(section, "CellWallWidth", cellWallWidth);
744     }

```

17.62.4.23 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> void tissue::Tissue< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::setCellPinchingParams (CellPinchingParams & params)
[inline]`

Set the pinching parameters to be used.

By default, the parameters will be read from the parameter file

Definition at line 751 of file tissue.h.

References tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellMaxPinch, tissue::CellPinchingParams::cellMaxPinch, tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cellPinch, and tissue::CellPinchingParams::cellPinch.

Referenced by tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getClosestMidAlgoParams(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getCloseWallAlgoParams(), and tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getShortWallAlgoParams().

```

752     {
753         params.cellPinch = cellPinch;
754         params.cellMaxPinch = cellMaxPinch;
755     }

```

17.62.4.24 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> util::Palette::Color
 tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
 CellEdgeContent, CellJunctionContent, JunctionCellContent,
 compact, LeafClass >::valueCenterColor (double value)
 [inline]`

Get the center color corresponding to a given value.

Definition at line 1289 of file tissue.h.

References `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::blending`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::colorBegin`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::colorCenter`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::colorEnd`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::palette`, and `util::Palette::selectColor()`.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawWalledCell()`.

```

1290     {
1291         if(value > 1.0)
1292             value = 1.0;
1293         value *= (1.0 - colorCenter);
1294         double value_center = value + colorCenter;
1295         return palette->selectColor(colorBegin, colorEnd, value_center, blending);
1296     }
```

17.62 **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**, **compact**,
LeafClass > **Class Template Reference** 551
17.62.4.25 **template**<typename **Model** , typename **CellContent** ,
typename **JunctionContent** , typename **WallContent** =
graph::_EmptyEdgeContent, typename **CellEdgeContent** =
graph::_EmptyEdgeContent, typename **CellJunctionContent** =
graph::_EmptyEdgeContent, typename **JunctionCellContent** =
graph::_EmptyEdgeContent, bool **compact** = **false**, typename
LeafClass = **template_utils::this_class**> **util::Palette::Color**
tissue::Tissue< **Model**, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**,
compact, **LeafClass** >::**valueColor** (**double value**) [**inline**]

Get the color corresponding to a given value.

Definition at line 1278 of file `tissue.h`.

References `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`,
`CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass`
>::`blending`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`,
`CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass`
>::`colorBegin`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`,
`CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass`
>::`colorCenter`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallCon-`
`tent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `Leaf-`
`Class` >::`colorEnd`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallCon-`
`tent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass`
>::`palette`, and `util::Palette::selectColor()`.

Referenced by `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`,
`CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass`
>::`drawCell()`, and `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallCon-`
`tent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass`
>::`drawWalledCell()`.

```

1279     {
1280         if (value > 1.0)
1281             value = 1.0;
1282         value *= (1.0 - colorCenter);
1283         return palette->selectColor(colorBegin, colorEnd, value, blending);
1284     }

```

17.62.5 Member Data Documentation

17.62.5.1 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> double tissue::Tissue<
Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent,
compact, LeafClass >::blending`

Blending for the exterior of the cell.

(i.e. alpha value)

Definition at line 609 of file tissue.h.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawWalledCell()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::readViewParms()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::valueCenterColor()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::valueColor()`.

17.62.5.2 `template<typename Model , typename CellContent ,
typename JunctionContent , typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent
= graph::_EmptyEdgeContent, bool compact = false,
typename LeafClass = template_utils::this_class>
CELL_DIVISION_ALGORITHM tissue::Tissue< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::cellDivAlg`

Default algorithm to use.

Definition at line 542 of file tissue.h.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass`

17.62.5.3 **template**<typename **Model** , typename **CellContent** ,
typename **JunctionContent** , typename **WallContent** =
graph::_EmptyEdgeContent, typename **CellEdgeContent** =
graph::_EmptyEdgeContent, typename **CellJunctionContent** =
graph::_EmptyEdgeContent, typename **JunctionCellContent** =
graph::_EmptyEdgeContent, bool **compact** = false, typename
LeafClass = **template_utils::this_class**> double **tissue::Tissue**<
Model, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**,
compact, **LeafClass** >::**cellMaxPinch**

Maximum cell pinching.

Definition at line 550 of file **tissue.h**.

Referenced by **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass**
>::getCellPinchingParams(), **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**,
WallContent, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**,
LeafClass >::readParms(), and **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**,
WallContent, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**,
LeafClass >::setCellPinchingParams().

17.62.5.4 **template**<typename **Model** , typename **CellContent** ,
typename **JunctionContent** , typename **WallContent** =
graph::_EmptyEdgeContent, typename **CellEdgeContent** =
graph::_EmptyEdgeContent, typename **CellJunctionContent** =
graph::_EmptyEdgeContent, typename **JunctionCellContent** =
graph::_EmptyEdgeContent, bool **compact** = false, typename
LeafClass = **template_utils::this_class**> double **tissue::Tissue**<
Model, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**,
compact, **LeafClass** >::**cellPinch**

Cell pinching ratio.

Definition at line 546 of file **tissue.h**.

Referenced by **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**, **compact**, **LeafClass**
>::getCellPinchingParams(), **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**,
WallContent, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**,
LeafClass >::readParms(), and **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**,
WallContent, **CellEdgeContent**, **CellJunctionContent**, **JunctionCellContent**, **compact**,
LeafClass >::setCellPinchingParams().

17.62.5.5 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> double tissue::Tissue<
 Model, CellContent, JunctionContent, WallContent,
 CellEdgeContent, CellJunctionContent, JunctionCellContent,
 compact, LeafClass >::cellWallCorner`

Size of the cell corners.

Definition at line 593 of file tissue.h.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawWalledCell()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::readViewParms()`.

17.62.5.6 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> double tissue::Tissue<
 Model, CellContent, JunctionContent, WallContent,
 CellEdgeContent, CellJunctionContent, JunctionCellContent,
 compact, LeafClass >::cellWallMin`

Minimum size of a cell wall.

Definition at line 554 of file tissue.h.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getClosestMidAlgoParams()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getCloseWallAlgoParams()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getShortWallAlgoParams()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::readParms()`.

17.62 **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**, **compact**,
LeafClass > **Class Template Reference** 555

17.62.5.7 **template**<**typename Model** , **typename CellContent** ,
typename JunctionContent , **typename WallContent** =
graph::_EmptyEdgeContent, **typename CellEdgeContent** =
graph::_EmptyEdgeContent, **typename CellJunctionContent** =
graph::_EmptyEdgeContent, **typename JunctionCellContent** =
graph::_EmptyEdgeContent, **bool compact** = **false**, **typename**
LeafClass = **template_utils::this_class**> **double tissue::Tissue**<
Model, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**,
compact, **LeafClass** >::**cellWallSample**

Number of samples to find the shortest wall.

Definition at line 563 of file `tissue.h`.

Referenced by `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`,
`CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass`
>::`getShortWallAlgoParams()`, and `tissue::Tissue`< `Model`, `CellContent`, `JunctionCon-`
`tent`, `WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `com-`
`pact`, `LeafClass` >::`readParms()`.

17.62.5.8 **template**<**typename Model** , **typename CellContent** ,
typename JunctionContent , **typename WallContent** =
graph::_EmptyEdgeContent, **typename CellEdgeContent** =
graph::_EmptyEdgeContent, **typename CellJunctionContent** =
graph::_EmptyEdgeContent, **typename JunctionCellContent** =
graph::_EmptyEdgeContent, **bool compact** = **false**, **typename**
LeafClass = **template_utils::this_class**> **double tissue::Tissue**<
Model, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**,
compact, **LeafClass** >::**cellWallWidth**

Width of the "thin" cell wall.

Definition at line 619 of file `tissue.h`.

Referenced by `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallCon-`
`tent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `Leaf-`
`Class` >::`drawCell()`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallCon-`
`tent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass`
>::`drawWalledCell()`, and `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `Wall-`
`Content`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `Leaf-`
`Class` >::`readViewParms()`.

17.62.5.9 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> double tissue::Tissue<
 Model, CellContent, JunctionContent, WallContent,
 CellEdgeContent, CellJunctionContent, JunctionCellContent,
 compact, LeafClass >::center_blending`

Blending for the center of the cell.

(i.e. alpha value)

Definition at line 615 of file tissue.h.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawWalledCell()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::readViewParms()`.

17.62.5.10 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> int tissue::Tissue< Model,
 CellContent, JunctionContent, WallContent, CellEdgeContent,
 CellJunctionContent, JunctionCellContent, compact, LeafClass
 >::colorBegin`

Color of the high values for the cell.

Definition at line 577 of file tissue.h.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::readViewParms()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::valueCenterColor()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::valueColor()`.

17.62 **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**, **compact**,
LeafClass > **Class Template Reference** 557

17.62.5.11 **template**<**typename Model** , **typename CellContent** ,
typename JunctionContent , **typename WallContent** =
graph::_EmptyEdgeContent, **typename CellEdgeContent** =
graph::_EmptyEdgeContent, **typename CellJunctionContent** =
graph::_EmptyEdgeContent, **typename JunctionCellContent** =
graph::_EmptyEdgeContent, **bool compact** = **false**,
typename LeafClass = **template_utils::this_class**> **double**
tissue::Tissue< **Model**, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**,
compact, **LeafClass** >::**colorCenter**

Increase in color of the center ($0 < \text{colorCenter} < 1$).

Definition at line 585 of file `tissue.h`.

Referenced by `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`,
`CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass`
>::`readViewParms()`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallCon-`
`tent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `Leaf-`
`Class` >::`valueCenterColor()`, and `tissue::Tissue`< `Model`, `CellContent`, `JunctionCon-`
`tent`, `WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `com-`
`compact`, `LeafClass` >::`valueColor()`.

17.62.5.12 **template**<**typename Model** , **typename CellContent** ,
typename JunctionContent , **typename WallContent** =
graph::_EmptyEdgeContent, **typename CellEdgeContent** =
graph::_EmptyEdgeContent, **typename CellJunctionContent** =
graph::_EmptyEdgeContent, **typename JunctionCellContent** =
graph::_EmptyEdgeContent, **bool compact** = **false**, **typename**
LeafClass = **template_utils::this_class**> **int** **tissue::Tissue**< **Model**,
CellContent, **JunctionContent**, **WallContent**, **CellEdgeContent**,
CellJunctionContent, **JunctionCellContent**, **compact**, **LeafClass**
>::**colorEnd**

Color of the low values for the cell.

Definition at line 581 of file `tissue.h`.

Referenced by `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallCon-`
`tent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `Leaf-`
`Class` >::`drawCell()`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `Wall-`
`Content`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `Leaf-`
`Class` >::`drawWalledCell()`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`,
`WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`,
`LeafClass` >::`readViewParms()`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`,
`WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`,
`LeafClass` >::`valueCenterColor()`, and `tissue::Tissue`< `Model`, `CellContent`, `Junction-`
`Content`, `WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`,
`compact`, `LeafClass` >::`valueColor()`.

17.62.5.13 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> int tissue::Tissue< Model,
 CellContent, JunctionContent, WallContent, CellEdgeContent,
 CellJunctionContent, JunctionCellContent, compact, LeafClass
 >::contourColor`

Color of the contour of the cell.

Definition at line 589 of file tissue.h.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCellContour()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::readViewParms()`.

17.62.5.14 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> bool tissue::Tissue< Model,
 CellContent, JunctionContent, WallContent, CellEdgeContent,
 CellJunctionContent, JunctionCellContent, compact, LeafClass
 >::drawBorders`

Draw thick corners if true, otherwise draw `cellWallWidth` pixel borders.

Definition at line 603 of file tissue.h.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawWalledCell()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::readViewParms()`.

17.62 **tissue::Tissue**< **Model**, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**, **compact**,
LeafClass > **Class Template Reference** 559

17.62.5.15 **template**<**typename Model** , **typename CellContent** ,
typename JunctionContent , **typename WallContent** =
graph::_EmptyEdgeContent, **typename CellEdgeContent** =
graph::_EmptyEdgeContent, **typename CellJunctionContent** =
graph::_EmptyEdgeContent, **typename JunctionCellContent** =
graph::_EmptyEdgeContent, **bool compact** = **false**, **typename**
LeafClass = **template_utils::this_class**> **bool tissue::Tissue**< **Model**,
CellContent, **JunctionContent**, **WallContent**, **CellEdgeContent**,
CellJunctionContent, **JunctionCellContent**, **compact**, **LeafClass**
>::**drawInsides**

The inside of the cell is drawn depending on the value is true, otherwise the end color is used.

Definition at line 598 of file `tissue.h`.

Referenced by `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass` >::`drawCell()`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass` >::`drawWalledCell()`, and `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass` >::`readViewParms()`.

17.62.5.16 **template**<**typename Model** , **typename CellContent** ,
typename JunctionContent , **typename WallContent** =
graph::_EmptyEdgeContent, **typename CellEdgeContent** =
graph::_EmptyEdgeContent, **typename CellJunctionContent** =
graph::_EmptyEdgeContent, **typename JunctionCellContent** =
graph::_EmptyEdgeContent, **bool compact** = **false**, **typename**
LeafClass = **template_utils::this_class**> **const util::Palette***
tissue::Tissue< **Model**, **CellContent**, **JunctionContent**, **WallContent**,
CellEdgeContent, **CellJunctionContent**, **JunctionCellContent**,
compact, **LeafClass** >::**palette**

Palette to use for the colors.

Definition at line 573 of file `tissue.h`.

Referenced by `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass` >::`drawCell()`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass` >::`drawCellContour()`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass` >::`drawWalledCell()`, `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass` >::`valueCenterColor()`, and `tissue::Tissue`< `Model`, `CellContent`, `JunctionContent`, `WallContent`, `CellEdgeContent`, `CellJunctionContent`, `JunctionCellContent`, `compact`, `LeafClass` >::`valueColor()`.

17.62.5.17 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent
 = graph::_EmptyEdgeContent, bool compact = false,
 typename LeafClass = template_utils::this_class> double
 tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
 CellEdgeContent, CellJunctionContent, JunctionCellContent,
 compact, LeafClass >::sampleDx`

precision required to consider a point on a segment.

Definition at line 567 of file tissue.h.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getShortWallAlgoParams()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::readParms()`.

17.62.5.18 `template<typename Model , typename CellContent ,
 typename JunctionContent , typename WallContent =
 graph::_EmptyEdgeContent, typename CellEdgeContent =
 graph::_EmptyEdgeContent, typename CellJunctionContent =
 graph::_EmptyEdgeContent, typename JunctionCellContent =
 graph::_EmptyEdgeContent, bool compact = false, typename
 LeafClass = template_utils::this_class> bool tissue::Tissue< Model,
 CellContent, JunctionContent, WallContent, CellEdgeContent,
 CellJunctionContent, JunctionCellContent, compact, LeafClass
 >::strictCellWallMin`

If true, the cellWallMin property is strictly enforced.

If false, it's loosely enforced, meaning it will never lead to a change of cell wall.

Definition at line 559 of file tissue.h.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getClosestMidAlgoParams()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getCloseWallAlgoParams()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::getShortWallAlgoParams()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::readParms()`.

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

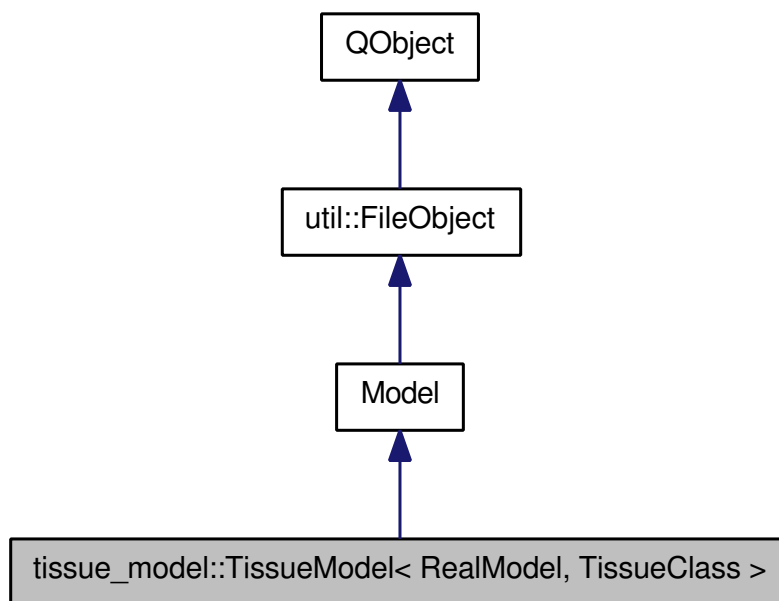
- [vvelib/algorithms/tissue.h](#)

17.63 `tissue_model::TissueModel< RealModel, TissueClass >` Struct Template Reference

Base class for the `tissue_model` helper.

```
#include <tissue_model.h>
```

Inheritance diagram for `tissue_model::TissueModel< RealModel, TissueClass >`:



Classes

- struct `CompareSize`
Operator class to compare the size of cells.

Public Types

- typedef `std::set< cell, CompareSize >` `ordered_cells_t`
Ordered set of vertices.

Public Member Functions

- `IMPORT_COMPLEX_TYPES` (`TissueClass`)
- void `initialize` ()
Function to override to initialize the model.

- void [modifiedFiles](#) (const std::set< std::string > &filenames)
This function is called anytime one or more registered files are modified.
- virtual void [readParam](#) (util::Parms &)
Function to be redefined to read extra parameters.
- void [readTissueParam](#) ()
Read the parameters for the [tissue_model](#) helper (automatically called by the helper).
- void **registerFiles** ()
- void [step](#) ()
Default step method: do nothing.
- [TissueModel](#) (QObject *parent)
Default constructor.
- void [updateCellsArea](#) ()
Update the area of all the cells in the tissue.

Selection methods

- const cell & [cellFromId](#) (int i)
Return a reference of the cell whose id is i.
- void [drawWithNames](#) ()
Draw for cell selection.

Drawing methods

- void [draw](#) (Viewer *viewer)
Draw the tissue using the `ModelClass::getCellColor` and `ModelClass::getCellCenterColor` methods.
- virtual [Color](#) [getCellCenterColor](#) (const cell &c)
Color using the `value` attribute of the cell.
- virtual [Color](#) [getCellColor](#) (const cell &c)
Color using the `value` attribute of the cell.
- void [initDraw](#) ()
Convenience function used if the viewer is not necessary.
- void [postDraw](#) ()
Convenience function used if the viewer is not necessary.
- void [preDraw](#) ()

Convenience function used if the viewer is not necessary.

Methods required by the tissue library

- `Point3d normal` (const junction &v) const
- `Point3d normal` (const cell &v) const
- `Point3d position` (const junction &v) const
- `Point3d position` (const cell &v) const
- void `setPosition` (const junction &v, const `Point3d` &pos)
- void `setPosition` (const cell &v, const `Point3d` &pos)
- void `setVertexPositionHint` (const junction &, const junction &, const junction &, double)
- void `updateFromOld` (const cell &, const cell &, const cell &, const type-name `TissueClass::division_data` &, `TissueClass` &)

Public Attributes

- int `backColor`
Color of the background.
- bool `drawNeighborhood`
Draw the neighborhood as arrows ?
- `util::Palette` `palette`
Color palette.
- `util::WatchDog` `rex`
Watchdog object to monitor file modification.
- `TissueClass` `T`
Tissue created.

17.63.1 Detailed Description

`template<typename RealModel, typename TissueClass> struct tissue_model::TissueModel< RealModel, TissueClass >`

Base class for the `tissue_model` helper.

Definition at line 180 of file `tissue_model.h`.

17.63.2 Member Typedef Documentation

17.63.2.1 `template<typename RealModel , typename TissueClass > typedef
std::set<cell,CompareSize> tissue_model::TissueModel<
RealModel, TissueClass >::ordered_cells_t`

Ordered set of vertices.

The vertices must represent cells. They will be ordered by area.

Definition at line 231 of file tissue_model.h.

17.63.3 Constructor & Destructor Documentation

17.63.3.1 `template<typename RealModel , typename TissueClass
> tissue_model::TissueModel< RealModel, TissueClass
>::TissueModel (QObject *parent) [inline]`

Default constructor.

Initialise the watchdog, palette, tissue and aliases.

Definition at line 250 of file tissue_model.h.

References `util::WatchDog::addObject()`, `tissue_model::TissueModel< RealModel, TissueClass >::palette`, and `tissue_model::TissueModel< RealModel, TissueClass >::rex`.

```
251      : Model(parent)
252      , rex(this)
253      , palette("pal.map")
254      , T(palette, (RealModel*)this)
255      {
256      rex.addObject(&palette);
257      }
```

17.63.4 Member Function Documentation

17.63.4.1 `template<typename RealModel , typename TissueClass > const cell&
tissue_model::TissueModel< RealModel, TissueClass >::cellFromId
(int i) [inline]`

Return a reference of the cell whose id is `i`.

The id is the one used to name cells in `drawWithNames`.

Definition at line 430 of file tissue_model.h.

References `tissue_model::TissueModel< RealModel, TissueClass >::T`.

```
431      {
432      if(i == -1)
433      return cell::null;
434      return T.S.get_cell(i);
```

```
435     }
```

17.63.4.2 template<typename RealModel , typename TissueClass > void tissue_model::TissueModel< RealModel, TissueClass >::draw (Viewer * viewer) [inline, virtual]

Draw the tissue using the ModelClass::getCellColor and Model-
Class::getCellCenterColor methods.

Reimplemented from [Model](#).

Definition at line 340 of file tissue_model.h.

References [QGLViewer::drawArrow\(\)](#), [tissue_model::TissueModel< RealModel, Tis-
sueClass >::drawNeighborhood](#), [forall](#), [forall_named](#), [tissue_model::TissueModel<
RealModel, TissueClass >::getCellCenterColor\(\)](#), [tissue_model::TissueModel<
RealModel, TissueClass >::getCellColor\(\)](#), [QGLViewer::setSceneBoundingBox\(\)](#),
[tissue_model::TissueModel< RealModel, TissueClass >::T](#), [util::Vector< dim, T
>::x\(\)](#), [util::Vector< dim, T >::y\(\)](#), and [util::Vector< dim, T >::z\(\)](#).

```
341     {
342         Vec vmin(HUGE_VAL,HUGE_VAL,HUGE_VAL), vmax(-HUGE_VAL,-HUGE_VAL,-HUGE_VAL);
343         forall(const junction& v, T.W)
344         {
345             const Point3d& pos = v->pos;
346             if(pos.x() < vmin.x)
347                 vmin.x = pos.x();
348             if(pos.y() < vmin.y)
349                 vmin.y = pos.y();
350             if(pos.z() < vmin.z)
351                 vmin.z = pos.z();
352             if(pos.x() > vmax.x)
353                 vmax.x = pos.x();
354             if(pos.y() > vmax.y)
355                 vmax.y = pos.y();
356             if(pos.z() > vmax.z)
357                 vmax.z = pos.z();
358         }
359         viewer->setSceneBoundingBox(vmin, vmax);
360
361         forall_named(const cell& c, T.S, cells)
362         {
363             Color color = getCellColor(c);
364             Color center_color = getCellCenterColor(c);
365             T.drawCell(c, color, center_color);
366             glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
367             if(drawNeighborhood)
368             {
369                 int cnt = 3;
370                 double ds = 0.1 / (3+T.C.valence(c));
371                 glColor3f(1.0, 0, 0);
372                 Vec from(c->pos);
373                 forall(const cell& n, T.C.neighbors(c))
374                 {
375                     glColor3f(1.0, 1.0, 1.0);
376                     Vec to(n->pos);
377                     Vec u = to-from;
378                     u.normalize();
```

```

379         to = from + cnt*ds*u;
380         cnt++;
381         viewer->drawArrow(from, to, -1, 6);
382     }
383 }
384 }
385 }

```

17.63.4.3 `template<typename RealModel , typename TissueClass >
void tissue_model::TissueModel< RealModel, TissueClass
>::drawWithNames () [inline, virtual]`

Draw for cell selection.

Reimplemented from [Model](#).

Definition at line 411 of file tissue_model.h.

References `forall_named`, and `tissue_model::TissueModel< RealModel, TissueClass >::T`.

```

412     {
413         bool db = T.drawBorders;
414         T.drawBorders = false;
415         int i = 0;
416         forall_named(const cell& c, T.S, cells)
417         {
418             glPushName(i++);
419             T.drawCell(c, c->value);
420             glPopName();
421         }
422         T.drawBorders = db;
423     }

```

17.63.4.4 `template<typename RealModel , typename TissueClass > virtual
Color tissue_model::TissueModel< RealModel, TissueClass
>::getCellCenterColor (const cell & c) [inline, virtual]`

Color using the `value` attribute of the cell.

Definition at line 398 of file tissue_model.h.

References `tissue_model::TissueModel< RealModel, TissueClass >::T`.

Referenced by `tissue_model::TissueModel< RealModel, TissueClass >::draw()`.

```

399     {
400         return T.valueCenterColor(c->value);
401     }

```

**17.63.4.5 template<typename RealModel , typename TissueClass > virtual
Color tissue_model::TissueModel< RealModel, TissueClass
>::getCellColor (const cell & c) [inline, virtual]**

Color using the `value` attribute of the cell.

Definition at line 390 of file `tissue_model.h`.

References `tissue_model::TissueModel< RealModel, TissueClass >::T`.

Referenced by `tissue_model::TissueModel< RealModel, TissueClass >::draw()`.

```
391      {  
392          return T.valueColor(c->value);  
393      }
```

**17.63.4.6 template<typename RealModel , typename TissueClass > void
tissue_model::TissueModel< RealModel, TissueClass >::initDraw ()
[inline, virtual]**

Convenience function used if the viewer is not necessary.

Reimplemented from [Model](#).

Definition at line 315 of file `tissue_model.h`.

```
316      {  
317          glEnable (GL_NICEST);  
318          glDisable (GL_CULL_FACE);  
319      }
```

**17.63.4.7 template<typename RealModel , typename TissueClass > void
tissue_model::TissueModel< RealModel, TissueClass >::initialize ()
[inline]**

Function to override to initialize the model.

Definition at line 488 of file `tissue_model.h`.

```
488 {}
```

**17.63.4.8 template<typename RealModel , typename TissueClass >
void tissue_model::TissueModel< RealModel, TissueClass
>::modifiedFiles (const std::set< std::string > &files) [inline,
virtual]**

This function is called anytime one or more registered files are modified.

Parameters

files Registered files modified since last call to this function.

Reimplemented from [Model](#).

Definition at line 259 of file tissue_model.h.

References [tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam\(\)](#), [tissue_model::TissueModel< RealModel, TissueClass >::rex](#), and [util::WatchDog::watch\(\)](#).

```

260     {
261         if (filenames.find("view.v") != filenames.end())
262             readTissueParam();
263         rex.watch(filenames);
264     }

```

17.63.4.9 **template<typename RealModel , typename TissueClass > void tissue_model::TissueModel< RealModel, TissueClass >::postDraw () [inline, virtual]**

Convenience function used if the viewer is not necessary.

Reimplemented from [Model](#).

Definition at line 330 of file tissue_model.h.

References [tissue_model::TissueModel< RealModel, TissueClass >::T](#).

```

331     {
332         glDisable (GL_BLEND);
333         T.postDraw();
334     }

```

17.63.4.10 **template<typename RealModel , typename TissueClass > void tissue_model::TissueModel< RealModel, TissueClass >::preDraw () [inline, virtual]**

Convenience function used if the viewer is not necessary.

Reimplemented from [Model](#).

Definition at line 321 of file tissue_model.h.

References [util::Color< T >::b\(\)](#), [tissue_model::TissueModel< RealModel, TissueClass >::backColor](#), [util::Color< T >::g\(\)](#), [util::Palette::getColor\(\)](#), [tissue_model::TissueModel< RealModel, TissueClass >::palette](#), [util::Color< T >::r\(\)](#), and [tissue_model::TissueModel< RealModel, TissueClass >::T](#).

```

322     {
323         Color bg = palette.getColor(backColor);
324         glClearColor(bg.r(), bg.g(), bg.b(), 1.0);
325         T.preDraw();
326         glEnable(GL_BLEND);
327         glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
328     }

```

17.63 tissue_model::TissueModel< RealModel, TissueClass > Struct Template Reference 569

17.63.4.11 `template<typename RealModel , typename TissueClass > virtual
void tissue_model::TissueModel< RealModel, TissueClass
>::readParam (util::Parms &) [inline, virtual]`

Function to be redefined to read extra parameters.

Definition at line 269 of file tissue_model.h.

Referenced by tissue_model::TissueModel< RealModel, TissueClass
>::readTissueParam().

```
269 {}
```

17.63.4.12 `template<typename RealModel , typename TissueClass >
void tissue_model::TissueModel< RealModel, TissueClass
>::readTissueParam () [inline]`

Read the parameters for the [tissue_model](#) helper (automatically called by the helper).

Definition at line 302 of file tissue_model.h.

References tissue_model::TissueModel< RealModel, TissueClass >::backColor,
tissue_model::TissueModel< RealModel, TissueClass >::drawNeighborhood,
tissue_model::TissueModel< RealModel, TissueClass >::readParam(), and tissue_
model::TissueModel< RealModel, TissueClass >::T.

Referenced by tissue_model::TissueModel< RealModel, TissueClass
>::modifiedFiles().

```
303 {  
304     util::Parms parms("view.v");  
305     T.readViewParms(parms, "View");  
306     T.readParms(parms, "Tissue");  
307     parms("View", "BackColor", backColor);  
308     parms("View", "DrawNeighborhood", drawNeighborhood);  
309     readParam(parms);  
310 }
```

17.63.4.13 `template<typename RealModel , typename TissueClass > void
tissue_model::TissueModel< RealModel, TissueClass >::step ()
[inline, virtual]`

Default step method: do nothing.

Implements [Model](#).

Definition at line 483 of file tissue_model.h.

```
483 { }
```

17.63.4.14 **template<typename RealModel , typename TissueClass > void tissue_model::TissueModel< RealModel, TissueClass >::updateCellsArea () [inline]**

Update the area of all the cells in the tissue.

Definition at line 274 of file tissue_model.h.

References tissue_model::epsilon, forall, forall_named, tissue_model::TissueModel< RealModel, TissueClass >::T, and geometry::triangleArea().

```

275     {
276         forall_named(const cell& c, T.S, cells)
277         {
278             c->area = 0.0;
279             forall( const junction& n, T.S.neighbors(c) )
280             {
281                 // Find area
282                 const junction& m = T.S.nextTo(c, n);
283                 c->area += geometry::triangleArea(c->pos, m->pos, n->pos);
284             }
285             if(c->area <= epsilon ) // Avoid inf
286             {
287                 std::cout << "Bad area " << c->area << std::endl;
288                 c->area = 0.1;
289             }
290         }
291     }

```

17.63.5 Member Data Documentation

17.63.5.1 **template<typename RealModel , typename TissueClass > int tissue_model::TissueModel< RealModel, TissueClass >::backColor**

Color of the background.

Definition at line 241 of file tissue_model.h.

Referenced by tissue_model::TissueModel< RealModel, TissueClass >::preDraw(), and tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam().

17.63.5.2 **template<typename RealModel , typename TissueClass > bool tissue_model::TissueModel< RealModel, TissueClass >::drawNeighborhood**

Draw the neighborhood as arrows ?

Definition at line 243 of file tissue_model.h.

Referenced by tissue_model::TissueModel< RealModel, TissueClass >::draw(), and tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam().

**17.63.5.3 template<typename RealModel , typename TissueClass >
 util::Palette tissue_model::TissueModel< RealModel, TissueClass
 >::palette**

Color palette.

Definition at line 236 of file tissue_model.h.

Referenced by tissue_model::TissueModel< RealModel, TissueClass >::preDraw(), and tissue_model::TissueModel< RealModel, TissueClass >::TissueModel().

**17.63.5.4 template<typename RealModel , typename TissueClass >
 util::WatchDog tissue_model::TissueModel< RealModel, TissueClass
 >::rex**

Watchdog object to monitor file modification.

Definition at line 234 of file tissue_model.h.

Referenced by tissue_model::TissueModel< RealModel, TissueClass >::modifiedFiles(), and tissue_model::TissueModel< RealModel, TissueClass >::TissueModel().

**17.63.5.5 template<typename RealModel , typename TissueClass >
 TissueClass tissue_model::TissueModel< RealModel, TissueClass
 >::T**

Tissue created.

Definition at line 238 of file tissue_model.h.

Referenced by tissue_model::TissueModel< RealModel, TissueClass >::cellFromId(), tissue_model::TissueModel< RealModel, TissueClass >::draw(), tissue_model::TissueModel< RealModel, TissueClass >::drawWithNames(), tissue_model::TissueModel< RealModel, TissueClass >::getCellCenterColor(), tissue_model::TissueModel< RealModel, TissueClass >::getCellColor(), tissue_model::TissueModel< RealModel, TissueClass >::postDraw(), tissue_model::TissueModel< RealModel, TissueClass >::preDraw(), tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam(), and tissue_model::TissueModel< RealModel, TissueClass >::updateCellsArea().

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

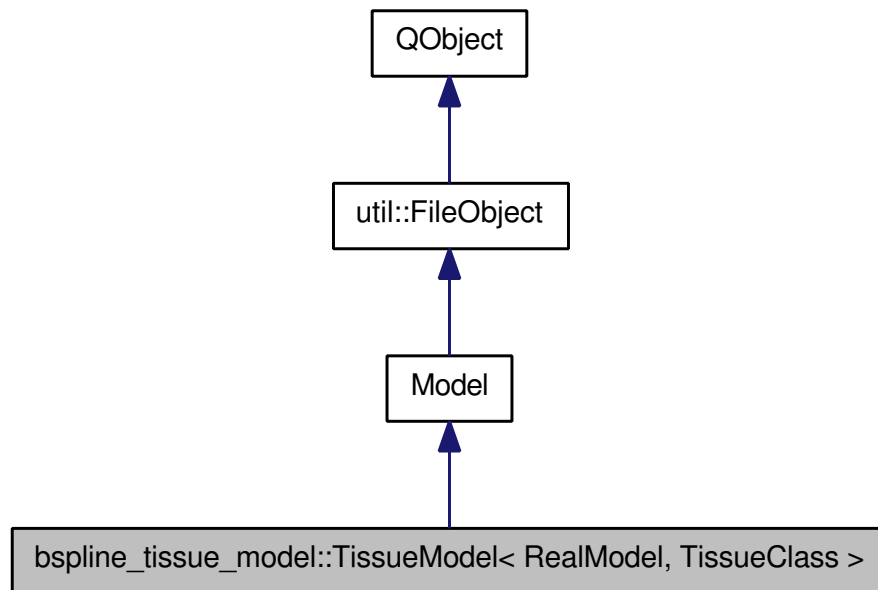
- [vvelib/tissue_model.h](#)

17.64 bspline_tissue_model::TissueModel< RealModel, TissueClass > Class Template Reference

Base class for the bspline tissue model helper.

```
#include <bspline_tissue_model.h>
```

Inheritance diagram for bspline_tissue_model::TissueModel< RealModel, TissueClass >:



Classes

- struct [CompareSize](#)
Operator class to compare the size of cells.

Public Types

- typedef `std::set< cell, CompareSize >` [ordered_cells_t](#)
Ordered set of vertices.

Public Member Functions

- const cell & [cellFromId](#) (int i)

Returns the cell of id i .

- void **draw** (**Viewer** *viewer)
This function should be redefined to draw the representation.
- void **drawWithNames** ()
Draw for cell selection.
- virtual **Color** **getCellCenterColor** (const cell &c)
Function to override to change the color used for the center of the cell.
- virtual **Color** **getCellColor** (const cell &c)
Function to override to change the color used for the cell.
- **IMPORT_COMPLEX_TYPES** (TissueClass)
- void **initDraw** ()
Convenience function used if the viewer is not necessary.
- void **initialize** ()
Method to override for user's initialization.
- void **initTissue** ()
Initialize the tissue as one single cell whose shape is the shape of the current b-spline surface.
- void **modifiedFiles** (const std::set< std::string > &filenames)
Keep track of the modification of the parameter file ("view.v"), and of all the files registered to the watchdog.
- void **postDraw** ()
Convenience function used if the viewer is not necessary.
- void **preDraw** ()
Convenience function used if the viewer is not necessary.
- virtual void **readParam** (**util::Parms** &)
Method to redefine to read extra parameters.
- void **readTissueParam** ()
Read parameters needed for the tissue and tissue growth.
- void **registerFiles** ()
Register the parameter file.
- void **SetPos** (const junction &j)
Set the position of the junction j with respect to the current time.

- void **SetPos** (const cell &c)
Set the position of the cell x with respect to the current time.
- void **step** ()
Method to override to provide the user's model.
- **TissueModel** (QObject *parent)
Current time.
- void **updateCellsArea** ()
Update the cells area using the current positions of the vertices.
- void **updatePositions** ()
Update the positions of all the vertices to reflect the new b-spline surface.

Methods required by the tissue library

- **Point3d normal** (const junction &v) const
- **Point3d normal** (const cell &v) const
- **Point3d position** (const junction &v) const
- **Point3d position** (const cell &v) const
- void **setPosition** (const junction &v, const **Point3d** &pos)
- void **setPosition** (const cell &v, const **Point3d** &pos)
- void **setPositionHint** (const junction &v, const junction &n1, const junction &n2, double r)
- void **updateFromOld** (const cell &, const cell &, const cell &, const type-name TissueClass::division_data &, TissueClass &)

Public Attributes

- int **backColor**
Initial time for the leaf description.
- int **cellInitWalls**
One surface of the leaf at a time.
- bool **drawNeighborhood**
Color of the background.
- double **dt**
Initial number of walls in the cell.
- double **growthStartTime**
Time step.
- **util::KeyFramer** leaf

Main tissue.

- util::BSplineSurface [leafs](#)

Description of the leaf growth.

- util::Palette [palette](#)

Watchdog for file modification.

- util::WatchDog [rex](#)

- TissueClass [T](#)

Palette to retrieve colors.

- double [time](#)

Draw the neighborhood as arrows ?

17.64.1 Detailed Description

template<typename RealModel, typename TissueClass> class bspline_tissue_model::TissueModel< RealModel, TissueClass >

Base class for the bspline tissue model helper.

Definition at line 216 of file bspline_tissue_model.h.

17.64.2 Member Typedef Documentation

17.64.2.1 template<typename RealModel , typename TissueClass > typedef std::set<cell,CompareSize> bspline_tissue_model::TissueModel< RealModel, TissueClass >::ordered_cells_t

Ordered set of vertices.

The vertices must represent cells. They will be ordered by area.

Definition at line 236 of file bspline_tissue_model.h.

17.64.3 Constructor & Destructor Documentation

17.64.3.1 template<typename RealModel , typename TissueClass > bspline_tissue_model::TissueModel< RealModel, TissueClass >::TissueModel (QObject *parent) [inline]

Current time.

Default constructor Creates the tissue, the aliases, the palette and the watchdog

Definition at line 258 of file bspline_tissue_model.h.

References `util::WatchDog::addObject()`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::leaf`, and `bspline_tissue_model::TissueModel< RealModel, TissueClass >::palette`.

```

259         : Model(parent)
260         , rex(this)
261         , palette("pal.map")
262         , T(palette, (RealModel*)this)
263     {
264         rex.addObject(&palette);
265         rex.addObject(&leaf);
266     }

```

17.64.4 Member Function Documentation

17.64.4.1 `template<typename RealModel , typename TissueClass > const cell& bspline_tissue_model::TissueModel< RealModel, TissueClass >::cellFromId (int i) [inline]`

Returns the cell of id *i*.

This id is the one used for OpenGL selection and is returned by `QGLViewer::selectedName()`

Definition at line 498 of file `bspline_tissue_model.h`.

References `bspline_tissue_model::TissueModel< RealModel, TissueClass >::T`.

```

499     {
500         if(i == -1)
501             return cell::null;
502         return T.S.get_cell(i);
503     }

```

17.64.4.2 `template<typename RealModel , typename TissueClass > void bspline_tissue_model::TissueModel< RealModel, TissueClass >::draw (Viewer *viewer) [inline, virtual]`

This function should be redefined to draw the representation.

Parameters

viewer [Viewer](#) object. Useful to use the capabilities of `QGLViewer` object.

Reimplemented from [Model](#).

Definition at line 426 of file `bspline_tissue_model.h`.

References `QGLViewer::drawArrow()`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::drawNeighborhood`, `forall`, `forall_named`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::getCellCenterColor()`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::getCellColor()`,

bspline_tissue_model::TissueModel< RealModel, TissueClass >::leafs,
 QGLViewer::setSceneBoundingBox(), and bspline_tissue_model::TissueModel<
 RealModel, TissueClass >::T.

```

427     {
428         forall_named(const cell& c, T.S, cells)
429         {
430             Color color = getCellColor(c);
431             Color center_color = getCellCenterColor(c);
432             T.drawCell(c, color, center_color);
433             glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
434             if(drawNeighborhood)
435             {
436                 int cnt = 3;
437                 double ds = 0.1 / (3+T.C.valence(c));
438                 glColor3f(1.0, 0, 0);
439                 Vec from(c->pos);
440                 forall(const cell& n, T.C.neighbors(c))
441                 {
442                     glColor3f(1.0, 1.0, 1.0);
443                     Vec to(n->pos);
444                     Vec u = to-from;
445                     u.normalize();
446                     to = from + cnt*ds*u;
447                     cnt++;
448                     viewer->drawArrow(from, to, -1, 6);
449                 }
450             }
451         }
452
453         double vmin[3], vmax[3];
454         leafs.BoundingBox(vmin[0], vmin[1], vmin[2], vmax[0], vmax[1], vmax[2]);
455         Vec pmin(vmin), pmax(vmax);
456         viewer->setSceneBoundingBox(pmin, pmax);
457     }

```

17.64.4.3 template<typename RealModel , typename TissueClass > void bspline_tissue_model::TissueModel< RealModel, TissueClass >::drawWithNames () [inline, virtual]

Draw for cell selection.

Reimplemented from [Model](#).

Definition at line 462 of file bspline_tissue_model.h.

References forall_named, and bspline_tissue_model::TissueModel< RealModel, Tis-
 sueClass >::T.

```

463     {
464         bool db = T.drawBorders;
465         T.drawBorders = false;
466         int i = 0;
467         forall_named(const cell& c, T.S, cells)
468         {
469             glPushName(i++);
470             T.drawCell(c, c->value);
471             glPopName();

```

```

472     }
473     T.drawBorders = db;
474 }

```

17.64.4.4 **template<typename RealModel , typename TissueClass > virtual Color bspline_tissue_model::TissueModel< RealModel, TissueClass >::getCellCenterColor (const cell & c) [inline, virtual]**

Function to override to change the color used for the center of the cell.

Definition at line 487 of file bspline_tissue_model.h.

References bspline_tissue_model::TissueModel< RealModel, TissueClass >::T.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::draw().

```

488     {
489         return T.valueCenterColor(c->value);
490     }

```

17.64.4.5 **template<typename RealModel , typename TissueClass > virtual Color bspline_tissue_model::TissueModel< RealModel, TissueClass >::getCellColor (const cell & c) [inline, virtual]**

Function to override to change the color used for the cell.

Definition at line 479 of file bspline_tissue_model.h.

References bspline_tissue_model::TissueModel< RealModel, TissueClass >::T.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::draw().

```

480     {
481         return T.valueColor(c->value);
482     }

```

17.64.4.6 **template<typename RealModel , typename TissueClass > void bspline_tissue_model::TissueModel< RealModel, TissueClass >::initDraw () [inline, virtual]**

Convenience function used if the viewer is not necessary.

Reimplemented from [Model](#).

Definition at line 405 of file bspline_tissue_model.h.

```

406     {
407         glEnable(GL_NICEST);
408         glDisable(GL_CULL_FACE);
409     }

```


**17.64.4.7 template<typename RealModel , typename TissueClass > void
bspline_tissue_model::TissueModel< RealModel, TissueClass
>::initialize () [inline]**

Method to override for user's initialization.

Definition at line 557 of file bspline_tissue_model.h.

```
557 {}
```

**17.64.4.8 template<typename RealModel , typename TissueClass > void
bspline_tissue_model::TissueModel< RealModel, TissueClass
>::initTissue () [inline]**

Initialize the tissue as one single cell whose shape is the shape of the current b-spline surface.

Definition at line 306 of file bspline_tissue_model.h.

References bspline_tissue_model::TissueModel< RealModel, TissueClass
>::cellInitWalls, bspline_tissue_model::TissueModel< RealModel, TissueClass
>::growthStartTime, bspline_tissue_model::TissueModel< RealModel, Tissue-
Class >::leaf, bspline_tissue_model::TissueModel< RealModel, TissueClass
>::leafs, bspline_tissue_model::TissueModel< RealModel, TissueClass >::SetPos(),
bspline_tissue_model::TissueModel< RealModel, TissueClass >::T, bspline_-
tissue_model::TissueModel< RealModel, TissueClass >::time, and bspline_tissue_-
model::TissueModel< RealModel, TissueClass >::updateCellsArea().

```
307     {
308         time = growthStartTime;
309         leaf.SetTime(time);
310         leafs = leaf.GetSurfaceTime();
311
312         // Find center and set cell area and length of cell walls
313         cell c;
314         c->uv[0] = c->uv[1] = .5;
315         c->value = 0.5;
316         SetPos(c);
317
318         std::vector<junction> junctions(cellInitWalls, junction(0));
319         for(int i = 0 ; i < cellInitWalls ; i++)
320         {
321             double s = (double)(cellInitWalls-i)/(double)cellInitWalls;
322             junction v;
323             v->uv = leafs.ContourInverse(s);
324             SetPos(v);
325             junctions[i] = v;
326         }
327
328         T.addCell(c, junctions);
329
330         updateCellsArea();
331     }
```

17.64.4.9 `template<typename RealModel , typename TissueClass > void
bspline_tissue_model::TissueModel< RealModel, TissueClass
>::modifiedFiles (const std::set< std::string > & filenames)
[inline, virtual]`

Keep track of the modification of the parameter file ("view.v"), and of all the files registered to the watchdog.

Reimplemented from [Model](#).

Definition at line 272 of file bspline_tissue_model.h.

References `bspline_tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam()`, and `util::WatchDog::watch()`.

```
273     {
274         if (filenames.count ("view.v"))
275             readTissueParam();
276         rex.watch (filenames);
277     }
```

17.64.4.10 `template<typename RealModel , typename TissueClass > void
bspline_tissue_model::TissueModel< RealModel, TissueClass
>::postDraw () [inline, virtual]`

Convenience function used if the viewer is not necessary.

Reimplemented from [Model](#).

Definition at line 420 of file bspline_tissue_model.h.

References `bspline_tissue_model::TissueModel< RealModel, TissueClass >::T`.

```
421     {
422         glDisable (GL_BLEND);
423         T.postDraw();
424     }
```

17.64.4.11 `template<typename RealModel , typename TissueClass > void
bspline_tissue_model::TissueModel< RealModel, TissueClass
>::preDraw () [inline, virtual]`

Convenience function used if the viewer is not necessary.

Reimplemented from [Model](#).

Definition at line 411 of file bspline_tissue_model.h.

References `util::Color< T >::b()`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::backColor`, `util::Color< T >::g()`, `util::Palette::getColor()`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::palette`, `util::Color< T >::r()`, and `bspline_tissue_model::TissueModel< RealModel, TissueClass >::T`.

```
412     {
413         Color bg = palette.getColor(backColor);
414         glClearColor(bg.r(), bg.g(), bg.b(), 1.0);
415         T.preDraw();
416         glEnable(GL_BLEND);
417         glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
418     }
```

**17.64.4.12 template<typename RealModel , typename TissueClass > virtual
void bspline_tissue_model::TissueModel< RealModel, TissueClass
>::readParam (util::Parms &) [inline, virtual]**

Method to redefine to read extra parameters.

Definition at line 282 of file bspline_tissue_model.h.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass
>::readTissueParam().

```
282 { }
```

**17.64.4.13 template<typename RealModel , typename TissueClass > void
bspline_tissue_model::TissueModel< RealModel, TissueClass
>::readTissueParam () [inline]**

Read parameters needed for the tissue and tissue growth.

Definition at line 384 of file bspline_tissue_model.h.

References bspline_tissue_model::TissueModel< RealModel, TissueClass
>::backColor, bspline_tissue_model::TissueModel< RealModel, Tissue-
Class >::cellInitWalls, util::WatchDog::changeFilename(), bspline_tissue_-
model::TissueModel< RealModel, TissueClass >::drawNeighborhood, bspline_-
tissue_model::TissueModel< RealModel, TissueClass >::dt, bspline_tissue_-
model::TissueModel< RealModel, TissueClass >::growthStartTime, bspline_-
tissue_model::TissueModel< RealModel, TissueClass >::leaf, bspline_tissue_-
model::TissueModel< RealModel, TissueClass >::readParam(), and bspline_tissue_-
model::TissueModel< RealModel, TissueClass >::T.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass
>::modifiedFiles().

```
385     {
386         util::Parms parms("view.v");
387         std::string newKeyFrameFile;
388
389         parms("Main", "dt", dt);
390         parms("Main", "KeyFrameFile", newKeyFrameFile);
391         parms("Main", "GrowthStartTime", growthStartTime);
392         parms("Main", "CellInitWalls", cellInitWalls);
393
394         rex.changeFilename(&leaf, newKeyFrameFile);
395     }
```

```

396     T.readViewParms(parms, "View");
397     parms("View", "BackColor", backColor);
398     parms("View", "DrawNeighborhood", drawNeighborhood);
399
400     T.readParms(parms, "Tissue");
401
402     readParam(parms);
403 }
```

17.64.4.14 **template<typename RealModel , typename TissueClass > void bspline_tissue_model::TissueModel< RealModel, TissueClass >::registerFiles () [inline]**

Register the parameter file.

Definition at line 376 of file bspline_tissue_model.h.

References Model::registerFile().

```

377     {
378         registerFile("view.v");
379     }
```

17.64.4.15 **template<typename RealModel , typename TissueClass > void bspline_tissue_model::TissueModel< RealModel, TissueClass >::SetPos (const junction & j) [inline]**

Set the position of the junction *j* with respect to the current time.

Definition at line 296 of file bspline_tissue_model.h.

References bspline_tissue_model::TissueModel< RealModel, TissueClass >::leafs.

```

297     {
298         j->pos = leafs.EvalN(j->uv[0], j->uv[1]);
299         j->normal = leafs.NormalEvalN(j->uv[0], j->uv[1]);
300     }
```

17.64.4.16 **template<typename RealModel , typename TissueClass > void bspline_tissue_model::TissueModel< RealModel, TissueClass >::SetPos (const cell & c) [inline]**

Set the position of the cell *x* with respect to the current time.

Definition at line 287 of file bspline_tissue_model.h.

References bspline_tissue_model::TissueModel< RealModel, TissueClass >::leafs.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::initTissue(), and bspline_tissue_model::TissueModel< RealModel, TissueClass >::updatePositions().

```
288     {  
289         c->pos = leafs.EvalN(c->uv[0], c->uv[1]);  
290         c->normal = leafs.NormalEvalN(c->uv[0], c->uv[1]);  
291     }
```

17.64.4.17 `template<typename RealModel , typename TissueClass > void
bspline_tissue_model::TissueModel< RealModel, TissueClass
>::step() [inline, virtual]`

Method to override to provide the user's model.

Implements [Model](#).

Definition at line 552 of file bspline_tissue_model.h.

```
552 { }
```

17.64.4.18 `template<typename RealModel , typename TissueClass > void
bspline_tissue_model::TissueModel< RealModel, TissueClass
>::updateCellsArea() [inline]`

Update the cells area using the current positions of the vertices.

Definition at line 354 of file bspline_tissue_model.h.

References `bspline_tissue_model::epsilon`, `forall`, `forall_named`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::T`, and `geometry::triangleArea()`.

Referenced by `bspline_tissue_model::TissueModel< RealModel, TissueClass >::initTissue()`.

```
355     {  
356         forall_named( const cell& c, T.S, cells )  
357         {  
358             c->area = 0.0;  
359             forall( const junction& n, T.S.neighbors(c) )  
360             {  
361                 // Find area  
362                 const junction& m = T.S.nextTo(c, n);  
363                 c->area += geometry::triangleArea(c->pos, m->pos, n->pos);  
364             }  
365             if(c->area <= epsilon ) // Avoid inf  
366             {  
367                 cout << "Bad area " << c->area << endl;  
368                 c->area = 0.1;  
369             }  
370         }  
371     }
```

17.64.4.19 `template<typename RealModel , typename TissueClass > void
bspline_tissue_model::TissueModel< RealModel, TissueClass
>::updatePositions() [inline]`

Update the positions of all the vertices to reflect the new b-spline surface.

Definition at line 337 of file bspline_tissue_model.h.

References forall, bspline_tissue_model::TissueModel< RealModel, TissueClass >::leaf, bspline_tissue_model::TissueModel< RealModel, TissueClass >::leafs, bspline_tissue_model::TissueModel< RealModel, TissueClass >::SetPos(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::T, and bspline_tissue_model::TissueModel< RealModel, TissueClass >::time.

```

338     {
339         leaf.SetTime(time);
340         leafs = leaf.GetSurfaceTime();
341         forall(const cell& c, T.C)
342         {
343             SetPos(c);
344         }
345         forall(const junction& j, T.W)
346         {
347             SetPos(j);
348         }
349     }

```

17.64.5 Member Data Documentation

17.64.5.1 template<typename RealModel , typename TissueClass > int bspline_tissue_model::TissueModel< RealModel, TissueClass >::backColor

Initial time for the leaf description.

Definition at line 248 of file bspline_tissue_model.h.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::preDraw(), and bspline_tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam().

17.64.5.2 template<typename RealModel , typename TissueClass > int bspline_tissue_model::TissueModel< RealModel, TissueClass >::cellInitWalls

One surface of the leaf at a time.

Definition at line 245 of file bspline_tissue_model.h.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::initTissue(), and bspline_tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam().

17.64.5.3 template<typename RealModel , typename TissueClass > bool bspline_tissue_model::TissueModel< RealModel, TissueClass >::drawNeighborhood

Color of the background.

17.64 bspline_tissue_model::TissueModel< RealModel, TissueClass > Class 585

Definition at line 249 of file bspline_tissue_model.h.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::draw(), and bspline_tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam().

17.64.5.4 template<typename RealModel , typename TissueClass > double bspline_tissue_model::TissueModel< RealModel, TissueClass >::dt

Initial number of walls in the cell.

Definition at line 246 of file bspline_tissue_model.h.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam().

17.64.5.5 template<typename RealModel , typename TissueClass > double bspline_tissue_model::TissueModel< RealModel, TissueClass >::growthStartTime

Time step.

Definition at line 247 of file bspline_tissue_model.h.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::initTissue(), and bspline_tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam().

17.64.5.6 template<typename RealModel , typename TissueClass > util::KeyFramer bspline_tissue_model::TissueModel< RealModel, TissueClass >::leaf

Main tissue.

Definition at line 242 of file bspline_tissue_model.h.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::initTissue(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::TissueModel(), and bspline_tissue_model::TissueModel< RealModel, TissueClass >::updatePositions().

17.64.5.7 template<typename RealModel , typename TissueClass > util::BSplineSurface bspline_tissue_model::TissueModel< RealModel, TissueClass >::leafs

Description of the leaf growth.

Definition at line 243 of file bspline_tissue_model.h.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::draw(), bspline_tissue_model::TissueModel< RealModel, TissueClass

>::initTissue(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::SetPos(), and bspline_tissue_model::TissueModel< RealModel, TissueClass >::updatePositions().

17.64.5.8 `template<typename RealModel , typename TissueClass >
util::Palette bspline_tissue_model::TissueModel< RealModel,
TissueClass >::palette`

Watchdog for file modification.

Definition at line 239 of file bspline_tissue_model.h.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::preDraw(), and bspline_tissue_model::TissueModel< RealModel, TissueClass >::TissueModel().

17.64.5.9 `template<typename RealModel , typename TissueClass >
TissueClass bspline_tissue_model::TissueModel< RealModel,
TissueClass >::T`

Palette to retrieve colors.

Definition at line 240 of file bspline_tissue_model.h.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::cellFromId(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::draw(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::drawWithNames(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::getCellCenterColor(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::getCellColor(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::initTissue(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::postDraw(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::preDraw(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam(), bspline_tissue_model::TissueModel< RealModel, TissueClass >::updateCellsArea(), and bspline_tissue_model::TissueModel< RealModel, TissueClass >::updatePositions().

17.64.5.10 `template<typename RealModel , typename TissueClass > double
bspline_tissue_model::TissueModel< RealModel, TissueClass
>::time`

Draw the neighborhood as arrows ?

Definition at line 251 of file bspline_tissue_model.h.

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::initTissue(), and bspline_tissue_model::TissueModel< RealModel, TissueClass >::updatePositions().

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

- [vvelib/bspline_tissue_model.h](#)

17.65 algorithms::TriangleGrowth Class Reference

Growth description using a set of triangles moving.

```
#include <algorithms/triangle_growth.h>
```

Public Types

- typedef [TriangleSurface::Position](#) **Position**

Public Member Functions

- `std::vector< Position > contour () const`
Return a description of the contour of the surface independent of the time.
- `double endTime () const`
Returns the end time.
- `const QString & errorMsg () const`
If the object is not valid, return the reason.
- `template<typename Model , typename ComplexGraph >
bool init (const ComplexGraph &cplx, std::vector< double > times, Model
*model)`
Initialize the growth from a data structure.
- `bool readOBJs (const std::string &desc_filepath)`
Initialize the growth with a description file or a directory.
- `bool readOBJs (QString desc_filepath)`
Initialize the growth with a description file or a directory.
- `void setTime (double t)`
Set the current time.
- `size_t size () const`
Returns the number of time points.
- `double startTime () const`
Returns the start time.
- `TriangleSurface surface (double time)`
Return the surface at the current time.
- `const TriangleSurface & surface () const`
Return the surface at the current time.

- double [time](#) () const
Return the current time.
- int [timePos](#) () const
Return the current position in the time scale.
- bool [valid](#) () const

Protected Types

- typedef [vvcomplex::VVComplexGraph](#)< Cell, Vertex > **CellComplex**

Protected Member Functions

- **EXPORT_COMPLEX_VERTICES** ([CellComplex](#))
- void **setSurface** ([TriangleSurface](#) &s) const
- [Position](#) **vertexPosition** (const junction &j) const

Protected Attributes

- **QString _errorMsg**
- double **_time**
- int **_time_pos**
- bool **_valid**
- std::vector< cell > **faces**
- [CellComplex](#) **S**
- std::vector< double > **times**
- [TriangleSurface](#) **TS**
- std::vector< junction > **vertices**

17.65.1 Detailed Description

Growth description using a set of triangles moving. The number of points and triangles must be constant for this class to work. Also, the different OBJ files must keep the same indices for the same points.

Definition at line 205 of file `triangle_growth.h`.

17.65.2 Member Function Documentation

17.65.2.1 `std::vector< TriangleGrowth::Position > algorithms::TriangleGrowth::contour () const`

Return a description of the contour of the surface independent of the time.

Definition at line 912 of file triangle_growth.cpp.

References `algorithms::TriangleSurface::contour()`, and `surface()`.

```
913 {
914     return surface().contour();
915 }
```

17.65.2.2 `double algorithms::TriangleGrowth::endTime () const [inline]`

Returns the end time.

Definition at line 286 of file triangle_growth.h.

```
286 { return times.back(); }
```

17.65.2.3 `const QString& algorithms::TriangleGrowth::errorMsg () const [inline]`

If the object is not valid, return the reason.

Definition at line 271 of file triangle_growth.h.

```
271 { return _errorMsg; }
```

17.65.2.4 `template<typename Model , typename ComplexGraph > bool algorithms::TriangleGrowth::init (const ComplexGraph & cplx, std::vector< double > times, Model * model) [inline]`

Initialize the growth from a data structure.

Parameters

cplx Graph complex representing the triangulated mesh

times List of key frame times, ordered from the earliest to the latest (i.e. times must be strictly increasing).

model Object able interrogated for informations about vertices and cells.

The [Model](#) class is suppose to include a few methods:

```
Point3d position(const junction& v, int keyframe) const;
```

Returns the position of the junction *v* at the specified key frame.

```
Point3d normal(const junction& v, int keyframe) const;
```

Returns the normal to the junction *v* at the specified key frame.

Definition at line 343 of file `triangle_growth.h`.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::clear()`, `forall`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insert()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertEdge()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::reference()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::valence()`.

```

344 {
345     typedef typename ComplexGraph::cell ext_cell;
346     typedef typename ComplexGraph::junction ext_junction;
347
348     //QTextStream out(stdout);
349
350     _valid = false;
351
352     std::unordered_map<ext_cell, cell> cell_convert;
353     std::unordered_map<ext_junction, junction> junction_convert;
354
355     faces.clear();
356     vertices.clear();
357     S.clear();
358
359     //out << "Init the structure " << flush;
360
361     if(_times.empty())
362         return false;
363     size_t nb_times = _times.size();
364     times = _times;
365     _time = times.front();
366     _time_pos = 0;
367     faces.resize(cplx.nb_cells(), cell(0));
368     vertices.resize(cplx.nb_junctions(), junction(0));
369
370     //out << "done" << endl;
371
372     //out << "Create the faces " << flush;
373
374     // Create faces
375     int i = 0;
376     forall(const ext_cell& ec, cplx.cells())
377     {
378         //out << '.' << flush;
379         cell c;
380         c->id = i;
381         faces[i] = c;
382         S.insert(c);
383         cell_convert[ec] = c;
384         i++;
385     }
386     //out << " OK" << endl;
387
388     //out << "Create the vertices " << flush;
389

```

```

390 // Create vertices
391 i = 0;
392 forall(const ext_junction& ej, cplx.junctions())
393 {
394     //out << '.' << flush;
395     junction j;
396     j->id = i;
397     j->pos.resize(nb_times);
398     j->normal.resize(nb_times);
399     for(size_t t = 0 ; t < nb_times ; ++t)
400     {
401         j->pos[t] = model->position(ej, t);
402         j->normal[t] = model->normal(ej, t);
403     }
404     vertices[i] = j;
405     S.insert(j);
406     junction_convert[ej] = j;
407     i++;
408 }
409 //out << " OK" << endl;
410
411 //out << "Copy faces neighborhood " << flush;
412
413 // Copy faces neighborhood
414 forall(const ext_cell& ec, cplx.cells())
415 {
416     //out << '.' << flush;
417     const cell& c = S.reference(cell_convert[ec]);
418     forall(const ext_junction& ej, cplx.neighbors(ec))
419     {
420         junction j = junction_convert[ej];
421         if(S.valence(c) < 2)
422         {
423             S.insertEdge(c, j);
424         }
425         else
426         {
427             junction pj = junction_convert[cplx.prevTo(ec, ej)];
428             S.spliceAfter(c, pj, j);
429         }
430     }
431     vvassert_msg(S.valence(c) == 3, QString("Face %1 has %2 junctions instead of 3").arg(c->id).arg(S.valence(c)));
432 }
433 //out << " OK" << endl;
434
435 //out << "Copy faces neighborhood " << flush;
436 // Copy vertices neighborhood
437 forall(const ext_junction& ej, cplx.junctions())
438 {
439     //out << '.' << flush;
440     const junction& j = S.reference(junction_convert[ej]);
441     forall(const ext_cell& ec, cplx.neighbors(ej))
442     {
443         cell c = cell_convert[ec];
444         if(S.valence(j) < 2)
445         {
446             S.insertEdge(j, c);
447         }
448         else
449         {
450             cell pc = cell_convert[cplx.prevTo(ej, ec)];

```

```

451         S.spliceAfter(j, pc, c);
452     }
453 }
454 }
455 //out << " OK" << endl;
456
457 _valid = true;
458 return true;
459 }
```

17.65.2.5 bool algorithms::TriangleGrowth::readOBJs (const std::string & desc_filepath) [inline]

Initialize the growth with a description file or a directory.

Returns

True if everything was fine. If not, the error is but in the error message of the class.

Definition at line 238 of file triangle_growth.h.

References QString::fromStdString(), and readOBJs().

Referenced by readOBJs().

```
238 { return readOBJs(QString::fromStdString(desc_filepath)); }
```

17.65.2.6 bool algorithms::TriangleGrowth::readOBJs (QString desc_filepath)

Initialize the growth with a description file or a directory.

Returns

True if everything was fine. If not, the error is but in the error message of the class.

Definition at line 313 of file triangle_growth.cpp.

References QString::arg(), QTextStream::atEnd(), QFile::close(), QFileInfo::dir(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::edge(), QDir::exists(), QFileInfo::exists(), QDir::filePath(), forall, graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iAnyIn(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iEmpty(), QString::indexOf(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iNeighbors(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insert(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertEdge(), QString::isEmpty(), QFileInfo::isFile(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iValence(), graph::VVBIGraph<

Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo(), QFile::open(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::prevTo(), QTextStream::readLine(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceBefore(), QString::split(), QDir::toNativeSeparators(), QString::truncate(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::valence().

```

314 {
315     desc_filepath = QDir::toNativeSeparators(desc_filepath);
316     _valid = false;
317     std::vector<QString> data_files;
318     std::vector<double> data_scales;
319     times.clear();
320     QFileInfo info(desc_filepath);
321     if(not (info.exists() and info.isFile()))
322     {
323         if(info.exists())
324             _errorMsg = QString(tr("Path '%1' exists but is not a file.")).arg(desc_f
ilepath);
325         else
326             _errorMsg = QString(tr("Path '%1' does not exists.")).arg(desc_filepath);
327         return false;
328     }
329     QDir main_dir = info.dir();
330     QFile desc_file(desc_filepath);
331     if(!desc_file.open(QIODevice::ReadOnly))
332     {
333         _errorMsg = QString(tr("Error while opening file '%1':\n%2")).arg(desc_file
path).arg(desc_file.errorString());
334         return false;
335     }
336     QTextStream desc(&desc_file);
337     double current_time = -HUGE_VAL;
338     size_t ln = 0;
339     while(!desc.atEnd())
340     {
341         QString line = desc.readLine().simplified();
342         int comment = line.indexOf('#');
343         if(comment != -1)
344             line.truncate(comment);
345         ln++;
346         if(!line.isEmpty())
347         {
348             QString data_file;
349             double data_time;
350             double data_scale = 1;
351             QStringList fields = line.split(" ");
352             bool ok;
353             switch(fields.size())
354             {
355                 case 3:
356                     data_scale = fields[2].toDouble(&ok);
357                     if(!ok)
358                     {
359                         _errorMsg = QString(tr("Line %1 of file '%2', the scaling field can

```



```

        not be parsed as a double.")).arg(ln).arg(desc_filepath);
360         return false;
361     }
362     if(data_scale <= 0)
363     {
364         _errorMsg = QString(tr("Line %1 of file '%2', the scaling field must
t be strictly positive.")).arg(ln).arg(desc_filepath);
365         return false;
366     }
367     case 2:
368         data_time = fields[0].toDouble(&ok);
369         if(!ok)
370         {
371             _errorMsg = QString(tr("Line %1 of file '%2', the time field cannot
be parsed as a double.")).arg(ln).arg(desc_filepath);
372             return false;
373         }
374         if(data_time <= current_time)
375         {
376             _errorMsg = QString(tr("Line %1 of file '%2', the time field is sma
lller or equal than the previous one.")).arg(ln).arg(desc_filepath);
377             desc_file.close();
378             return false;
379         }
380         data_file = fields[1];
381         if(!main_dir.exists(data_file))
382         {
383             _errorMsg = QString(tr("Line %1 of file '%2', the file '%3' does no
t exist.")).arg(ln).arg(desc_filepath).arg(data_file);
384             return false;
385         }
386         times.push_back(data_time);
387         data_files.push_back(data_file);
388         data_scales.push_back(data_scale);
389         break;
390     default:
391         _errorMsg = QString(tr("Line %1 of file '%2' is invalid.")).arg(ln).a
rg(desc_filepath);
392         return false;
393     }
394 }
395 }
396 desc_file.close();
397 vertices.clear();
398 faces.clear();
399 if(times.empty())
400     return false;
401 // Read the first OBJ file to create the cell complex
402 QString first_filepath = main_dir.filePath(data_files.front());
403 _time = times.front();
404 _time_pos = 0;
405 QFile first_file(first_filepath);
406 if(!first_file.open(QIODevice::ReadOnly))
407 {
408     _errorMsg = QString(tr("Impossible to open file '%1' for reading")).arg(fir
st_filepath);
409     times.clear();
410     return false;
411 }
412 ln = 0;
413 QTextStream first(&first_file);
414 size_t nb_timepoints = times.size();

```

```

415     size_t current_normal = 0;
416     while(!first.atEnd())
417     {
418         QString line = first.readLine().simplified();
419         ln++;
420         int comment = line.indexOf('#');
421         if(comment != -1)
422             line.truncate(comment);
423         if(!line.isEmpty())
424         {
425             QStringList fields = line.split(' ');
426             QString field_type = fields[0];
427             if(field_type == "v")
428             {
429                 if(fields.size() != 4)
430                 {
431                     _errorMsg = QString(tr("Error in OBJ file '%1', line %2: vertex posit
ion must 3D")).arg(first_filepath).arg(ln);
432                     return false;
433                 }
434                 junction j;
435                 double x,y,z;
436                 bool ok;
437                 x = fields[1].toDouble(&ok);
438                 if(!ok)
439                 {
440                     _errorMsg = QString(tr("Error in OBJ file '%1', line %2: x coordinate
is not convertible to a double number")).arg(first_filepath).arg(ln);
441                     return false;
442                 }
443                 y = fields[2].toDouble(&ok);
444                 if(!ok)
445                 {
446                     _errorMsg = QString(tr("Error in OBJ file '%1', line %2: y coordinate
is not convertible to a double number")).arg(first_filepath).arg(ln);
447                     return false;
448                 }
449                 z = fields[3].toDouble(&ok);
450                 if(!ok)
451                 {
452                     _errorMsg = QString(tr("Error in OBJ file '%1', line %2: z coordinate
is not convertible to a double number")).arg(first_filepath).arg(ln);
453                     return false;
454                 }
455                 j->pos.resize(nb_timepoints, Point3d(x,y,z));
456                 j->id = (int)vertices.size();
457                 vertices.push_back(j);
458                 S.insert(j);
459             }
460             else if(field_type == "vn")
461             {
462                 if(current_normal >= vertices.size())
463                 {
464                     _errorMsg = QString(tr("Error in OBJ file '%1', line %2: the vertex n
ormal is specified before the vertex")).arg(first_filepath).arg(ln);
465                     return false;
466                 }
467                 if(fields.size() != 4)
468                 {
469                     _errorMsg = QString(tr("Error in OBJ file '%1', line %2: vertex norma
l must 3D")).arg(first_filepath).arg(ln);
470                     return false;

```

```
471         }
472         double x,y,z;
473         bool ok;
474         x = fields[1].toDouble(&ok);
475         if(!ok)
476         {
477             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: x coordinate
is not convertible to a double number")).arg(first_filepath).arg(ln);
478             return false;
479         }
480         y = fields[2].toDouble(&ok);
481         if(!ok)
482         {
483             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: y coordinate
is not convertible to a double number")).arg(first_filepath).arg(ln);
484             return false;
485         }
486         z = fields[3].toDouble(&ok);
487         if(!ok)
488         {
489             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: z coordinate
is not convertible to a double number")).arg(first_filepath).arg(ln);
490             return false;
491         }
492         const junction& j = vertices[current_normal];
493         j->normal.resize(nb_timepoints, Point3d(x,y,z));
494     }
495     else if(field_type == "f")
496     {
497         if(fields.size() != 4)
498         {
499             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: face must be
a triangle")).arg(first_filepath).arg(ln);
500             return false;
501         }
502         int i1, i2, i3;
503         bool ok;
504         i1 = fields[1].toInt(&ok);
505         if(!ok)
506         {
507             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: first vertex
index is not an integer")).arg(first_filepath).arg(ln);
508             return false;
509         }
510         if(i1 > (int)vertices.size())
511         {
512             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: first vertex
index doesn't reference an existing vertex")).arg(first_filepath).arg(ln);
513             return false;
514         }
515         i2 = fields[2].toInt(&ok);
516         if(!ok)
517         {
518             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: second verte
x index is not an integer")).arg(first_filepath).arg(ln);
519             return false;
520         }
521         if(i2 > (int)vertices.size())
522         {
523             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: second verte
x index doesn't reference an existing vertex")).arg(first_filepath).arg(ln);
524             return false;
```

```

525         }
526         i3 = fields[3].toInt(&ok);
527         if(!ok)
528         {
529             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: third vertex
index is not an integer")).arg(first_filepath).arg(ln);
530             return false;
531         }
532         if(i3 > (int)vertices.size())
533         {
534             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: third vertex
index doesn't reference an existing vertex")).arg(first_filepath).arg(ln);
535             return false;
536         }
537         cell f;
538         f->id = (int)faces.size();
539         S.insert(f);
540         junction v1 = vertices[i1-1];
541         junction v2 = vertices[i2-1];
542         junction v3 = vertices[i3-1];
543         S.insertEdge(f, v1);
544         S.insertEdge(f, v2);
545         S.spliceAfter(f, v2, v3);
546         faces.push_back(f);
547     }
548 }
549 }
550 first_file.close();
551 // Read the other positions
552 for(size_t i = 1 ; i < times.size() ; ++i)
553 {
554     QString filepath = main_dir.filePath(data_files[i]);
555     QFile file(filepath);
556     if(!file.open(QIODevice::ReadOnly))
557     {
558         _errorMsg = QString(tr("Impossible to open file '%1' for reading")).arg(f
ilepath);
559         times.clear();
560         return false;
561     }
562     ln = 0;
563     QTextStream fstream(&file);
564     current_normal = 0;
565     size_t current_vertex = 0;
566     while(!fstream.atEnd())
567     {
568         QString line = fstream.readLine().simplified();
569         ln++;
570         int comment = line.indexOf('#');
571         if(comment != -1)
572             line.truncate(comment);
573         if(!line.isEmpty())
574         {
575             QStringList fields = line.split(' ');
576             QString field_type = fields[0];
577             if(field_type == "v")
578             {
579                 if(fields.size() != 4)
580                 {
581                     _errorMsg = QString(tr("Error in OBJ file '%1', line %2: vertex pos
ition must 3D")).arg(filepath).arg(ln);
582                     return false;

```

```

583         }
584         double x,y,z;
585         bool ok;
586         x = fields[1].toDouble(&ok);
587         if(!ok)
588         {
589             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: x coordina
te is not convertible to a double number")).arg(filepath).arg(ln);
590             return false;
591         }
592         y = fields[2].toDouble(&ok);
593         if(!ok)
594         {
595             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: y coordina
te is not convertible to a double number")).arg(filepath).arg(ln);
596             return false;
597         }
598         z = fields[3].toDouble(&ok);
599         if(!ok)
600         {
601             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: z coordina
te is not convertible to a double number")).arg(filepath).arg(ln);
602             return false;
603         }
604         vertices[current_vertex++]>pos[i] = Point3d(x,y,z);
605     }
606     else if(field_type == "vn")
607     {
608         if(fields.size() != 4)
609         {
610             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: vertex nor
mal must 3D")).arg(filepath).arg(ln);
611             return false;
612         }
613         double x,y,z;
614         bool ok;
615         x = fields[1].toDouble(&ok);
616         if(!ok)
617         {
618             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: x coordina
te is not convertible to a double number")).arg(filepath).arg(ln);
619             return false;
620         }
621         y = fields[2].toDouble(&ok);
622         if(!ok)
623         {
624             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: y coordina
te is not convertible to a double number")).arg(filepath).arg(ln);
625             return false;
626         }
627         z = fields[3].toDouble(&ok);
628         if(!ok)
629         {
630             _errorMsg = QString(tr("Error in OBJ file '%1', line %2: z coordina
te is not convertible to a double number")).arg(filepath).arg(ln);
631             return false;
632         }
633         vertices[current_normal++]>normal[i] = Point3d(x,y,z);
634     }
635 }
636 }
637 }

```

```

638 // Reconstruct the graphs
639 forall(const junction& j, S.junctions())
640 {
641     if(!S.iEmpty(j))
642     {
643         cell left = S.iAnyIn(j);
644         cell right = left;
645         S.insertEdge(j, left);
646         while(S.valence(j) != S.iValence(j))
647         {
648             forall(const cell& c, S.iNeighbors(j))
649             {
650                 if(S.edge(j,c))
651                     continue;
652                 //          const junction& right_j = S.prevTo(right, j);
653                 //          const junction& left_j = S.nextTo(left, j);
654                 if(S.edge(c, S.prevTo(right, j)))
655                 {
656                     S.spliceAfter(j, right, c);
657                     right = c;
658                 }
659                 else if(S.edge(c, S.nextTo(left, j)))
660                 {
661                     S.spliceBefore(j, left, c);
662                     left = c;
663                 }
664             }
665         }
666     }
667 }
668 // For all vertices that do not have their normals ... compute them !
669 forall(const junction& j, S.junctions())
670 {
671     if(j->normal.empty())
672     {
673         std::vector<Point3d> &normal = j->normal;
674         normal.resize(nb_timepoints);
675         const std::vector<Point3d>& jpos = j->pos;
676         forall(const cell& c, S.neighbors(j))
677         {
678             const junction& jp = S.prevTo(c, j);
679             const junction& jn = S.nextTo(c, j);
680             const std::vector<Point3d>& jppos = jp->pos;
681             const std::vector<Point3d>& jnpos = jn->pos;
682             for(size_t i = 0 ; i < nb_timepoints ; ++i)
683             {
684                 normal[i] += (jnpos[i]-jpos[i]) ^ (jppos[i]-jpos[i]);
685             }
686         }
687         for(size_t i = 0 ; i < nb_timepoints ; ++i)
688         {
689             normal[i].normalize();
690         }
691     }
692 }
693 _valid = true;
694 return true;
695 }

```

17.65.2.7 void algorithms::TriangleGrowth::setTime (double *t*)

Set the current time.

Warning

If you try to set a time less than the start time or greater than the end time, the time will be clamped to the start or end value.

Definition at line 702 of file triangle_growth.cpp.

```
703 {
704     if(!_valid)
705         return;
706     if(t < times.front())
707         t = times.front();
708     else if(t > times.back())
709         t = times.back();
710     _time = t;
711     for(size_t i = 0 ; i < times.size()-1 ; ++i)
712     {
713         if(times[i+1] > t)
714         {
715             _time_pos = (int)i;
716             return;
717         }
718     }
719     _time_pos = (int)times.size()-1;
720 }
```

17.65.2.8 size_t algorithms::TriangleGrowth::size () const

Returns the number of time points.

Definition at line 697 of file triangle_growth.cpp.

```
698 {
699     return times.size();
700 }
```

17.65.2.9 double algorithms::TriangleGrowth::startTime () const [inline]

Returns the start time.

Definition at line 281 of file triangle_growth.h.

```
281 { return times.front(); }
```

17.65.2.10 TriangleSurface algorithms::TriangleGrowth::surface (double *time*)

Return the surface at the current time.

Definition at line 733 of file triangle_growth.cpp.

References algorithms::TriangleSurface::time.

```

734 {
735     TriangleSurface ts;
736     if(time < times.front())
737         time = times.front();
738     else if(time > times.back())
739         time = times.back();
740     ts.time = time;
741     ts.time_index = (int)times.size()-1;
742     for(size_t i = 0 ; i < times.size()-1 ; ++i)
743     {
744         if(times[i+1] > time)
745         {
746             ts.time_index = (int)i;
747             break;
748         }
749     }
750     setSurface(ts);
751     return ts;
752 }
```

17.65.2.11 const TriangleSurface & algorithms::TriangleGrowth::surface () const

Return the surface at the current time.

Definition at line 722 of file triangle_growth.cpp.

References algorithms::TriangleSurface::time.

Referenced by contour().

```

723 {
724     if(TS.time != _time)
725     {
726         TS.time = _time;
727         TS.time_index = _time_pos;
728         setSurface(TS);
729     }
730     return TS;
731 }
```

17.65.2.12 double algorithms::TriangleGrowth::time () const [inline]

Return the current time.

Definition at line 299 of file triangle_growth.h.

```

299 { return _time; }
```


17.65.2.13 int algorithms::TriangleGrowth::timePos () const [inline]

Return the current position in the time scale.

Definition at line 304 of file triangle_growth.h.

```
304 { return _time_pos; }
```

17.65.2.14 bool algorithms::TriangleGrowth::valid () const [inline]**Returns**

True if the growth is correctly initialized

Definition at line 266 of file triangle_growth.h.

```
266 { return _valid; }
```

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

- [vvelib/algorithms/triangle_growth.h](#)
- [vvelib/algorithms/triangle_growth.cpp](#)

17.66 algorithms::TriangleSurface Class Reference

Class representing a triangulated surface.

```
#include <algorithms/triangle_growth.h>
```

Classes

- struct [Cell](#)
Type of a cell, i.e.
- struct [Position](#)
Structure describing the position of a point on a triangulated surface, relatively to this surface.
- struct [Vertex](#)
Type of a vertex, i.e.

Public Types

- typedef [vvcomplex::VVComplexGraph](#)< [Cell](#), [Vertex](#) > [CellComplex](#)
Type of the complex graph holding the triangle structure.

Public Member Functions

- Point3d [center3d](#) (int face) const
Returns the coordinates of the center of the face.
- [Position](#) [centerPosition](#) (int face) const
Returns the position of the center of the face.
- void [clear](#) ()
Reset the structure .
- std::vector< [Position](#) > [contour](#) () const
Return a description of the contour of the surface independent of the time.
- **EXPORT_COMPLEX_VERTICES** ([CellComplex](#))
- Point3d [normal](#) (const [Position](#) &pos) const
Get the normal to the surface at position pos.
- Point3d [point3d](#) (const [Position](#) &pos) const
Get the 3d position from the position relative to the triangular surface.

- [Position position](#) (const [Position](#) &start, const Point3d &pos) const
Same as position(const Point3d&), but specifies a starting point.
- [Position position](#) (const Point3d &pos) const
Give the position, relatively to the surface, of the point closest to pos.
- Point3d [smoothNormal](#) (const [Position](#) &pos) const
Get the "smoothed" normal.
- [Position vector](#) (const [Position](#) &ref, const Point3d &pos) const
Creates a vector attached to the surface.
- [Position vector](#) (const Point3d &ref, const Point3d &pos) const
Creates a vector attached to the surface.
- Point3d [vector3d](#) (const [Position](#) &pos) const
Return the vector represented by pos.

Public Attributes

- std::vector< cell > [faces](#)
List of faces.
- Point3d **pmax**
- Point3d [pmin](#)
Bounding box of the surface.
- [CellComplex S](#)
Complex graph holding the triangle structure.
- double [time](#)
Time of this triangle surface.
- int **time_index**
- std::vector< junction > [vertices](#)
List of vertices.

Protected Types

- typedef std::unordered_set< cell > **cell_set_t**

Protected Member Functions

- **Position** **find_position** (const cell &c, const Point3d &pos, cell_set_t &visited) const
- std::vector< **Position** > **nextContourVertex** (const junction &first, const cell &c, const junction &j, std::vector< **Position** > acc) const
- **Position** **vertexPosition** (const junction &j) const

17.66.1 Detailed Description

Class representing a triangulated surface.

Definition at line 31 of file triangle_growth.h.

17.66.2 Member Typedef Documentation

17.66.2.1 typedef vvcomplex::VVComplexGraph<Cell, Vertex> algorithms::TriangleSurface::CellComplex

Type of the complex graph holding the triangle structure.

Definition at line 59 of file triangle_growth.h.

17.66.3 Member Function Documentation

17.66.3.1 Point3d algorithms::TriangleSurface::center3d (int *face*) const

Returns the coordinates of the center of the face.

Effectively, this will be the points of barycentric coordinates (1,1,1)/3 on the face.

Definition at line 99 of file triangle_growth.cpp.

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::anyIn(), faces, graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::reference(), and S.

```

100  {
101      const cell& f = S.reference(faces[face]);
102      const junction& j1 = S.anyIn(f);
103      const junction& j2 = S.nextTo(f, j1);
104      const junction& j3 = S.nextTo(f, j2);
105      return (j1->pos + j2->pos + j3->pos)/3;
106  }
```

17.66.3.2 TriangleSurface::Position algorithms::TriangleSurface::centerPosition (int face) const

Returns the position of the center of the face.

Effectively, this will be the points of barycentric coordinates $(1,1,1)/3$ on the face.

Definition at line 90 of file triangle_growth.cpp.

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::anyIn(), faces, graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::reference(), and S.

```

91  {
92      const cell& f = S.reference(faces[face]);
93      const junction& j1 = S.anyIn(f);
94      const junction& j2 = S.nextTo(f, j1);
95      const junction& j3 = S.nextTo(f, j2);
96      return Position(Point3d(1,1,1)/3, j1->id, j2->id, j3->id, face);
97  }
```

17.66.3.3 void algorithms::TriangleSurface::clear ()

Reset the structure .

..

Definition at line 126 of file triangle_growth.cpp.

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::clear(), faces, S, time, and vertices.

```

127  {
128      S.clear();
129      vertices.clear();
130      faces.clear();
131      time = -HUGE_VAL;
132      time_index = -1;
133  }
```

17.66.3.4 std::vector< TriangleSurface::Position > algorithms::TriangleSurface::contour () const

Return a description of the contour of the surface independent of the time.

Definition at line 917 of file triangle_growth.cpp.

References faces, and forall.

Referenced by algorithms::TriangleGrowth::contour().

```

918 {
919     std::vector<Position> result;
920     switch(faces.size())
921     {
922     case 0:
923         return result; // No contour !
924     case 1:
925         {
926             const cell& c = S.any_cell();
927             const junction& j1 = S.anyIn(c);
928             const junction& j2 = S.nextTo(c, j1);
929             const junction& j3 = S.nextTo(c, j2);
930             result.push_back(vertexPosition(j1));
931             result.push_back(vertexPosition(j2));
932             result.push_back(vertexPosition(j3));
933         }
934         break;
935     case 2:
936         {
937             const cell& c1 = S.reference(faces[0]);
938             const cell& c2 = S.reference(faces[1]);
939             forall(const junction& j, S.neighbors(c1))
940             {
941                 if(S.valence(j) == 1)
942                 {
943                     result.push_back(vertexPosition(S.prevTo(c1, j)));
944                     result.push_back(vertexPosition(j));
945                     result.push_back(vertexPosition(S.nextTo(c1, j)));
946                     break;
947                 }
948             }
949             forall(const junction& j, S.neighbors(c2))
950             {
951                 if(S.valence(j) == 1)
952                 {
953                     result.push_back(vertexPosition(j));
954                     break;
955                 }
956             }
957         }
958         break;
959     default:
960         // First, find one point of the contour
961         forall(const junction& j, S.junctions())
962         {
963             if(S.border(j))
964             {
965                 result.push_back(vertexPosition(j));
966                 forall(const cell& c, S.neighbors(j))
967                 {
968                     const junction& jn = S.nextTo(c, j);
969                     int nb_common_cells = 0;
970                     forall(const cell& cn, S.neighbors(jn))
971                     {
972                         if(S.edge(j, cn))
973                             nb_common_cells++;
974                     }
975                     if(nb_common_cells == 1)
976                     {
977                         result.push_back(vertexPosition(jn));
978                         return nextContourVertex(j, c, jn, result);
979                     }

```

```

980         }
981         return result;
982     }
983 }
984 }
985 return result; // Should not be reached!
986 }
```

17.66.3.5 Point3d algorithms::TriangleSurface::normal (const Position & pos) const

Get the normal to the surface at position `pos`.

Definition at line 70 of file `triangle_growth.cpp`.

References `faces`.

```

71 {
72     return faces[pos.face]->normal;
73 }
```

17.66.3.6 Point3d algorithms::TriangleSurface::point3d (const Position & pos) const

Get the 3d position from the position relative to the triangular surface.

Definition at line 62 of file `triangle_growth.cpp`.

References `geometry::barycentricToCartesian()`, and `vertices`.

Referenced by `vector3d()`.

```

63 {
64     const Point3d& p1 = vertices[pos.v1]->pos;
65     const Point3d& p2 = vertices[pos.v2]->pos;
66     const Point3d& p3 = vertices[pos.v3]->pos;
67     return geometry::barycentricToCartesian(pos.barycentric, p1, p2, p3);
68 }
```

17.66.3.7 TriangleSurface::Position algorithms::TriangleSurface::position (const Position & start, const Point3d & pos) const

Same as `position(const Point3d&)`, but specifies a starting point.

The position will be searched, starting from the `start` position and going from triangle to triangle. The first traversed triangle in which `pos` can be projected will be used as reference.

Note

This method works only for surfaces regular enough, at least from `start` to `pos`.

Definition at line 292 of file triangle_growth.cpp.

References faces, and position().

```

293     {
294         if(start)
295         {
296             cell_set_t visited;
297             const cell& c = faces[start.face];
298             visited.insert(c);
299             return find_position(c, pos, visited);
300         }
301         else
302             return position(pos);
303     }

```

17.66.3.8 TriangleSurface::Position algorithms::TriangleSurface::position (const Point3d & pos) const

Give the position, relatively to the surface, of the point closest to pos.

Definition at line 135 of file triangle_growth.cpp.

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::anyIn(), geometry::cartesianToBarycentric(), forall, graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo(), geometry::projectPointOnTriangle(), S, and vvassert.

Referenced by position(), and vector().

```

136     {
137         Position result;
138         double best_dist = HUGE_VAL;
139         cell best_cell(0);
140         Point3d best_pos;
141         OUT("New matching for point " << pos);
142         forall(const cell& c, S.cells())
143         {
144             const junction& v1 = S.anyIn(c);
145             const junction& v2 = S.nextTo(c, v1);
146             const junction& v3 = S.nextTo(c, v2);
147             const Point3d& p1 = v1->pos;
148             const Point3d& p2 = v2->pos;
149             const Point3d& p3 = v3->pos;
150             // if(util::normsq(pos-p1) > max_sq_dist and
151             //     util::normsq(pos-p2) > max_sq_dist and
152             //     util::normsq(pos-p3) > max_sq_dist)
153             //     continue;
154             double s;
155             Point3d u = geometry::projectPointOnTriangle(pos, p1, p2, p3, s);
156             if(best_dist > s)
157             {
158                 OUT("Improved distance to: " << s << " for cell " << c.id());
159                 best_dist = s;
160                 best_cell = c;
161                 best_pos = u;
162             }
163         }
164     }

```



```

163     }
164     OUT("Best distance = " << best_dist);
165     // Now, find the barycentric coordinates
166     vvassert(best_cell);
167     if(best_cell)
168     {
169         const junction& v1 = S.anyIn(best_cell);
170         const junction& v2 = S.nextTo(best_cell, v1);
171         const junction& v3 = S.nextTo(best_cell, v2);
172         const Point3d& p1 = v1->pos;
173         const Point3d& p2 = v2->pos;
174         const Point3d& p3 = v3->pos;
175         result.barycentric = geometry::cartesianToBarycentric(best_pos, p1, p2, p3)
176     };
177     result.v1 = v1->id;
178     result.v2 = v2->id;
179     result.v3 = v3->id;
180     result.face = best_cell->id;
181     return result;
182 }

```

17.66.3.9 Point3d algorithms::TriangleSurface::smoothNormal (const Position & pos) const

Get the "smoothed" normal.

To each vertex is associated a normal, mostly used for visualization. This function will interpolate between these specified normals rather than using the actual normal of the triangle.

Definition at line 75 of file triangle_growth.cpp.

References geometry::barycentricToCartesian(), and vertices.

```

76 {
77     const Point3d& p1 = vertices[pos.v1]->normal;
78     const Point3d& p2 = vertices[pos.v2]->normal;
79     const Point3d& p3 = vertices[pos.v3]->normal;
80     return geometry::barycentricToCartesian(pos.barycentric, p1, p2, p3);
81 }

```

17.66.3.10 TriangleSurface::Position algorithms::TriangleSurface::vector (const Position & ref, const Point3d & pos) const

Creates a vector attached to the surface.

Definition at line 108 of file triangle_growth.cpp.

References geometry::cartesianToBarycentric(), and vertices.

```

109 {
110     const Point3d& p1 = vertices[ref.v1]->pos;
111     const Point3d& p2 = vertices[ref.v2]->pos;
112     const Point3d& p3 = vertices[ref.v3]->pos;

```

```

113     const Point3d& pt = p1 + pos;
114     const Point3d& vec = geometry::cartesianToBarycentric(pt, p1, p2, p3);
115     Position result = ref;
116     result.barycentric = vec;
117     return result;
118 }

```

17.66.3.11 TriangleSurface::Position algorithms::TriangleSurface::vector (const Point3d & *ref*, const Point3d & *pos*) const

Creates a vector attached to the surface.

Definition at line 120 of file triangle_growth.cpp.

References position().

```

121 {
122     const Position& ref_p = position(ref);
123     return vector(ref_p, pos);
124 }

```

17.66.3.12 Point3d algorithms::TriangleSurface::vector3d (const Position & *pos*) const

Return the vector represented by *pos*.

The vector is such that its orientation and size are deformed with the surface. Also, the vector will always be in the plane of the triangle it is defined on.

Definition at line 83 of file triangle_growth.cpp.

References point3d(), and vertices.

```

84 {
85     const Point3d& p = point3d(pos);
86     const Point3d& ref = vertices[pos.v1]->pos;
87     return p-ref;
88 }

```

17.66.4 Member Data Documentation

17.66.4.1 std::vector<cell> algorithms::TriangleSurface::faces

List of faces.

Definition at line 186 of file triangle_growth.h.

Referenced by center3d(), centerPosition(), clear(), contour(), normal(), and position().

17.66.4.2 Point3d algorithms::TriangleSurface::pmin

Bounding box of the surface.

Definition at line 189 of file triangle_growth.h.

17.66.4.3 CellComplex algorithms::TriangleSurface::S

Complex graph holding the triangle structure.

Definition at line 63 of file triangle_growth.h.

Referenced by center3d(), centerPosition(), clear(), and position().

17.66.4.4 double algorithms::TriangleSurface::time

Time of this triangle surface.

Definition at line 180 of file triangle_growth.h.

Referenced by clear(), and algorithms::TriangleGrowth::surface().

17.66.4.5 std::vector<junction> algorithms::TriangleSurface::vertices

List of vertices.

Definition at line 184 of file triangle_growth.h.

Referenced by clear(), point3d(), smoothNormal(), vector(), and vector3d().

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

- vvelib/algorithms/[triangle_growth.h](#)
- vvelib/algorithms/triangle_growth.cpp

17.67 storage::TypeTraits< T > Struct Template Reference

Defines main traits for types directly handled.

```
#include <types.h>
```

Static Public Attributes

- static const TYPES `id` = T_INVALID
id of the type
- static const QString `name` = "invalid"
Name of the type.
- static const NUMBER_CLASS `type` = NOT_A_NUMBER
Class of number.

17.67.1 Detailed Description

```
template<typename T> struct storage::TypeTraits< T >
```

Defines main traits for types directly handled.

Definition at line 221 of file types.h.

17.67.2 Member Data Documentation

17.67.2.1 `template<typename T > const TYPES storage::TypeTraits< T >::id = T_INVALID [inline, static]`

id of the type

Definition at line 230 of file types.h.

17.67.2.2 `template<typename T > const QString storage::TypeTraits< T >::name = "invalid" [inline, static]`

Name of the type.

The name of the type is a valid C identifier

Definition at line 228 of file types.h.

17.67.2.3 `template<typename T > const NUMBER_CLASS
storage::TypeTraits< T >::type = NOT_A_NUMBER [inline,
static]`

Class of number.

Definition at line 234 of file types.h.

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

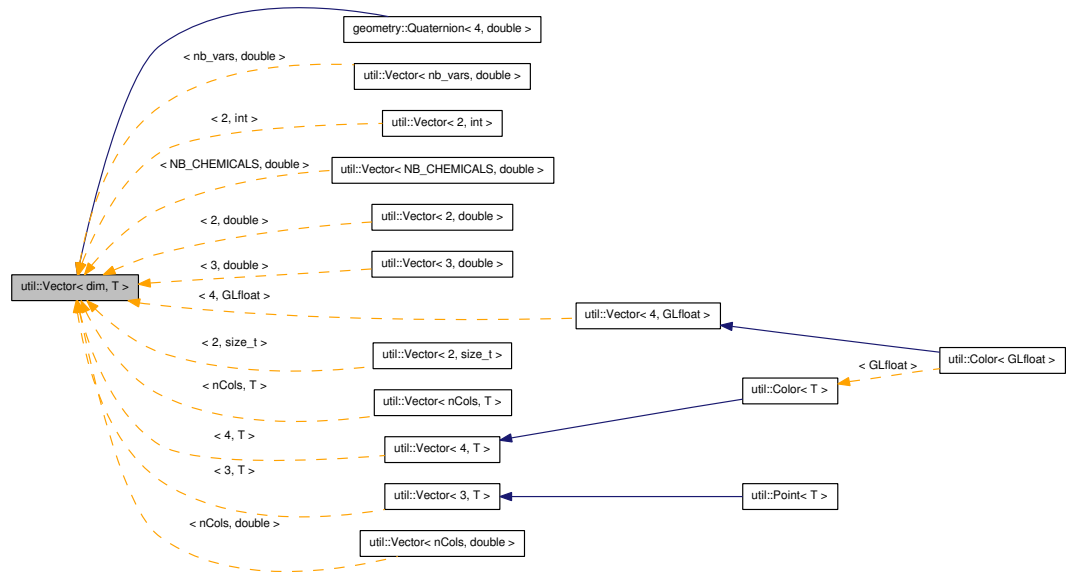
- [vvelib/storage/types.h](#)

17.68 util::Vector< dim, T > Class Template Reference

[Vector](#) class supporting all classic classic vector operations.

```
#include <util/vector.h>
```

Inheritance diagram for util::Vector< dim, T >:



Public Types

- `typedef const T * const_iterator`
- `typedef const T * const_pointer_type`
- `typedef const T & const_reference_type`
- `typedef T * iterator`
- `typedef T * pointer_type`
- `typedef T & reference_type`
- `typedef T value_type`

Public Member Functions

- `const_iterator begin ()` const
Stl-iteration constant begin.
- `iterator begin ()`
STL-iteration begin.
- `const T * c_data ()` const

Returns a constant raw pointer on the data.

- **Vector cross** (const **Vector** &other) const
Compute the cross product as `this x other`.
- **T * data** ()
Returns a raw pointer on the data.
- **const_iterator end** () const
Stl-iteration constant end.
- **iterator end** ()
STL-iteration end.
- const T & **i** () const
Short access to the first element.
- T & **i** ()
Short access to the first element.
- void **i** (const T &v)
Short access to the first element.
- bool **iszero** (void)
- const T & **j** () const
Short access to the second element.
- T & **j** ()
Short access to the second element.
- void **j** (const T &v)
Short access to the second element.
- const T & **k** () const
Short access to the third element.
- T & **k** ()
Short access to the third element.
- void **k** (const T &v)
Short access to the third element.
- const T & **l** () const
Short access to the fourth element.
- T & **l** ()

Short access to the fourth element.

- `void l (const T &v)`
Short access to the fourth element.
- `Vector mult (const Vector &vec) const`
Element-wise multiplication.
- `T norm () const`
Euclidean norm of the vector.
- `Vector & normalize (void)`
Normalize the vector.
- `Vector normalized (void) const`
Returns a normalized version of the vector.
- `T normsq () const`
Square of the Euclidean norm of the vector.
- `bool operator!= (const Vector &vec) const`
Element-wise inequality.
- `T operator* (const Vector &vec) const`
Dot product.
- `Vector operator* (const T &scalar) const`
Multiplication by a scalar.
- `template<typename T1 >
Vector & operator*= (const T1 &scalar)`
In-place multiplication by a scalar.
- `Vector & operator*= (const T &scalar)`
In-place multiplication by a scalar.
- `Vector operator+ (const Vector &vec) const`
Vector addition.
- `Vector & operator+= (const Vector &vec)`
In-place vector addition.
- `Vector operator- (const Vector &vec) const`
Vector subtraction.
- `Vector operator- (void) const`

Vector negation.

- **Vector** & **operator-=** (const **Vector** &vec)
In-place vector subtraction.
- **Vector** **operator/** (const **Vector** &vec) const
Element-wise division.
- **Vector** **operator/** (const T &scalar) const
Division by a scalar.
- template<typename T1 >
Vector & **operator/=** (const T1 &scalar)
In-place division by a scalar.
- **Vector** & **operator/=** (const T &scalar)
In-place division by a scalar.
- **Vector** & **operator/=** (const **Vector** &vec)
In-place element-wise division by a scalar.
- bool **operator<** (const **Vector** &other) const
Comparison operator.
- bool **operator<=** (const **Vector** &other) const
Comparison operator.
- **Vector** & **operator=** (const T &value)
Set all the elements to value.
- **Vector** & **operator=** (const **Vector** &vec)
Vector copy.
- bool **operator==** (const **Vector** &vec) const
Element-wise equality.
- bool **operator>** (const **Vector** &other) const
Comparison operator.
- bool **operator>=** (const **Vector** &other) const
Comparison operator.
- T **operator[]** (size_t idx) const
Access to the element idx.
- T & **operator[]** (size_t idx)

Access to the element `idx`.

- `Vector< 2, T > projectXY` (void)
Extract the two first elements of the vector.
- `void set` (const T &x, const T &y, const T &z, const T &t)
Set the values of a 4-D vector.
- `void set` (const T &x, const T &y, const T &z)
Set the values of a 3-D vector.
- `void set` (const T &x, const T &y)
Set the values of a 2-D vector.
- `void set` (const T &x)
Set the values of a 1-D vector.
- `const T & t` () const
Short access to the fourth element.
- `T & t` ()
Short access to the fourth element.
- `void t` (const T &v)
Short access to the fourth element.
- `Vector` (const T &x, const T &y, const T &z, const T &t)
Initialize a 4D vector.
- `Vector` (const T &x, const T &y, const T &z)
Initialize a 3D vector.
- `Vector` (const T &x, const T &y)
Initialize a 2D vector.
- `Vector` (const T &x=T())
Initialize a vector with all values to x .
- `template<class Vec >`
`Vector` (const **Vec** &el)
Initialize a vector from any object behaving like an array.
- `template<size_t d1, class T1 >`
`Vector` (const `Vector`< d1, T1 > &vec)
Copy another vector with different number of elements.

- **Vector** (const **Vector** &vec)
Copy another vector.
- const T & **x** () const
Short access to the first element.
- T & **x** ()
Short access to the first element.
- void **x** (const T &v)
Short access to the first element.
- const T & **y** () const
Short access to the second element.
- T & **y** ()
Short access to the second element.
- void **y** (const T &v)
Short access to the second element.
- const T & **z** () const
Short access to the third element.
- T & **z** ()
Short access to the third element.
- void **z** (const T &v)
Short access to the third element.
- **Vector** & **zero** (void)

Static Public Member Functions

- static size_t **size** ()
Returns the size of the vector (i.e.

Protected Attributes

- T **elems** [dim]

Friends

- **Vector** operator* (const T &scalar, const **Vector** &vec)

Multiplication by a scalar.

- **QTextStream** & operator<< (**QTextStream** &out, const **Vector** &vec)
- std::ostream & operator<< (std::ostream &out, const **Vector** &vec)
- **QTextStream** & operator>> (**QTextStream** &in, **Vector** &vec)
- std::istream & operator>> (std::istream &in, **Vector** &vec)

Related Functions

(Note that these are not member functions.)

- template<class T >
double **angle** (const **Vector**< 3, T > &v1, const **Vector**< 3, T > &v2, const **Vector**< 3, T > &ref)

Oriented angle between v1 and v2 with ref to orient the space.

- template<class T >
double **angle** (const **Vector**< 1, T > &v1, const **Vector**< 1, T > &v2)

Oriented angle between v1 and v2.

- template<class T >
double **angle** (const **Vector**< 2, T > &v1, const **Vector**< 2, T > &v2)

Oriented angle between v1 and v2.

- template<class T >
double **angle** (const **Vector**< 3, T > &v1, const **Vector**< 3, T > &v2)

Non-oriented angle between v1 and v2.

- template<class T >
double **angle** (const **Vector**< 2, T > &v)

Angle of the vector with (0,1).

- template<size_t dim, typename T >
Vector< dim, T > **divide** (const **Vector**< dim, T > &v1, const **Vector**< dim, T > &v2)

Return the vector whose component is the ratio of the two input vectors components.

- template<size_t dim, typename T, typename T1, typename T2 >
Vector< dim, T > **map** (T(*fct)(T1, T2), const **Vector**< dim, T1 > &v1, const **Vector**< dim, T2 > &v2)

Map a function to all elements of a vector.

- template<size_t dim, typename T, typename T1, typename T2 >
Vector< dim, T > **map** (T(*fct)(const T1 &, const T2 &), const **Vector**< dim, T1 > &v1, const **Vector**< dim, T2 > &v2)
Map a function to all elements of a vector.
- template<size_t dim, typename T, typename T1, typename T2 >
Vector< dim, T > **map** (const T &(*fct)(const T1 &, const T2 &), const **Vector**< dim, T1 > &v1, const **Vector**< dim, T2 > &v2)
Map a function to all elements of a vector.
- template<size_t dim, typename T >
Vector< dim, T > **map** (T(*fct)(T, T), const **Vector**< dim, T > &v1, const **Vector**< dim, T > &v2)
Map a function to all elements of a vector.
- template<size_t dim, typename T >
Vector< dim, T > **map** (T(*fct)(const T &, const T &), const **Vector**< dim, T > &v1, const **Vector**< dim, T > &v2)
Map a function to all elements of a vector.
- template<size_t dim, typename T >
Vector< dim, T > **map** (const T &(*fct)(const T &, const T &), const **Vector**< dim, T > &v1, const **Vector**< dim, T > &v2)
Map a function to all elements of a vector.
- template<size_t dim, typename T, typename T1 >
Vector< dim, T > **map** (T(*fct)(T1), const **Vector**< dim, T1 > &v)
Map a function to all elements of a vector.
- template<size_t dim, typename T >
Vector< dim, T > **max** (const **Vector**< dim, T > &v1, const **Vector**< dim, T > &v2)
Return the vector whose component is the max of the two input vectors components.
- template<size_t dim, typename T >
Vector< dim, T > **min** (const **Vector**< dim, T > &v1, const **Vector**< dim, T > &v2)
Return the vector whose component is the min of the two input vectors components.
- template<size_t dim, typename T >
Vector< dim, T > **multiply** (const **Vector**< dim, T > &v1, const **Vector**< dim, T > &v2)
Return the vector whose component is the product of the two input vectors components.
- template<size_t dim, typename T >
T **norm** (const **Vector**< dim, T > &v)
Function-version of the norm.

- double `norm` (double s)
Euclidian norm of a real.
- template<size_t dim, typename T >
`Vector< dim, T > normalized` (const `Vector< dim, T > &v`)
Function-version of the square norm.
- double `normalized` (double)
Euclidian square norm of a real.
- template<size_t dim, typename T >
`T normsq` (const `Vector< dim, T > &v`)
Function-version of the square norm.
- double `normsq` (double s)
Euclidian square norm of a real.
- template<class T >
`Vector< 3, T > operator%` (const `Vector< 3, T > &v1`, const `Vector< 3, T > &v2`)
Cross product $v_1 \times v_2$.
- template<class T >
`T operator%` (const `Vector< 2, T > &v1`, const `Vector< 2, T > &v2`)
Cross product $v_1 \times v_2$.
- template<size_t dim, typename T >
`Vector< dim, T > operator+` (const T &value, const `Vector< dim, T > &v`)
Add a value to all elements of a vector.
- template<size_t dim, typename T >
`Vector< dim, T > operator+` (const `Vector< dim, T > &v`, const T &value)
Add a value to all elements of a vector.
- template<size_t dim, typename T >
`Vector< dim, T > operator-` (const T &value, const `Vector< dim, T > &v`)
Equivalent to subtracting a vector with all component the same to another one.
- template<size_t dim, typename T >
`Vector< dim, T > operator-` (const `Vector< dim, T > &v`, const T &value)
Subtract a value to all elements of a vector.
- template<class T >
`Vector< 3, T > operator^` (const `Vector< 3, T > &v1`, const `Vector< 3, T > &v2`)
Cross product $v_1 \times v_2$ (French notation).

- `template<class T >`
`T operator^ (const Vector< 1, T > &v1, const Vector< 1, T > &v2)`
Cross product $v_1 \times v_2$ (French notation).
- `template<class T >`
`T operator^ (const Vector< 2, T > &v1, const Vector< 2, T > &v2)`
Cross product $v_1 \times v_2$ (French notation).
- `template<typename T >`
`util::Vector< 3, T > orthogonal (const util::Vector< 3, T > &v)`
Find a vector orthogonal to v.

17.68.1 Detailed Description

`template<size_t dim, class T = double> class util::Vector< dim, T >`

`Vector` class supporting all classic classic vector operations.

Note

The addition and subtraction with a scalar is possible and is equivalent to adding/-subtracting a vector filled with the scalar.

Definition at line 32 of file vector.h.

17.68.2 Constructor & Destructor Documentation

17.68.2.1 `template<size_t dim, class T = double> util::Vector< dim, T >::Vector (const Vector< dim, T > & vec) [inline]`

Copy another vector.

Definition at line 49 of file vector.h.

```
50     {
51         for(size_t i = 0 ; i < dim ; i++)
52             elems[i] = vec[i];
53     }
```

17.68.2.2 `template<size_t dim, class T = double> template<size_t d1, class T1 > util::Vector< dim, T >::Vector (const Vector< d1, T1 > & vec) [inline, explicit]`

Copy another vector with different number of elements.

Definition at line 64 of file vector.h.

```

65     {
66         if (d1 < dim)
67         {
68             for (size_t i = 0 ; i < d1 ; ++i)
69                 elems[i] = vec[i];
70             for (size_t i = d1 ; i < dim ; ++i)
71                 elems[i] = 0;
72         }
73     else
74     {
75         for (size_t i = 0 ; i < dim ; ++i)
76             elems[i] = vec[i];
77     }
78 }

```

17.68.2.3 `template<size_t dim, class T = double> template<class Vec > util::Vector< dim, T >::Vector (const Vec & el) [inline, explicit]`

Initialize a vector from any object behaving like an array.

The only constraints are:

- the type has to be convertible to T
- the vector needs a [] operator
- the size of the vector has to be at least dim

Definition at line 89 of file vector.h.

```

90     {
91         for (size_t i = 0 ; i < dim ; i++)
92             elems[i] = el[i];
93     }

```

17.68.2.4 `template<size_t dim, class T = double> util::Vector< dim, T >::Vector (const T & x = T()) [inline]`

Initialize a vector with all values to x.

x default to a default-constructed object.

Definition at line 100 of file vector.h.

```

101     {
102         for ( size_t i = 0 ; i < dim ; ++i )
103             elems[ i ] = x;
104     }

```


**17.68.2.5 template<size_t dim, class T = double> util::Vector< dim, T
>::Vector (const T & x, const T & y) [inline, explicit]**

Initialize a 2D vector.

Definition at line 109 of file vector.h.

```
110     {  
111         STATIC_ASSERT( dim == 2, "Constructor with 2 arguments can be used only for  
2D vectors" );  
112         elems[ 0 ] = x;  
113         elems[ 1 ] = y;  
114     }
```

**17.68.2.6 template<size_t dim, class T = double> util::Vector< dim, T
>::Vector (const T & x, const T & y, const T & z) [inline,
explicit]**

Initialize a 3D vector.

Definition at line 119 of file vector.h.

```
120     {  
121         STATIC_ASSERT( dim == 3, "Constructor with 3 arguments can be used only for  
3D vectors" );  
122         elems[ 0 ] = x;  
123         elems[ 1 ] = y;  
124         elems[ 2 ] = z;  
125     }
```

**17.68.2.7 template<size_t dim, class T = double> util::Vector< dim, T
>::Vector (const T & x, const T & y, const T & z, const T & t)
[inline, explicit]**

Initialize a 4D vector.

Definition at line 130 of file vector.h.

```
131     {  
132         STATIC_ASSERT( dim == 4, "Constructor with 3 arguments can be used only for  
3D vectors" );  
133         elems[ 0 ] = x;  
134         elems[ 1 ] = y;  
135         elems[ 2 ] = z;  
136         elems[ 3 ] = t;  
137     }
```

17.68.3 Member Function Documentation**17.68.3.1 template<size_t dim, class T = double> const_iterator util::Vector<
dim, T >::begin () const [inline]**

Stl-iteration constant begin.

Definition at line 156 of file vector.h.

```
156 { return elems; }
```

17.68.3.2 `template<size_t dim, class T = double> iterator util::Vector< dim, T >::begin () [inline]`

STL-iteration begin.

Definition at line 152 of file vector.h.

```
152 { return elems; }
```

17.68.3.3 `template<size_t dim, class T = double> const T* util::Vector< dim, T >::c_data () const [inline]`

Returns a constant raw pointer on the data.

Reimplemented in [util::Point< T >](#).

Definition at line 170 of file vector.h.

Referenced by `util::Matrix< nb_vars, nb_vars, double >::c_data()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell()`, `algorithms::drawSymetricVVGraph()`, and `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawWalledCell()`.

```
170 { return elems; }
```

17.68.3.4 `template<size_t dim, class T = double> Vector util::Vector< dim, T >::cross (const Vector< dim, T > & other) const [inline]`

Compute the cross product as `this x other`.

Definition at line 498 of file vector.h.

```
499 {
500     STATIC_ASSERT( dim == 3, "Cross product is only defined for 3D vectors." );

501     return ( *this ) ^ other;
502 }
```

17.68.3.5 template<size_t dim, class T = double> T* util::Vector< dim, T >::data () [inline]

Returns a raw pointer on the data.

Definition at line 147 of file vector.h.

Referenced by util::Matrix< nb_vars, nb_vars, double >::data().

```
147 { return elems; }
```

17.68.3.6 template<size_t dim, class T = double> const_iterator util::Vector< dim, T >::end () const [inline]

Stl-iteration constant end.

Definition at line 165 of file vector.h.

```
165 { return elems+dim; }
```

17.68.3.7 template<size_t dim, class T = double> iterator util::Vector< dim, T >::end () [inline]

STL-iteration end.

Definition at line 161 of file vector.h.

```
161 { return elems+dim; }
```

17.68.3.8 template<size_t dim, class T = double> const T& util::Vector< dim, T >::i () const [inline]

Short access to the first element.

Definition at line 588 of file vector.h.

```
588 { STATIC_ASSERT( dim > 0, "Accessing i requires at least a 1D vector." ); return  
    elems[0]; }
```

17.68.3.9 template<size_t dim, class T = double> T& util::Vector< dim, T >::i () [inline]

Short access to the first element.

Definition at line 572 of file vector.h.

Referenced by `util::Vector< nCols, double >::mult()`, `util::Vector< nCols, double >::normsq()`, `util::Vector< nCols, double >::operator*()`, `util::Vector< nCols, double >::operator*=(, geometry::Quaternion::operator*=(, util::Vector< nCols, double >::operator+=(, geometry::Quaternion::operator+=(, util::Vector< nCols, double >::operator-(), util::Vector< nCols, double >::operator-=(, util::Vector< nCols, double >::operator/=(, geometry::Quaternion::operator/=(, util::Vector< nCols, double >::operator<(), util::Vector< nCols, double >::operator<=(, util::Vector< nCols, double >::operator=(), util::Vector< nCols, double >::operator==(, util::Vector< nCols, double >::operator>(), util::Vector< nCols, double >::operator>=(), and util::Vector< nCols, double >::Vector().`

```
572 { STATIC_ASSERT( dim > 0, "Accessing i requires at least a 1D vector." ); return
    elems[0]; }
```

17.68.3.10 `template<size_t dim, class T = double> void util::Vector< dim, T >::i (const T & v) [inline]`

Short access to the first element.

Definition at line 556 of file `vector.h`.

```
556 { STATIC_ASSERT( dim > 0, "Accessing i requires at least a 1D vector." ); elems[0]
    = v; }
```

17.68.3.11 `template<size_t dim, class T = double> const T& util::Vector< dim, T >::j () const [inline]`

Short access to the second element.

Definition at line 592 of file `vector.h`.

```
592 { STATIC_ASSERT( dim > 1, "Accessing j requires at least a 2D vector." ); return
    elems[1]; }
```

17.68.3.12 `template<size_t dim, class T = double> T& util::Vector< dim, T >::j () [inline]`

Short access to the second element.

Definition at line 576 of file `vector.h`.

```
576 { STATIC_ASSERT( dim > 1, "Accessing j requires at least a 2D vector." ); return
    elems[1]; }
```

17.68.3.13 `template<size_t dim, class T = double> void util::Vector< dim, T
>::j(const T & v) [inline]`

Short access to the second element.

Definition at line 560 of file vector.h.

```
560 { STATIC_ASSERT( dim > 1, "Accessing j requires at least a 2D vector." ); elems[1  
    ] = v; }
```

17.68.3.14 `template<size_t dim, class T = double> const T& util::Vector< dim,
T >::k() const [inline]`

Short access to the third element.

Definition at line 596 of file vector.h.

```
596 { STATIC_ASSERT( dim > 2, "Accessing k requires at least a 3D vector." ); return  
    elems[2]; }
```

17.68.3.15 `template<size_t dim, class T = double> T& util::Vector< dim, T
>::k() [inline]`

Short access to the third element.

Definition at line 580 of file vector.h.

```
580 { STATIC_ASSERT( dim > 2, "Accessing k requires at least a 3D vector." ); return  
    elems[2]; }
```

17.68.3.16 `template<size_t dim, class T = double> void util::Vector< dim, T
>::k(const T & v) [inline]`

Short access to the third element.

Definition at line 564 of file vector.h.

```
564 { STATIC_ASSERT( dim > 2, "Accessing k requires at least a 3D vector." ); elems[2  
    ] = v; }
```

17.68.3.17 `template<size_t dim, class T = double> const T& util::Vector< dim,
T >::l() const [inline]`

Short access to the fourth element.

Definition at line 600 of file vector.h.

```
600 { STATIC_ASSERT( dim > 3, "Accessing l requires at least a 4D vector." ); return  
    elems[3]; }
```

17.68.3.18 `template<size_t dim, class T = double> T& util::Vector< dim, T >::l () [inline]`

Short access to the fourth element.

Definition at line 584 of file vector.h.

```
584 { STATIC_ASSERT( dim > 3, "Accessing l requires at least a 4D vector." ); return
    elems[3]; }
```

17.68.3.19 `template<size_t dim, class T = double> void util::Vector< dim, T >::l (const T & v) [inline]`

Short access to the fourth element.

Definition at line 568 of file vector.h.

```
568 { STATIC_ASSERT( dim > 3, "Accessing l requires at least a 4D vector." ); elems[3
    ] = v; }
```

17.68.3.20 `template<size_t dim, class T = double> Vector util::Vector< dim, T >::mult (const Vector< dim, T > & vec) const [inline]`

Element-wise multiplication.

Definition at line 207 of file vector.h.

```
208 {
209     Vector ans;
210     for(size_t i = 0 ; i < dim ; i++)
211         ans[i] = elems[i] * vec.elems[i];
212     return ans;
213 }
```

17.68.3.21 `template<size_t dim, class T = double> T util::Vector< dim, T >::norm () const [inline]`

Euclidean norm of the vector.

Definition at line 391 of file vector.h.

Referenced by `util::Vector< dim, T >::norm()`, and `util::Vector< nCols, double >::normalize()`.

```
392 {
393     return std::sqrt(normsq());
394 }
```

17.68.3.22 `template<size_t dim, class T = double> Vector& util::Vector< dim, T >::normalize (void) [inline]`

Normalize the vector.

Definition at line 411 of file vector.h.

Referenced by `util::Vector< nCols, double >::normalized()`, `complex_factory::objreader()`, `cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_division()`, and `util::Contour::tangent()`.

```
412     {
413         T sz = norm();
414         return ((*this) /= sz);
415     }
```

17.68.3.23 `template<size_t dim, class T = double> Vector util::Vector< dim, T >::normalized (void) const [inline]`

Returns a normalized version of the vector.

Definition at line 420 of file vector.h.

Referenced by `util::Vector< dim, T >::normalized()`.

```
421     {
422         Vector ans(*this);
423         return ans.normalize();
424     }
```

17.68.3.24 `template<size_t dim, class T = double> T util::Vector< dim, T >::normsq () const [inline]`

Square of the Euclidean norm of the vector.

Definition at line 399 of file vector.h.

Referenced by `geometry::Quaternion::inverse()`, `util::Vector< nCols, double >::norm()`, and `util::Vector< dim, T >::normsq()`.

```
400     {
401         T ans = 0;
402         for(size_t i = 0 ; i < dim ; i++)
403             ans += elems[i] * elems[i];
404
405         return ans;
406     }
```

17.68.3.25 `template<size_t dim, class T = double> bool util::Vector< dim, T >::operator!= (const Vector< dim, T > & vec) const [inline]`

Element-wise inequality.

Definition at line 367 of file vector.h.

```

368     {
369         return (!(*this) == vec);
370     }

```

17.68.3.26 `template<size_t dim, class T = double> T util::Vector< dim, T >::operator* (const Vector< dim, T > & vec) const` `[inline]`

Dot product.

Definition at line 270 of file vector.h.

```

271     {
272         T ans = 0;
273         for(size_t i = 0 ; i < dim ; i++)
274             ans += elems[i] * vec.elems[i];
275
276         return ans;
277     }

```

17.68.3.27 `template<size_t dim, class T = double> Vector util::Vector< dim, T >::operator* (const T & scalar) const` `[inline]`

Multiplication by a scalar.

Definition at line 218 of file vector.h.

```

219     {
220         Vector ans(*this);
221         ans *= scalar;
222         return ans;
223     }

```

17.68.3.28 `template<size_t dim, class T = double> template<typename T1 > Vector& util::Vector< dim, T >::operator*= (const T1 & scalar)` `[inline]`

In-place multiplication by a scalar.

Definition at line 324 of file vector.h.

```

325     {
326         for(size_t i = 0 ; i < dim ; i++)
327             elems[i] = (T) (elems[i]*scalar);
328         return *this;
329     }

```


17.68.3.29 `template<size_t dim, class T = double> Vector& util::Vector< dim, T >::operator*=(const T & scalar) [inline]`

In-place multiplication by a scalar.

Reimplemented in [util::Point< T >](#).

Definition at line 313 of file vector.h.

```
314     {
315         for(size_t i = 0 ; i < dim ; i++)
316             elems[i] *= scalar;
317         return *this;
318     }
```

17.68.3.30 `template<size_t dim, class T = double> Vector util::Vector< dim, T >::operator+(const Vector< dim, T > & vec) const [inline]`

[Vector](#) addition.

Definition at line 187 of file vector.h.

```
188     {
189         Vector ans(*this);
190         ans += vec;
191         return ans;
192     }
```

17.68.3.31 `template<size_t dim, class T = double> Vector& util::Vector< dim, T >::operator+=(const Vector< dim, T > & vec) [inline]`

In-place vector addition.

Definition at line 293 of file vector.h.

```
294     {
295         for(size_t i = 0 ; i < dim ; i++)
296             elems[i] += vec.elems[i];
297         return *this;
298     }
```

17.68.3.32 `template<size_t dim, class T = double> Vector util::Vector< dim, T >::operator-(const Vector< dim, T > & vec) const [inline]`

[Vector](#) subtraction.

Definition at line 197 of file vector.h.

```
198     {
199         Vector ans(*this);
200         ans -= vec;
201         return ans;
202     }
```

17.68.3.33 `template<size_t dim, class T = double> Vector util::Vector< dim, T >::operator- (void) const [inline]`

[Vector](#) negation.

Reimplemented in [util::Point< T >](#).

Definition at line 175 of file vector.h.

```

176     {
177         Vector ans;
178         for(size_t i = 0 ; i < dim ; i++)
179             ans[i] = -elems[i];
180
181         return ans;
182     }
```

17.68.3.34 `template<size_t dim, class T = double> Vector& util::Vector< dim, T >::operator-= (const Vector< dim, T > & vec) [inline]`

In-place vector subtraction.

Definition at line 303 of file vector.h.

```

304     {
305         for(size_t i = 0 ; i < dim ; i++)
306             elems[i] -= vec.elems[i];
307         return *this;
308     }
```

17.68.3.35 `template<size_t dim, class T = double> Vector util::Vector< dim, T >::operator/ (const Vector< dim, T > & vec) const [inline]`

Element-wise division.

Definition at line 238 of file vector.h.

```

239     {
240         Vector ans = *this;
241         ans /= vec;
242         return ans;
243     }
```

17.68.3.36 `template<size_t dim, class T = double> Vector util::Vector< dim, T >::operator/ (const T & scalar) const [inline]`

Division by a scalar.

Reimplemented in [util::Point< T >](#).

Definition at line 228 of file vector.h.

```
229     {
230         Vector ans(*this);
231         ans /= scalar;
232         return ans;
233     }
```

17.68.3.37 `template<size_t dim, class T = double> template<typename T1 >
Vector& util::Vector< dim, T >::operator/= (const T1 & scalar)
[inline]`

In-place division by a scalar.

Definition at line 345 of file vector.h.

```
346     {
347         for(size_t i = 0 ; i < dim ; ++i)
348             elems[i] = (T) (elems[i] / scalar);
349         return *this;
350     }
```

17.68.3.38 `template<size_t dim, class T = double> Vector& util::Vector< dim,
T >::operator/= (const T & scalar) [inline]`

In-place division by a scalar.

Reimplemented in [util::Point< T >](#).

Definition at line 334 of file vector.h.

```
335     {
336         for(size_t i = 0 ; i < dim ; ++i)
337             elems[i] /= scalar;
338         return *this;
339     }
```

17.68.3.39 `template<size_t dim, class T = double> Vector& util::Vector< dim,
T >::operator/= (const Vector< dim, T > & vec) [inline]`

In-place element-wise division by a scalar.

Definition at line 248 of file vector.h.

```
249     {
250         for(size_t i = 0 ; i < dim ; ++i)
251             elems[i] /= vec.elems[i];
252         return *this;
253     }
```

17.68.3.40 `template<size_t dim, class T = double> bool util::Vector< dim, T >::operator< (const Vector< dim, T > & other) const [inline]`

Comparison operator.

Compare the axis in order.

Definition at line 616 of file vector.h.

```

617     {
618         for(size_t i = 0 ; i < dim ; ++i)
619         {
620             if(elems[i] < other.elems[i]) return true;
621             if(elems[i] > other.elems[i]) return false;
622         }
623         return false;
624     }
```

17.68.3.41 `template<size_t dim, class T = double> bool util::Vector< dim, T >::operator<= (const Vector< dim, T > & other) const [inline]`

Comparison operator.

Compare the axis in order.

Definition at line 631 of file vector.h.

```

632     {
633         for(size_t i = 0 ; i < dim ; ++i)
634         {
635             if(elems[i] < other.elems[i]) return true;
636             if(elems[i] > other.elems[i]) return false;
637         }
638         return true;
639     }
```

17.68.3.42 `template<size_t dim, class T = double> Vector& util::Vector< dim, T >::operator= (const T & value) [inline]`

Set all the elements to value.

Reimplemented in [util::Color< T >](#), and [util::Color< GLfloat >](#).

Definition at line 486 of file vector.h.

```

487     {
488         for( size_t i = 0 ; i < dim ; ++i )
489         {
490             elems[ i ] = value;
491         }
492         return *this;
493     }
```

17.68.3.43 `template<size_t dim, class T = double> Vector& util::Vector< dim, T >::operator= (const Vector< dim, T > & vec) [inline]`

Vector copy.

Definition at line 282 of file vector.h.

```

283     {
284         for(size_t i = 0 ; i < dim ; i++)
285             elems[i] = vec.elems[i];
286
287         return (*this);
288     }
```

17.68.3.44 `template<size_t dim, class T = double> bool util::Vector< dim, T >::operator== (const Vector< dim, T > & vec) const [inline]`

Element-wise equality.

Definition at line 355 of file vector.h.

```

356     {
357         for(size_t i = 0 ; i < dim ; i++)
358             if(elems[i] != vec.elems[i])
359                 return false;
360
361         return true;
362     }
```

17.68.3.45 `template<size_t dim, class T = double> bool util::Vector< dim, T >::operator> (const Vector< dim, T > & other) const [inline]`

Comparison operator.

Compare the axis in order.

Definition at line 646 of file vector.h.

```

647     {
648         for(size_t i = 0 ; i < dim ; ++i)
649         {
650             if(elems[i] > other.elems[i]) return true;
651             if(elems[i] < other.elems[i]) return false;
652         }
653         return false;
654     }
```

17.68.3.46 `template<size_t dim, class T = double> bool util::Vector< dim, T >::operator>= (const Vector< dim, T > & other) const [inline]`

Comparison operator.

Compare the axis in order.

Definition at line 661 of file vector.h.

```
662     {
663         for(size_t i = 0 ; i < dim ; ++i)
664         {
665             if(elems[i] > other.elems[i]) return true;
666             if(elems[i] < other.elems[i]) return false;
667         }
668         return true;
669     }
```

17.68.3.47 `template<size_t dim, class T = double> T util::Vector< dim, T >::operator[] (size_t idx) const` `[inline]`

Access to the element *idx*.

Definition at line 383 of file vector.h.

```
384     {
385         return elems[idx];
386     }
```

17.68.3.48 `template<size_t dim, class T = double> T& util::Vector< dim, T >::operator[] (size_t idx)` `[inline]`

Access to the element *idx*.

Definition at line 375 of file vector.h.

```
376     {
377         return elems[idx];
378     }
```

17.68.3.49 `template<size_t dim, class T = double> Vector<2,T> util::Vector< dim, T >::projectXY (void)` `[inline]`

Extract the two first elements of the vector.

Definition at line 605 of file vector.h.

```
606     {
607         STATIC_ASSERT( dim>1, "2D projection requires at least a 2D vector." );
608         return Vector<2,T>(elems[0],elems[1]);
609     }
```

17.68.3.50 `template<size_t dim, class T = double> void util::Vector< dim, T
>::set (const T & x, const T & y, const T & z, const T & t)
[inline]`

Set the values of a 4-D vector.

Definition at line 474 of file vector.h.

```
475     {  
476         STATIC_ASSERT( dim == 4, "Set method with 4 arguments can be used only for  
4D vectors." );  
477         elems[ 0 ] = x;  
478         elems[ 1 ] = y;  
479         elems[ 2 ] = z;  
480         elems[ 3 ] = t;  
481     }
```

17.68.3.51 `template<size_t dim, class T = double> void util::Vector< dim, T
>::set (const T & x, const T & y, const T & z) [inline]`

Set the values of a 3-D vector.

Definition at line 463 of file vector.h.

```
464     {  
465         STATIC_ASSERT( dim == 3, "Set method with 3 arguments can be used only for  
3D vectors." );  
466         elems[ 0 ] = x;  
467         elems[ 1 ] = y;  
468         elems[ 2 ] = z;  
469     }
```

17.68.3.52 `template<size_t dim, class T = double> void util::Vector< dim, T
>::set (const T & x, const T & y) [inline]`

Set the values of a 2-D vector.

Definition at line 453 of file vector.h.

```
454     {  
455         STATIC_ASSERT( dim == 2, "Set method with 2 arguments can be used only for  
2D vectors." );  
456         elems[ 0 ] = x;  
457         elems[ 1 ] = y;  
458     }
```

17.68.3.53 `template<size_t dim, class T = double> void util::Vector< dim, T
>::set (const T & x) [inline]`

Set the values of a 1-D vector.

Definition at line 444 of file vector.h.

```

445     {
446         STATIC_ASSERT( dim == 1, "Set method with 1 argument can be used only for 1
D vectors." );
447         elems[ 0 ] = x;
448     }

```

17.68.3.54 `template<size_t dim, class T = double> static size_t util::Vector<dim, T>::size() [inline, static]`

Returns the size of the vector (i.e.

the number of elements)

Definition at line 142 of file vector.h.

Referenced by `complex_factory::objreader()`.

```

142 { return dim; }

```

17.68.3.55 `template<size_t dim, class T = double> const T& util::Vector<dim, T>::t() const [inline]`

Short access to the fourth element.

Definition at line 551 of file vector.h.

```

551 { STATIC_ASSERT( dim > 3, "Accessing t requires at least a 4D vector." ); return
    elems[3]; }

```

17.68.3.56 `template<size_t dim, class T = double> T& util::Vector<dim, T>::t() [inline]`

Short access to the fourth element.

Definition at line 535 of file vector.h.

Referenced by `util::Vector<nCols, double>::set()`.

```

535 { STATIC_ASSERT( dim > 3, "Accessing t requires at least a 4D vector." ); return
    elems[3]; }

```

17.68.3.57 `template<size_t dim, class T = double> void util::Vector<dim, T>::t(const T & v) [inline]`

Short access to the fourth element.

Definition at line 519 of file vector.h.

Referenced by `util::Matrix<nb_vars, nb_vars, double>::rotation()`.

```

519 { STATIC_ASSERT( dim > 3, "Accessing t requires at least a 4D vector." ); elems[3
    ] = v; }

```


17.68.3.58 `template<size_t dim, class T = double> const T& util::Vector< dim, T >::x () const [inline]`

Short access to the first element.

Definition at line 539 of file vector.h.

```
539 { STATIC_ASSERT( dim > 0, "Accessing x requires at least a 1D vector." ); return
    elems[0]; }
```

17.68.3.59 `template<size_t dim, class T = double> T& util::Vector< dim, T >::x () [inline]`

Short access to the first element.

Definition at line 523 of file vector.h.

Referenced by `geometry::Quaternion::conjugate()`, `geometry::Quaternion::inverse()`, `util::Vector< nCols, double >::set()`, and `util::Vector< nCols, double >::Vector()`.

```
523 { STATIC_ASSERT( dim > 0, "Accessing x requires at least a 1D vector." ); return
    elems[0]; }
```

17.68.3.60 `template<size_t dim, class T = double> void util::Vector< dim, T >::x (const T & v) [inline]`

Short access to the first element.

Definition at line 507 of file vector.h.

Referenced by `util::Vector< dim, T >::angle()`, `tissue_model::TissueModel< RealModel, TissueClass >::draw()`, `complex_factory::hex_grid()`, `util::Contour::normal()`, `util::Function::normalizeX()`, `util::Function::normalizeY()`, `complex_factory::objreader()`, `util::Function::operator()`, `util::Vector< dim, T >::orthogonal()`, `util::Function::reread()`, `util::Matrix< nb_vars, nb_vars, double >::rotation()`, `solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson()`, and `complex_factory::square_grid()`.

```
507 { STATIC_ASSERT( dim > 0, "Accessing x requires at least a 1D vector." ); elems[0]
    = v; }
```

17.68.3.61 `template<size_t dim, class T = double> const T& util::Vector< dim, T >::y () const [inline]`

Short access to the second element.

Definition at line 543 of file vector.h.

```
543 { STATIC_ASSERT( dim > 1, "Accessing y requires at least a 2D vector." ); return
    elems[1]; }
```

17.68.3.62 `template<size_t dim, class T = double> T& util::Vector< dim, T >::y () [inline]`

Short access to the second element.

Definition at line 527 of file vector.h.

Referenced by `geometry::Quaternion::conjugate()`, `geometry::Quaternion::inverse()`, and `util::Vector< nCols, double >::set()`.

```
527 { STATIC_ASSERT( dim > 1, "Accessing y requires at least a 2D vector." ); return
    elems[1]; }
```

17.68.3.63 `template<size_t dim, class T = double> void util::Vector< dim, T >::y (const T & v) [inline]`

Short access to the second element.

Definition at line 511 of file vector.h.

Referenced by `util::Vector< dim, T >::angle()`, `tissue_model::TissueModel< RealModel, TissueClass >::draw()`, `complex_factory::hex_grid()`, `util::Contour::normal()`, `complex_factory::objreader()`, `util::Vector< dim, T >::orthogonal()`, `util::Function::reread()`, `util::Matrix< nb_vars, nb_vars, double >::rotation()`, and `complex_factory::square_grid()`.

```
511 { STATIC_ASSERT( dim > 1, "Accessing y requires at least a 2D vector." ); elems[1]
    = v; }
```

17.68.3.64 `template<size_t dim, class T = double> const T& util::Vector< dim, T >::z () const [inline]`

Short access to the third element.

Definition at line 547 of file vector.h.

```
547 { STATIC_ASSERT( dim > 2, "Accessing z requires at least a 3D vector." ); return
    elems[2]; }
```

17.68.3.65 `template<size_t dim, class T = double> T& util::Vector< dim, T >::z () [inline]`

Short access to the third element.

Definition at line 531 of file vector.h.

Referenced by `geometry::Quaternion::conjugate()`, `geometry::Quaternion::inverse()`, and `util::Vector< nCols, double >::set()`.

```
531 { STATIC_ASSERT( dim > 2, "Accessing z requires at least a 3D vector." ); return
    elems[2]; }
```

17.68.3.66 `template<size_t dim, class T = double> void util::Vector< dim, T >::z(const T & v) [inline]`

Short access to the third element.

Definition at line 515 of file vector.h.

Referenced by `tissue_model::TissueModel< RealModel, TissueClass >::draw()`, `complex_factory::objreader()`, `util::Vector< dim, T >::orthogonal()`, and `util::Matrix< nb_vars, nb_vars, double >::rotation()`.

```
515 { STATIC_ASSERT( dim > 2, "Accessing z requires at least a 3D vector." ); elems[2]
    = v; }
```

17.68.4 Friends And Related Function Documentation

17.68.4.1 `template<class T> double angle(const Vector< 3, T> & v1, const Vector< 3, T> & v2, const Vector< 3, T> & ref) [related]`

Oriented angle between `v1` and `v2` with `ref` to orient the space.

Definition at line 825 of file vector.h.

```
826 {
827     double x = v1*v2;
828     Vector<3,T> n = v1^v2;
829     double y = norm( n );
830     if( n*ref < 0 )
831         return atan2( -y, x );
832     else
833         return atan2( y, x );
834 }
```

17.68.4.2 `template<class T> double angle(const Vector< 1, T> & v1, const Vector< 1, T> & v2) [related]`

Oriented angle between `v1` and `v2`.

Definition at line 814 of file vector.h.

```
815 {
816     return ( v1*v2 < 0 )? -1 : 1;
817 }
```

17.68.4.3 `template<class T> double angle(const Vector< 2, T> & v1, const Vector< 2, T> & v2) [related]`

Oriented angle between `v1` and `v2`.

Definition at line 801 of file vector.h.

```

802  {
803      double x = v1*v2;
804      double y = v1^v2;
805      return atan2( y, x );
806  }

```

17.68.4.4 `template<class T> double angle (const Vector< 3, T> & v1, const Vector< 3, T> & v2)` **[related]**

Non-oriented angle between v1 and v2.

Definition at line 788 of file vector.h.

```

789  {
790      double x = v1*v2;
791      double y = norm( v1^v2 );
792      return atan2( y, x );
793  }

```

17.68.4.5 `template<class T> double angle (const Vector< 2, T> & v)` **[related]**

Angle of the vector with (0,1).

Definition at line 777 of file vector.h.

References `util::Vector< dim, T>::x()`, and `util::Vector< dim, T>::y()`.

```

778  {
779      return atan2( v.y(), v.x() );
780  }

```

17.68.4.6 `template<size_t dim, typename T> Vector< dim, T> divide (const Vector< dim, T> & v1, const Vector< dim, T> & v2)` **[related]**

Return the vector whose component is the ratio of the two input vectors components.

Definition at line 966 of file vector.h.

```

967  {
968      Vector<dim,T> result;
969      for(size_t i = 0 ; i < dim ; ++i)
970      {
971          result[i] = v1[i] / v2[i];
972      }
973      return result;
974  }

```

17.68.4.7 `template<size_t dim, typename T , typename T1 , typename T2 >
Vector< dim, T > map (T(*) (T1, T2) fct, const Vector< dim, T1 > &
v1, const Vector< dim, T2 > & v2) [related]`

Map a function to all elements of a vector.

Definition at line 1176 of file vector.h.

```
1177 {
1178     Vector<dim,T> result;
1179     for(size_t i = 0 ; i < dim ; ++i)
1180     {
1181         result[i] = (*fct) (v1[i], v2[i]);
1182     }
1183     return result;
1184 }
```

17.68.4.8 `template<size_t dim, typename T , typename T1 , typename T2
> Vector< dim, T > map (T(*) (const T1 &, const T2 &) fct,
const Vector< dim, T1 > & v1, const Vector< dim, T2 > & v2)
[related]`

Map a function to all elements of a vector.

Definition at line 1160 of file vector.h.

```
1161 {
1162     Vector<dim,T> result;
1163     for(size_t i = 0 ; i < dim ; ++i)
1164     {
1165         result[i] = (*fct) (v1[i], v2[i]);
1166     }
1167     return result;
1168 }
```

17.68.4.9 `template<size_t dim, typename T , typename T1 , typename T2 >
Vector< dim, T > map (const T &(*) (const T1 &, const T2 &) fct,
const Vector< dim, T1 > & v1, const Vector< dim, T2 > & v2)
[related]`

Map a function to all elements of a vector.

Definition at line 1144 of file vector.h.

```
1145 {
1146     Vector<dim,T> result;
1147     for(size_t i = 0 ; i < dim ; ++i)
1148     {
1149         result[i] = (*fct) (v1[i], v2[i]);
1150     }
1151     return result;
1152 }
```

17.68.4.10 `template<size_t dim, typename T > Vector< dim, T > map (T(*) (T, T)fct, const Vector< dim, T > & v1, const Vector< dim, T > & v2)` [**related**]

Map a function to all elements of a vector.

Definition at line 1128 of file vector.h.

```

1129  {
1130      Vector<dim,T> result;
1131      for(size_t i = 0 ; i < dim ; ++i)
1132      {
1133          result[i] = (*fct) (v1[i], v2[i]);
1134      }
1135      return result;
1136  }
```

17.68.4.11 `template<size_t dim, typename T > Vector< dim, T > map (T(*) (const T &, const T &)fct, const Vector< dim, T > & v1, const Vector< dim, T > & v2)` [**related**]

Map a function to all elements of a vector.

Definition at line 1112 of file vector.h.

```

1113  {
1114      Vector<dim,T> result;
1115      for(size_t i = 0 ; i < dim ; ++i)
1116      {
1117          result[i] = (*fct) (v1[i], v2[i]);
1118      }
1119      return result;
1120  }
```

17.68.4.12 `template<size_t dim, typename T > Vector< dim, T > map (const T &(*) (const T &, const T &)fct, const Vector< dim, T > & v1, const Vector< dim, T > & v2)` [**related**]

Map a function to all elements of a vector.

Definition at line 1096 of file vector.h.

```

1097  {
1098      Vector<dim,T> result;
1099      for(size_t i = 0 ; i < dim ; ++i)
1100      {
1101          result[i] = (*fct) (v1[i], v2[i]);
1102      }
1103      return result;
1104  }
```

17.68.4.13 `template<size_t dim, typename T, typename T1 > Vector< dim, T
> map (T(*) (T1) fct, const Vector< dim, T1 > & v)` [**related**]

Map a function to all elements of a vector.

Definition at line 1080 of file vector.h.

```
1081 {  
1082     Vector<dim,T> result;  
1083     for(size_t i = 0 ; i < dim ; ++i)  
1084     {  
1085         result[i] = (*fct) (v[i]);  
1086     }  
1087     return result;  
1088 }
```

17.68.4.14 `template<size_t dim, typename T > Vector< dim, T > max
(const Vector< dim, T > & v1, const Vector< dim, T > & v2)` [**related**]

Return the vector whose component is the max of the two input vectors components.

Definition at line 915 of file vector.h.

```
916 {  
917     Vector<dim,T> result;  
918     for(size_t i = 0 ; i < dim ; ++i)  
919     {  
920         result[i] = std::max(v1[i], v2[i]);  
921     }  
922     return result;  
923 }
```

17.68.4.15 `template<size_t dim, typename T > Vector< dim, T > min
(const Vector< dim, T > & v1, const Vector< dim, T > & v2)` [**related**]

Return the vector whose component is the min of the two input vectors components.

Definition at line 932 of file vector.h.

```
933 {  
934     Vector<dim,T> result;  
935     for(size_t i = 0 ; i < dim ; ++i)  
936     {  
937         result[i] = std::min(v1[i], v2[i]);  
938     }  
939     return result;  
940 }
```

17.68.4.16 `template<size_t dim, typename T > Vector< dim, T > multiply` `(const Vector< dim, T > & v1, const Vector< dim, T > & v2)` **[related]**

Return the vector whose component is the product of the two input vectors components.

Definition at line 949 of file vector.h.

```

950     {
951         Vector<dim,T> result;
952         for (size_t i = 0 ; i < dim ; ++i)
953         {
954             result[i] = v1[i] * v2[i];
955         }
956         return result;
957     }
```

17.68.4.17 `template<size_t dim, typename T > T norm (const Vector< dim, T` `> & v) [related]`

Function-version of the norm.

See also

[Vector::norm\(\)](#)

Definition at line 879 of file vector.h.

References `util::Vector< dim, T >::norm()`.

```

880     {
881         return v.norm();
882     }
```

17.68.4.18 `template<size_t dim, class T = double> double norm (double s)` **[related]**

Euclidian norm of a real.

Just the absolute value of that real

Definition at line 867 of file vector.h.

```

868     {
869         return (s<0)?-s:s;
870     }
```

17.68.4.19 `template<size_t dim, typename T > Vector< dim, T > normalized` `(const Vector< dim, T > & v) [related]`

Function-version of the square norm.

See also[Vector::normsq\(\)](#)

Definition at line 903 of file vector.h.

References util::Vector< dim, T >::normalized().

```
904 {  
905     return v.normalized();  
906 }
```

**17.68.4.20 template<size_t dim, class T = double> double normalized (double)
[related]**

Euclidian square norm of a real.

Just the square of the real

Definition at line 843 of file vector.h.

```
844 {  
845     return l;  
846 }
```

**17.68.4.21 template<size_t dim, typename T > T normsq (const Vector< dim,
T > & v) [related]**

Function-version of the square norm.

See also[Vector::normsq\(\)](#)

Definition at line 891 of file vector.h.

References util::Vector< dim, T >::normsq().

```
892 {  
893     return v.normsq();  
894 }
```

**17.68.4.22 template<size_t dim, class T = double> double normsq (double s)
[related]**

Euclidian square norm of a real.

Just the square of the real

Definition at line 855 of file vector.h.

```
856 {  
857     return s*s;  
858 }
```

17.68.4.23 `template<class T> Vector< 3, T> operator% (const Vector< 3, T> & v1, const Vector< 3, T> & v2) [related]`

Cross product $v1 \times v2$.

Definition at line 751 of file vector.h.

```
752  {
753      return v1^v2;
754  }
```

17.68.4.24 `template<class T> T operator% (const Vector< 2, T> & v1, const Vector< 2, T> & v2) [related]`

Cross product $v1 \times v2$.

Definition at line 717 of file vector.h.

```
718  {
719      return v1^v2;
720  }
```

17.68.4.25 `template<size_t dim, class T = double> Vector operator* (const T & scalar, const Vector< dim, T> & vec) [friend]`

Multiplication by a scalar.

Definition at line 258 of file vector.h.

```
259  {
260      Vector ans;
261      for(size_t i = 0 ; i < dim ; i++)
262          ans[i] = scalar * vec.elems[i];
263
264      return ans;
265  }
```

17.68.4.26 `template<size_t dim, typename T> Vector< dim, T> operator+ (const T & value, const Vector< dim, T> & v) [related]`

Add a value to all elements of a vector.

Definition at line 1206 of file vector.h.

```
1207  {
1208      Vector<dim,T> res;
1209      for(size_t i = 0 ; i < dim ; ++i)
1210          res[i] = v[i] + value;
1211      return res;
1212  }
```

17.68.4.27 `template<size_t dim, typename T> Vector< dim, T > operator+(const Vector< dim, T > & v, const T & value)` [related]

Add a value to all elements of a vector.

Definition at line 1192 of file vector.h.

```

1193 {
1194     Vector<dim,T> res;
1195     for(size_t i = 0 ; i < dim ; ++i)
1196         res[i] = v[i] + value;
1197     return res;
1198 }
```

17.68.4.28 `template<size_t dim, typename T> Vector< dim, T > operator-(const T & value, const Vector< dim, T > & v)` [related]

Equivalent to subtracting a vector with all component the same to another one.

Definition at line 1235 of file vector.h.

```

1236 {
1237     Vector<dim,T> res;
1238     for(size_t i = 0 ; i < dim ; ++i)
1239         res[i] = value - v[i];
1240     return res;
1241 }
```

17.68.4.29 `template<size_t dim, typename T> Vector< dim, T > operator-(const Vector< dim, T > & v, const T & value)` [related]

Subtract a value to all elements of a vector.

Definition at line 1220 of file vector.h.

```

1221 {
1222     Vector<dim,T> res;
1223     for(size_t i = 0 ; i < dim ; ++i)
1224         res[i] = v[i] - value;
1225     return res;
1226 }
```

17.68.4.30 `template<class T> Vector< 3, T > operator^(const Vector< 3, T > & v1, const Vector< 3, T > & v2)` [related]

Cross product $v_1 \times v_2$ (French notation).

Definition at line 762 of file vector.h.

```

763 {
764     Vector<3,T> ans;
765     ans[0] = v1[1]*v2[2] - v1[2]*v2[1];
766     ans[1] = v1[2]*v2[0] - v1[0]*v2[2];
767     ans[2] = v1[0]*v2[1] - v1[1]*v2[0];
768     return ans;
769 }

```

17.68.4.31 `template<class T> T operator^ (const Vector< 1, T> &v1, const Vector< 1, T> &v2)` [[related](#)]

Cross product $v_1 \times v_2$ (French notation).

Definition at line 740 of file vector.h.

```

741 {
742     return 0;
743 }

```

17.68.4.32 `template<class T> T operator^ (const Vector< 2, T> &v1, const Vector< 2, T> &v2)` [[related](#)]

Cross product $v_1 \times v_2$ (French notation).

Definition at line 728 of file vector.h.

```

729 {
730     return ((v1[0] * v2[1]) -
731            (v1[1] * v2[0]));
732 }

```

17.68.4.33 `template<typename T> util::Vector< 3, T> orthogonal (const util::Vector< 3, T> &v)` [[related](#)]

Find a vector orthogonal to v.

Definition at line 982 of file vector.h.

References `util::Vector< dim, T>::x()`, `util::Vector< dim, T>::y()`, and `util::Vector< dim, T>::z()`.

```

983 {
984     const double ratio = 1-1e-8;
985     if ((std::abs(v.y()) >= ratio*std::abs(v.x())) && (std::abs(v.z()) >= ratio*std::abs(v.x())))
986         return util::Vector<3,T>(0, -v.z(), v.y());
987     else
988         if ((std::abs(v.x()) >= ratio*std::abs(v.y())) && (std::abs(v.z()) >= ratio*std::abs(v.y())))
989             return util::Vector<3,T>(-v.z(), 0, v.x());
990     else
991         return util::Vector<3,T>(-v.y(), v.x(), 0);
992 }

```

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

- vvelib/util/[vector.h](#)

17.69 algorithms::TriangleSurface::Vertex Struct Reference

Type of a vertex, i.e.

```
#include <triangle_growth.h>
```

Public Attributes

- `int` `id`
Id of the junction.
- `Point3d` `normal`
Normal.
- `Point3d` `pos`
3D position

17.69.1 Detailed Description

Type of a vertex, i.e. a junction

Definition at line 48 of file `triangle_growth.h`.

17.69.2 Member Data Documentation

17.69.2.1 `int` algorithms::TriangleSurface::Vertex::id

Id of the junction.

Definition at line 55 of file `triangle_growth.h`.

17.69.2.2 `Point3d` algorithms::TriangleSurface::Vertex::normal

Normal.

Definition at line 53 of file `triangle_growth.h`.

17.69.2.3 `Point3d` algorithms::TriangleSurface::Vertex::pos

3D position

Definition at line 51 of file `triangle_growth.h`.

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

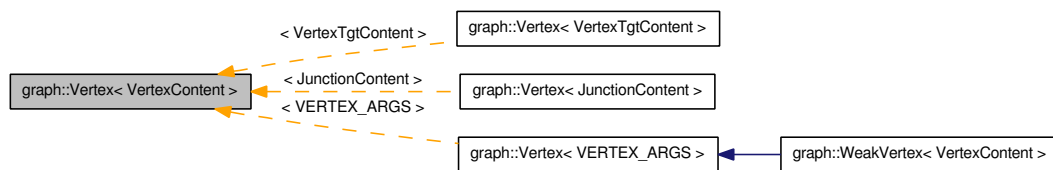
- `vvelib/algorithms/triangle_growth.h`

17.70 graph::Vertex< VertexContent > Class Template Reference

[Vertex](#) of a vv graph.

```
#include <graph/vertex.h>
```

Inheritance diagram for graph::Vertex< VertexContent >:



Public Types

- typedef VertexContent [content_t](#)
Type of the content of the vertex.
- typedef [CountedContent](#)< VertexContent > [counted_content_t](#)
Type of the reference-counted content.
- typedef [vertex_identity_t](#) identity_t
Type of the identifier of the vertex.
- typedef [content_t](#) * [pointer](#)
Type of the equivalent pointer.
- typedef [counted_content_t](#) * [real_pointer](#)
Pointer to the reference-counted content.
- typedef [content_t](#) & **reference**
- typedef [WeakVertex](#)< VERTEX_ARGS > **weak_ref_t**

Public Member Functions

- [real_pointer](#) [acquire](#) ()
Return the content, and reset the value in the vertex.
- [real_pointer](#) [content](#) () const
Return the reference-counted object.
- [identity_t](#) [id](#) () const
Return the identifier of a vertex.

- `bool isNull () const`
Test if a vertex is a null vertex.
- `bool isWeakRef () const`
Return `true` if the current object hold a weak reference on a vertex.
- `size_t num () const`
Return a number unique to each vertex, globally.
- `operator bool () const`
Convert a vertex to `true` if it is not null.
- `template<typename T1 >`
`bool operator!= (const Vertex< T1 > &other) const`
Two vertices of different types cannot be equal.
- `bool operator!= (const Vertex &other) const`
Comparison operators.
- `reference operator* () const`
Access to the data.
- `pointer operator-> () const`
Access to the data.
- `template<typename R >`
`const R & operator->* (R VertexContent::*ptr) const`
Constant access to the data via pointer to member.
- `template<typename R >`
`R & operator->* (R VertexContent::*ptr)`
Access to the data via pointer to member.
- `bool operator< (const Vertex &other) const`
Comparison operators.
- `bool operator<= (const Vertex &other) const`
Comparison operators.
- `Vertex & operator= (const weak_ref_t &other)`
- `Vertex & operator= (const identity_t &id)`
- `Vertex & operator= (const Vertex &other)`
Change the vertex held by the current object.
- `template<typename T1 >`
`bool operator== (const Vertex< T1 > &) const`

Two vertices of different types cannot be equal.

- `bool operator== (const Vertex &other) const`
Comparison operators.
- `bool operator> (const Vertex &other) const`
Comparison operators.
- `bool operator>= (const Vertex &other) const`
Comparison operators.
- `bool serialize (storage::VVEStorage &)`
Serialization method.
- `Vertex (const weak_ref_t &w)`
Construct a strong reference from a weak one.
- `Vertex (real_pointer content)`
Create a vertex using a counted content.
- `Vertex (const Vertex ©)`
Copy a vertex.
- `Vertex (identity_t id)`
Creates a reference on the vertex of identifier `id`.
- `Vertex (int i)`
Create an empty vertex (argument really should be 0).
- `Vertex ()`
Creates a new vertex with a new content.
- `weak_ref_t weakRef () const`
Construct a weak reference on the current vertex.
- `~Vertex ()`
Desctructor.

Public Attributes

- `void * cache`
Cache information for iteration optimization.
- `identity_t cache_source`
Label of the reference vertex for the cache information.

Static Public Attributes

- static [Vertex](#) `null`

Null vertex.

Protected Member Functions

- void [release](#) ()

Release the current pointer.

Protected Attributes

- [real_pointer_content](#)

Content of the vertex.

Friends

- class `WeakVertex< VERTEX_ARGS >`

17.70.1 Detailed Description

`template<typename VertexContent> class graph::Vertex< VertexContent >`

[Vertex](#) of a vv graph. The vertexes handle their associated data using a reference counting scheme. As such, they can be used as smart pointers. They are also comparable (`<`, `>`, `==`, `!=`), which allow for use in any sorted structure and hashable for use in any hash table-based structure.

They also all have a unique identifier. This identifier can be used to retrieve a weak reference on the data.

Warning

A weak reference won't keep the data alive! So be careful when you use those. You can test if a vertex hold a weak reference using the [Vertex<VertexContent>::isWeakRef\(\)](#) method.

Definition at line 116 of file vertex.h.

17.70.2 Member Typedef Documentation

17.70.2.1 `template<typename VertexContent> typedef VertexContent graph::Vertex< VertexContent >::content_t`

Type of the content of the vertex.

Reimplemented in [graph::WeakVertex< VertexContent >](#).

Definition at line 163 of file `vertex.h`.

17.70.2.2 `template<typename VertexContent> typedef CountedContent<VertexContent> graph::Vertex< VertexContent >::counted_content_t`

Type of the reference-counted content.

Useful to share with other smart-pointers in VVE

Definition at line 126 of file `vertex.h`.

17.70.2.3 `template<typename VertexContent> typedef vertex_identity_t graph::Vertex< VertexContent >::identity_t`

Type of the identifier of the vertex.

Reimplemented in [graph::WeakVertex< VertexContent >](#).

Definition at line 158 of file `vertex.h`.

17.70.2.4 `template<typename VertexContent> typedef content_t* graph::Vertex< VertexContent >::pointer`

Type of the equivalent pointer.

Reimplemented in [graph::WeakVertex< VertexContent >](#).

Definition at line 170 of file `vertex.h`.

17.70.2.5 `template<typename VertexContent> typedef counted_content_t* graph::Vertex< VertexContent >::real_pointer`

Pointer to the reference-counted content.

Useful to share with other smart-pointers in VVE

Definition at line 152 of file `vertex.h`.

17.70.3 Constructor & Destructor Documentation

17.70.3.1 `template<TEMPLATE_VERTEX > graph::Vertex<TEMPLATE_VERTEX >::Vertex () [inline]`

Creates a new vertex with a new content.

Example:

```
struct VertexContent { int a; }
// ...
Vertex<VertexContent> v;
v->a = 10;
```

Definition at line 623 of file vertex.h.

References `graph::Vertex< VertexContent >::_content`.

```
624      : _content(0)
625 #ifndef VVERTEX_NO_CACHE
626      , cache_source(0)
627      , cache(0)
628 #endif
629 {
630 #ifdef USE_ALLOCATOR
631     _content = alloc.allocate(1);
632     ::new(_content) counted_content_t();
633 #elif defined(DEBUG_ALLOC) // defined USE_ALLOCATOR
634     try
635     {
636         _content = new counted_content_t();
637     }
638     catch(std::bad_alloc& )
639     {
640         _content = 0;
641     }
642 #else
643     _content = new counted_content_t();
644 #endif // defined USE_ALLOCATOR
645 }
```

17.70.3.2 `template<TEMPLATE_VERTEX > graph::Vertex<TEMPLATE_VERTEX >::Vertex (int i) [inline, explicit]`

Create an empty vertex (argument really should be 0).

Definition at line 648 of file vertex.h.

References `vvassert`.

```
649      : _content(0)
650 #ifndef VVERTEX_NO_CACHE
651      , cache_source(0)
652      , cache(0)
653 #endif
654 {
655     vvassert(id == 0);
656 }
```

17.70.3.3 `template<TEMPLATE_VERTEX > graph::Vertex< TEMPLATE_VERTEX >::Vertex (identity_t id) [inline, explicit]`

Creates a reference on the vertex of identifier `id`.

If `id` is 0, creates a null vertex.

Parameters

← *id* Label of the vertex to retrieve.

Warning

This function is very unsafe if used with anything but 0 as an identifier.

Example:

```
typedef Vertex<VertexContent> vertex;
vertex v;
vertex::identity_t i = v.id();
vertex vl(i);
assert(vl == v);
```

Definition at line 659 of file `vertex.h`.

References `graph::Vertex< VertexContent >::_content`, and `graph::CountedContent< Content >::count`.

```
660      : _content(reinterpret_cast<real_pointer>(id))
661 #ifndef VVERTEX_NO_CACHE
662      , cache_source(0)
663      , cache(0)
664 #endif
665 {
666     if(_content)
667     {
668         if(_content->count == 0)
669             _content = 0;
670         else
671             ++(_content->count);
672     }
673 }
```

17.70.3.4 `template<TEMPLATE_VERTEX > graph::Vertex< TEMPLATE_VERTEX >::Vertex (const Vertex< VertexContent > ©) [inline]`

Copy a vertex.

The data is not copied. The quality of the copy (i.e. weak/strong reference) is the same as the copied.

Definition at line 724 of file `vertex.h`.

References `graph::Vertex< VertexContent >::_content`, and `graph::CountedContent< Content >::count`.

```

725     : _content (copy._content)
726 #ifndef VVVERTEX_NO_CACHE
727     , cache_source (0)
728     , cache (0)
729 #endif
730 {
731     if (_content)
732     {
733         ++ (_content->count);
734     }
735 }
```

17.70.3.5 `template<TEMPLATE_VERTEX > graph::Vertex< TEMPLATE_VERTEX >::Vertex (real_pointer content) [inline]`

Create a vertex using a counted content.

The vertex add to the ownership of the content

Definition at line 676 of file vertex.h.

References `graph::Vertex< VertexContent >::_content`, and `graph::CountedContent< Content >::count`.

```

677     : _content (content)
678 #ifndef VVVERTEX_NO_CACHE
679     , cache_source (0)
680     , cache (0)
681 #endif
682 {
683     if (_content)
684         ++ (_content->count);
685 }
```

17.70.3.6 `template<typename VertexContent> graph::Vertex< VertexContent >::Vertex (const weak_ref_t & w) [explicit]`

Construct a strong reference from a weak one.

17.70.3.7 `template<TEMPLATE_VERTEX > graph::Vertex< TEMPLATE_VERTEX >::~~Vertex () [inline]`

Destructor.

Definition at line 617 of file vertex.h.

```

618     {
619         this->release();
620     }
```

17.70.4 Member Function Documentation

17.70.4.1 `template<typename VertexContent> real_pointer graph::Vertex< VertexContent >::acquire () [inline]`

Return the content, and reset the value in the vertex.

Definition at line 267 of file vertex.h.

```
267 { real_pointer p = _content; _content = 0; return p; }
```

17.70.4.2 `template<typename VertexContent> real_pointer graph::Vertex< VertexContent >::content () const [inline]`

Return the reference-counted object.

Definition at line 262 of file vertex.h.

```
262 { return _content; }
```

17.70.4.3 `template<typename VertexContent> identity_t graph::Vertex< VertexContent >::id () const [inline]`

Return the identifier of a vertex.

Definition at line 395 of file vertex.h.

Referenced by `graph::VVGraph< VertexContent, EdgeContent, compact >::arc()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::edge()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::edge()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdge()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertEdge()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdges()`, `graph::Vertex< VERTEX_ARGS >::operator!=()`, `graph::Vertex< VERTEX_ARGS >::operator<()`, `graph::Vertex< VERTEX_ARGS >::operator<=()`, `graph::Vertex< VERTEX_ARGS >::operator==()`, `graph::Vertex< VERTEX_ARGS >::operator>()`, `graph::Vertex< VERTEX_ARGS >::operator>=()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::replace()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::replace()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::spliceAfter()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::spliceBefore()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceBefore()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateEdgeCache()`.

```
395 { return (identity_t)_content; }
```

17.70.4.4 `template<typename VertexContent> bool graph::Vertex<VertexContent >::isNull () const [inline]`

Test if a vertex is a null vertex.

Reimplemented in [graph::WeakVertex<VertexContent >](#).

Definition at line 390 of file vertex.h.

Referenced by `graph::VVGraph<VertexContent, EdgeContent, compact >::anyIn()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::anyIn()`, `vvcomplex::VVComplex<MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue<ALL_COMPLEX_TEMPLATE_ARGS >)>::connectFromJunctions()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::empty()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::empty()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::iAnyIn()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iAnyIn()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::iEmpty()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iEmpty()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::iNeighbors()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iNeighbors()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::insertEdge()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertEdge()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::insertEdges()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::iValence()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iValence()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::neighbors()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::replace()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::replace()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::spliceAfter()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::spliceBefore()`, `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceBefore()`, `graph::VVGraph<VertexContent, EdgeContent, compact >::valence()`, and `graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::valence()`.

```
390 { return _content == 0; }
```


17.70.4.5 `template<typename VertexContent> bool graph::Vertex< VertexContent >::isWeakRef () const [inline]`

Return `true` if the current object hold a weak reference on a vertex.

Definition at line 416 of file vertex.h.

```
416 { return false; }
```

17.70.4.6 `template<typename VertexContent> size_t graph::Vertex< VertexContent >::num () const [inline]`

Return a number unique to each vertex, globally.

This method can be removed by defining `NO_NUMBER_VERTICES` at compile time

Definition at line 405 of file vertex.h.

Referenced by `graph::Vertex< VERTEX_ARGS >::operator<()`, `graph::Vertex< VERTEX_ARGS >::operator<=()`, `graph::Vertex< VERTEX_ARGS >::operator>()`, and `graph::Vertex< VERTEX_ARGS >::operator>=()`.

```
405 { return _content->num; }
```

17.70.4.7 `template<typename VertexContent> graph::Vertex< VertexContent >::operator bool () const [inline]`

Convert a vertex to `true` if it is not null.

Definition at line 411 of file vertex.h.

```
411 { return _content; }
```

17.70.4.8 `template<typename VertexContent> template<typename T1 > bool graph::Vertex< VertexContent >::operator!= (const Vertex< T1 > & other) const [inline]`

Two vertices of different types cannot be equal.

Definition at line 331 of file vertex.h.

```
331 { return true; }
```

17.70.4.9 `template<typename VertexContent> bool graph::Vertex< VertexContent >::operator!= (const Vertex< VertexContent > & other) const [inline]`

Comparison operators.

Note

the comparisons work on the identifiers, not the contents!

Definition at line 325 of file vertex.h.

```
325 { return id() != other.id(); }
```

17.70.4.10 template<typename VertexContent> reference graph::Vertex<VertexContent >::operator* () const [inline]

Access to the data.

Warning

Do not try to access the data of the null vertex.

Definition at line 257 of file vertex.h.

```
257 { return *_content; }
```

17.70.4.11 template<typename VertexContent> pointer graph::Vertex<VertexContent >::operator-> () const [inline]

Access to the data.

Warning

Do not try to access the data of the null vertex.

Definition at line 251 of file vertex.h.

```
251 { return _content; }
```

17.70.4.12 template<typename VertexContent> template<typename R > const R& graph::Vertex< VertexContent >::operator->* (R VertexContent::* ptr) const [inline]

Constant access to the data via pointer to member.

Definition at line 281 of file vertex.h.

```
282     {
283         return _content->*ptr;
284     }
```

17.70.4.13 `template<typename VertexContent> template<typename R > R& graph::Vertex< VertexContent >::operator->*(R VertexContent::* ptr) [inline]`

Access to the data via pointer to member.

Definition at line 273 of file vertex.h.

```
274     {  
275         return _content->*ptr;  
276     }
```

17.70.4.14 `template<typename VertexContent> bool graph::Vertex< VertexContent >::operator< (const Vertex< VertexContent > & other) const [inline]`

Comparison operators.

Note

the comparisons work on the identifiers, not the contents!

Definition at line 351 of file vertex.h.

```
352     {  
353 #ifdef VERTEX_ORDER_BY_NUM  
354         return num() < other.num();  
355 #else  
356         return id() < other.id();  
357 #endif  
358     }
```

17.70.4.15 `template<typename VertexContent> bool graph::Vertex< VertexContent >::operator<= (const Vertex< VertexContent > & other) const [inline]`

Comparison operators.

Note

the comparisons work on the identifiers, not the contents!

Definition at line 378 of file vertex.h.

```
379     {  
380 #ifdef VERTEX_ORDER_BY_NUM  
381         return num() <= other.num();  
382 #else  
383         return id() <= other.id();  
384 #endif  
385     }
```

17.70.4.16 `template<TEMPLATE_VERTEX > Vertex< VERTEX_ARGS >
& graph::Vertex< TEMPLATE_VERTEX >::operator= (const
Vertex< VertexContent > & other) [inline]`

Change the vertex held by the current object.

The data is never modified by this operation. If you wish to copy the data of a vertex v1 into a vertex v2 use:

```
*v2 = *v1;
```

Definition at line 821 of file vertex.h.

References `graph::Vertex< VertexContent >::_content`.

```
822 {
823     if(_content == other._content)
824     {
825         return *this;
826     }
827     else
828         this->release();
829     _content = other._content;
830     if(_content)
831     {
832         ++(_content->count);
833     }
834     return *this;
835 }
```

17.70.4.17 `template<typename VertexContent> template<typename T1 >
bool graph::Vertex< VertexContent >::operator== (const Vertex<
T1 > &) const [inline]`

Two vertices of different types cannot be equal.

Definition at line 318 of file vertex.h.

```
318 { return false; }
```

17.70.4.18 `template<typename VertexContent> bool graph::Vertex<
VertexContent >::operator== (const Vertex< VertexContent > &
other) const [inline]`

Comparison operators.

Note

the comparisons work on the identifiers, not the contents!

Definition at line 312 of file vertex.h.

```
312 { return id() == other.id(); }
```

17.70.4.19 `template<typename VertexContent> bool graph::Vertex< VertexContent >::operator> (const Vertex< VertexContent > & other) const [inline]`

Comparison operators.

Note

the comparisons work on the identifiers, not the contents!

Definition at line 338 of file vertex.h.

```
339     {
340 #ifdef VERTEX_ORDER_BY_NUM
341     return num() > other.num();
342 #else
343     return id() > other.id();
344 #endif
345     }
```

17.70.4.20 `template<typename VertexContent> bool graph::Vertex< VertexContent >::operator>= (const Vertex< VertexContent > & other) const [inline]`

Comparison operators.

Note

the comparisons work on the identifiers, not the contents!

Definition at line 365 of file vertex.h.

```
366     {
367 #ifdef VERTEX_ORDER_BY_NUM
368     return num() >= other.num();
369 #else
370     return id() >= other.id();
371 #endif
372     }
```

17.70.4.21 `template<TEMPLATE_VERTEX > void graph::Vertex< TEMPLATE_VERTEX >::release () [inline, protected]`

Release the current pointer.

Definition at line 797 of file vertex.h.

```
798     {
799     if( _content )
800     {
```

```

801         --(_content->count);
802         if(_content->count == 0)
803         {
804 #ifdef USE_ALLOCATOR
805             _content->~counted_content_t();
806             alloc.deallocate(_content, 1);
807 #else // defined USE_ALLOCATOR
808             delete _content;
809 #endif
810         }
811     }
812 }
```

17.70.4.22 `template<typename VertexContent> bool graph::Vertex<VertexContent >::serialize (storage::VVEStorage &)`

Serialization method.

Warning

You need to include `<storage/graph.h>` to use this serialization method

17.70.4.23 `template<TEMPLATE_VERTEX > Vertex< VERTEX_ARGS >::weak_ref_t graph::Vertex< TEMPLATE_VERTEX >::weakRef() const [inline]`

Construct a weak reference on the current vertex.

Definition at line 815 of file vertex.h.

```

816     {
817         return weak_ref_t(*this);
818     }
```

17.70.5 Member Data Documentation

17.70.5.1 `template<typename VertexContent> real_pointer graph::Vertex<VertexContent >::_content [mutable, protected]`

Content of the vertex.

This member is mutable to allow for modification of constant references. This is useful as no operation on the vertex depend on this.

Definition at line 457 of file vertex.h.

Referenced by `graph::Vertex< VERTEX_ARGS >::acquire()`, `graph::Vertex< VERTEX_ARGS >::content()`, `graph::Vertex< VERTEX_ARGS >::id()`, `graph::Vertex< VERTEX_ARGS >::isNull()`, `graph::Vertex< VERTEX_ARGS >::num()`, `graph::Vertex< VERTEX_ARGS >::operator bool()`, `graph::Vertex< VERTEX_ARGS >::operator*()`, `graph::Vertex< VERTEX_ARGS >::operator->()`,

graph::Vertex< VERTEX_ARGS >::operator->*, graph::Vertex< VertexContent >::operator=(), graph::WeakVertex< VertexContent >::operator=(), graph::Vertex< VertexContent >::Vertex(), and graph::WeakVertex< VertexContent >::WeakVertex().

17.70.5.2 template<typename VertexContent> void* graph::Vertex< VertexContent >::cache [mutable]

Cache information for iteration optimization.

Warning

If you dare to modify this variable, I will haunt your dreams until you give up and leave that member alone.

Definition at line 479 of file vertex.h.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex(), graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::store_vertices(), graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateEdgeCache().

17.70.5.3 template<typename VertexContent> identity_t graph::Vertex< VertexContent >::cache_source [mutable]

Label of the reference vertex for the cache information.

Warning

You should never, on any condition, even try to read this variable, it may have no meaning whatsoever. If you modify it, expect a crash!

Definition at line 472 of file vertex.h.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex(), graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::store_vertices(), graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateEdgeCache().

17.70.5.4 `template<typename VertexContent> Vertex< VERTEX_ARGS >` `graph::Vertex< TEMPLATE_VERTEX >::null [inline, static]`

Null vertex.

Useful to return a constant reference on a vertex all the time.

Example:

```
const vertex& fct(bool cond)
{
    static vertex a;
    if(cond)
        return a;
    return vertex::null;
}
```

Definition at line 439 of file vertex.h.

Referenced by `vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::adjacentCell()`, `vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::adjacentCells()`, `graph::VVGrahp< VertexContent, EdgeContent, compact >::any()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::any_vertex1()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::any_vertex2()`, `graph::VVGrahp< VertexContent, EdgeContent, compact >::anyIn()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::anyIn()`, `graph::VVGrahp< VertexContent, EdgeContent, compact >::flagged()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::flagged()`, `graph::VVGrahp< VertexContent, EdgeContent, compact >::iAnyIn()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iAnyIn()`, `vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::interfaceWall()`, `vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::interfaceWallSpan()`, `graph::VVGrahp< VertexContent, EdgeContent, compact >::nextTo()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo()`, `graph::VVGrahp< VertexContent, EdgeContent, compact >::prevTo()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::prevTo()`, `graph::VVGrahp< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::reference()`, `graph::VVBigraph< typename cell::content_t, typename junction::content_t >::reference()`, `graph::VVGrahp< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::source()`,

graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::source(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::store_vertices(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::target(), and graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::target().

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

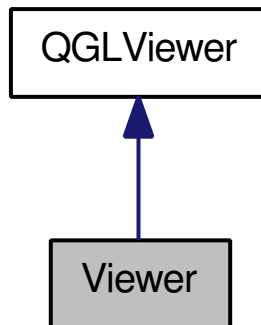
- vvelib/graph/[vertex.h](#)

17.71 Viewer Class Reference

[Viewer](#) widget

```
#include <viewer.h>
```

Inheritance diagram for Viewer:



Public Slots

- void [configOpenGL](#) ()
Open the dialog box to configure the OpenGL context.
- virtual void [initFromDOMElement](#) (const **QDomElement** &element)
Read a saved state.
- virtual void [setModel](#) (**Model** *model)
This function registers the model creation function to the viewer.
- void [updateAll](#) ()
Update first OpenGL, then the view.

Snapshot methods

- void **blockDraw** (bool block=true)

Signals

Internal signals

- void [cameraRecordingStart](#) ()
Signal sent to the GUI to start recording animations.
- void [cameraRecordingStop](#) ()

Signal sent to the GUI to top recording animations.

- void [createView](#) (QWidget *w, const QGLWidget *shareWidget)
Signal sent to request a new viewer.
- void [createView](#) (QLayout *l, const QGLWidget *shareWidget)
Signal sent to request a new viewer.
- void [createView](#) (const QGLWidget *shareWidget)
Signal sent to request a new viewer.
- void [modelRecordingStart](#) ()
Signal sent to the GUI to start recording animations.
- void [modelRecordingStop](#) ()
Signal sent to the GUI to top recording animations.
- void [openOpenGLConfig](#) ()
Open OpenGL config dialog box.
- void [stepDrawFinished](#) (bool)
Signal emitted at the end of each simulation step (including the drawing).

Public Member Functions

- void [addToolBar](#) (Qt::ToolBarArea area, QToolBar *toolbar)
Add a toolbar to the main window of the application.
- QToolBar * [addToolBar](#) (const QString &title)
Creates a toolbar associated with the main window of the application.
- void [keyPressEvent](#) (QKeyEvent *e)
Define a new shortcut .
- QMainWindow * [mainWindow](#) ()
Returns the main window associated with the application.
- Model * [model](#) ()
Function returning a pointer onto the current model.
- void [setStatusMessage](#) (const QString &msg)
Change the status message.
- void [showEvent](#) (QShowEvent *event)
Restore the state (including position) on the show event.
- QString [statusMessage](#) () const

Get the current status message.

Internal methods

- virtual **QDomElement** [domElement](#) (const **QString** &name, **QDomDocument** &document) const
Function used to save the current state of the viewer (i.e.
- virtual **QString** [helpString](#) () const
Returns a description of the model to show on the help screen.
- void [setIndex](#) (int idx)
Set the creation index to the position in the viewer list.

Method to redefine to personalize drawing or user interaction

- virtual void [draw](#) ()
Redraw the simulation result.
- virtual void [drawWithNames](#) ()
Draw with object naming.
- virtual void [init](#) ()
Initialize the viewer with a valid OpenGL context.
- virtual void [postDraw](#) ()
Restore the original state of the OpenGL context.
- virtual void [postSelection](#) (const **QPoint** &pos)
Handle post selection actions.
- virtual void [preDraw](#) ()
Initialize the OpenGL context before drawing.

Constructors/destructors

- [Viewer](#) (**QWidget** *parent, const **QGLWidget** *shareWidget=0, int idx=-1)
Creates the object within the parent.
- [~Viewer](#) ()
Destroy the viewer properly.

Static Public Member Functions

- static void [initFormat](#) ()

Screenshot methods

- void [addViewer](#) (**QWidget** *w, const **QGLWidget** *shareWidget=0)
Create a new viewer.
- void [addViewer](#) (**QLayout** *l, const **QGLWidget** *shareWidget=0)
Create a new viewer.
- virtual void [drawLight](#) (GLenum light, float scale=1.0f)
Exposing the drawLight method .
- **QMenu** * [helpMenu](#) ()
Returns a pointer to the help menu.
- int [index](#) () const
Return the creation index, or -1 if the viewer is not registered with the main GUI.
- **QMenuBar** * [menuBar](#) (bool clear=true)
Returns a pointer to the menu bar of the application.
- void [newViewer](#) (const **QGLWidget** *shareWidget=0)
Create a new viewer.
- int [screenshotCounter](#) () const
Get the current screenshot counter.
- const **QString** & [screenshotFileName](#) () const
Get the current screenshot file name.
- const **QString** & [screenshotFormat](#) () const
Get the current screenshot format.
- int [screenshotQuality](#) ()
Get the current screenshot quality.
- bool [wasInitialized](#) () const
Returns true if this viewer has already been initialized.
- bool [openScreenshotFormatDialog](#) ()
Open the screenshot format dialog.
- void [saveScreenshot](#) (const **QString** &fileName, bool overwrite=false)
Save a screenshot.
- void [saveScreenshot](#) (bool automatic=true, bool overwrite=false)
Save a screenshot.

- void [setScreenshotCounter](#) (int counter)
Set the screenshot counter.
- void [setScreenshotFileName](#) (const **QString** &name)
Set the screenshot file name.
- void [setScreenshotFormat](#) (const **QString** &format)
Set the screenshot format.
- void [setScreenshotQuality](#) (int quality)
Set the screenshot quality.
- void [startRecordingCameraAnimation](#) ()
Start recording a movie with the various frames.
- void [startRecordingModelAnimation](#) ()
Start recording a movie with the various frames.
- void [stopRecordingCameraAnimation](#) ()
Start recording a movie with the various frames.
- void [stopRecordingModelAnimation](#) ()
Start recording a movie with the various frames.

17.71.1 Detailed Description

[Viewer](#) widget This class handle the simulation window and the life of the simulation object.

You might want to inherit this class to personalize the interactions. You might also want to look at the documentation of its base class.

If you inherit from it, you have to register the derived class using the `DEFINE_VIEWER(classname)` macro defined in `<vval.h>`.

Definition at line 30 of file `viewer.h`.

17.71.2 Constructor & Destructor Documentation

17.71.2.1 `Viewer::Viewer (QWidget *parent, const QGLWidget *shareWidget = 0, int idx = -1)`

Creates the object within the parent.

Parameters

parent Enclosing widget.

Definition at line 26 of file viewer.cpp.

```
27 : QGLViewer(parent, shareWidget)
28 , _model(0)
29 , animation(false)
30 , initialized(false)
31 , lastInitCamera(0)
32 , _index(idx)
33 , block_draw(0)
34 , minFrameInterval(0)
35 , main_window(0)
36 {
37     initObject();
38 }
```

17.71.2.2 Viewer::~Viewer ()

Destroy the viewer properly.

Definition at line 86 of file viewer.cpp.

References QGLViewer::saveStateToFile().

```
87 {
88     if(_model)
89     {
90         _model->finalizeDraw(this);
91     }
92     saveStateToFile();
93 }
```

17.71.3 Member Function Documentation

17.71.3.1 void Viewer::addToolBar (Qt::ToolBarArea *area*, QToolBar * *toolbar*)

Add a toolbar to the main window of the application.

Definition at line 321 of file viewer.cpp.

References QMainWindow::addToolBar(), and mainWindow().

```
322 {
323     mainWindow()->addToolBar(area, toolbar);
324 }
```

17.71.3.2 QToolBar * Viewer::addToolBar (const QString & *title*)

Creates a toolbar associated with the main window of the application.

Definition at line 316 of file viewer.cpp.

References QMainWindow::addToolBar(), and mainWindow().

```
317 {  
318     return mainWindow()->addToolBar(title);  
319 }
```

17.71.3.3 void Viewer::addViewer (QWidget * *w*, const QGLWidget * *shareWidget* = 0)

Create a new viewer.

The viewer created is boxed in a widget for easy replacement.

Definition at line 415 of file viewer.cpp.

References createViewer().

```
416 {  
417     emit createViewer(w, shareWidget);  
418 }
```

17.71.3.4 void Viewer::addViewer (QLayout * *l*, const QGLWidget * *shareWidget* = 0)

Create a new viewer.

The viewer created is boxed in a widget for easy replacement.

Definition at line 420 of file viewer.cpp.

References createViewer().

```
421 {  
422     emit createViewer(l, shareWidget);  
423 }
```

17.71.3.5 void Viewer::cameraRecordingStart () [signal]

Signal sent to the GUI to start recording animations.

Referenced by startRecordingCameraAnimation().

17.71.3.6 void Viewer::cameraRecordingStop () [signal]

Signal sent to the GUI to stop recording animations.

Referenced by stopRecordingCameraAnimation().

17.71.3.7 void Viewer::configOpenGL () [slot]

Open the dialog box to configure the OpenGL context.

Definition at line 362 of file viewer.cpp.

References `openOpenGLConfig()`.

```
363 {
364     emit openOpenGLConfig();
365 }
```

17.71.3.8 void Viewer::createView (QWidget * w, const QGLWidget * shareWidget) [signal]

Signal sent to request a new viewer.

17.71.3.9 void Viewer::createView (QLayout * l, const QGLWidget * shareWidget) [signal]

Signal sent to request a new viewer.

17.71.3.10 void Viewer::createView (const QGLWidget * shareWidget) [signal]

Signal sent to request a new viewer.

Referenced by `addViewer()`, and `newViewer()`.

17.71.3.11 QDomElement Viewer::domElement (const QString & name, QDomDocument & document) const [virtual]

Function used to save the current state of the viewer (i.e.

camera, ...)

Reimplemented from **QGLViewer**.

Definition at line 181 of file viewer.cpp.

References `QGLFormat::accum()`, `QGLFormat::accumBufferSize()`, `QGLFormat::alpha()`, `QGLFormat::alphaBufferSize()`, `QGLFormat::blueBufferSize()`, `QDomDocument::createElement()`, `QGLFormat::defaultFormat()`, `QGLFormat::depth()`, `QGLFormat::depthBufferSize()`, `QGLFormat::directRendering()`, `QGLFormat::doubleBuffer()`, `QGLFormat::greenBufferSize()`, `QGLFormat::hasOverlay()`, `QGLFormat::redBufferSize()`, `QGLFormat::rgba()`, `QGLFormat::sampleBuffers()`, `QGLFormat::samples()`, `QDomElement::setAttribute()`, `QGLFormat::stencil()`, `QGLFormat::stencilBufferSize()`, and `QGLFormat::stereo()`.

```
182 {
183     // Creates a custom node for a light
184     QDomElement de = document.createElement("GLFormat");
185     QGLFormat def = QGLFormat::defaultFormat();
186     de.setAttribute("accum", (def.accum()?"yes":"no"));
187     if(def.accum() && def.accumBufferSize() != -1)
```

```

188     de.setAttribute("accumBufferSize", max(0, def.accumBufferSize()));
189     de.setAttribute("alpha", (def.alpha()? "yes": "no"));
190     if(def.alpha() && def.alphaBufferSize() != -1)
191         de.setAttribute("alphaBufferSize", max(0, def.alphaBufferSize()));
192     de.setAttribute("depth", (def.depth()? "yes": "no"));
193     if(def.depth() && def.depthBufferSize() != -1)
194         de.setAttribute("depthBufferSize", max(0, def.depthBufferSize()));
195     de.setAttribute("doubleBuffer", (def.doubleBuffer()? "yes": "no"));
196     de.setAttribute("directRendering", (def.directRendering()? "yes": "no"));
197     de.setAttribute("hasOverlay", (def.hasOverlay()? "yes": "no"));
198     de.setAttribute("rgba", (def.rgba()? "yes": "no"));
199     if(def.rgba())
200     {
201         if(def.redBufferSize() != -1)
202             de.setAttribute("redBufferSize", max(0, def.redBufferSize()));
203         if(def.greenBufferSize() != -1)
204             de.setAttribute("greenBufferSize", max(0, def.greenBufferSize()));
205         if(def.blueBufferSize() != -1)
206             de.setAttribute("blueBufferSize", max(0, def.blueBufferSize()));
207     }
208     de.setAttribute("sampleBuffers", (def.sampleBuffers()? "yes": "no"));
209     if(def.sampleBuffers() && def.samples() != -1)
210         de.setAttribute("samples", max(0, def.samples()));
211     de.setAttribute("stereo", (def.stereo()? "yes": "no"));
212     de.setAttribute("stencil", (def.stencil()? "yes": "no"));
213     if(def.stencil() && def.stencilBufferSize() != -1)
214         de.setAttribute("stencilBufferSize", max(0, def.stencilBufferSize()));
215
216     // Get default state domElement and append custom node
217     QDomElement res = QGLViewer::domElement(name, document);
218     res.appendChild(de);
219     return res;
220 }

```

17.71.3.12 virtual void Viewer::draw () [inline, virtual]

Redraw the simulation result.

See also

[Model::draw\(Viewer*\)](#)

Reimplemented from **QGLViewer**.

Definition at line 344 of file viewer.h.

```

345     { if( _model and !block_draw ) _model->draw(this); }

```

17.71.3.13 virtual void Viewer::drawLight (GLenum *light*, float *scale* = 1.0f) [inline, virtual]

Exposing the drawLight method .

..

See also

QGLViewer::drawLight

Definition at line 121 of file viewer.h.

```
121 { QGLViewer::drawLight(light, scale); }
```

17.71.3.14 virtual void Viewer::drawWithNames () [inline, virtual]

Draw with object naming.

Used for object selection.

See also

[Model::drawWithNames\(Viewer*\)](#)

Reimplemented from **QGLViewer**.

Definition at line 365 of file viewer.h.

```
366 { if (_model) _model->drawWithNames(this); }
```

17.71.3.15 QMenu * Viewer::helpMenu ()

Returns a pointer to the help menu.

This is useful for two purposes: 1 - Adding your own help elements 2 - Placing other menus before the help

Definition at line 351 of file viewer.cpp.

References [QVariant::isValid\(\)](#), [mainWindow\(\)](#), and [QVariant::value\(\)](#).

```
352 {  
353     QVariant hm = mainWindow()->property("helpMenu");  
354     if(hm.isValid())  
355     {  
356         QMenu* menu = hm.value<QMenu*>();  
357         return menu;  
358     }  
359     return 0;  
360 }
```

17.71.3.16 virtual QString Viewer::helpString () const [inline, virtual]

Returns a description of the model to show on the help screen.

See also

[Model::helpString\(\)](#)

Reimplemented from **QGLViewer**.

Definition at line 69 of file viewer.h.

```
70 {
71     if(_model) return _model->helpString();
72     else return QString(tr("Generic VV model"));
73 }
```

17.71.3.17 int Viewer::index () const [inline]

Return the creation index, or -1 if the viewer is not registered with the main GUI.

This number can be used to identified what viewer is suppose to do what.

Definition at line 151 of file viewer.h.

Referenced by keyPressEvent().

```
151 { return _index; }
```

17.71.3.18 void Viewer::init () [virtual]

Initialize the viewer with a valid OpenGL context.

See also

[Model::initDraw\(Viewer*\)](#)

Reimplemented from **QGLViewer**.

Definition at line 95 of file viewer.cpp.

Referenced by setModel().

```
96 {
97     QGLViewer::init();
98     initCamera();
99     if(_model)
100     {
101         _model->initDraw( this );
102         //showEntireScene();
103     }
104 }
```

17.71.3.19 void Viewer::initFromDOMELEMENT (const QDomElement & element) [virtual, slot]

Read a saved state.

Reimplemented from **QGLViewer**.

Definition at line 121 of file viewer.cpp.

References QGLFormat::accum(), QGLFormat::alpha(), QDomElement::attribute(), QGLFormat::defaultFormat(), QGLFormat::depth(), QDomElement::hasAttribute(), QGLFormat::rgba(), QGLFormat::sampleBuffers(), QGLFormat::setAccumBufferSize(), QGLFormat::setAlpha(), QGLFormat::setAlphaBufferSize(), QGLFormat::setBlueBufferSize(), QGLFormat::setDefaultFormat(), QGLFormat::setDepth(), QGLFormat::setDepthBufferSize(), QGLFormat::setDirectRendering(), QGLFormat::setDoubleBuffer(), QGLFormat::setGreenBufferSize(), QGLFormat::setOverlay(), QGLFormat::setRedBufferSize(), QGLFormat::setRgba(), QGLFormat::setSampleBuffers(), QGLFormat::setSamples(), QGLFormat::setStencil(), QGLFormat::setStencilBufferSize(), QGLFormat::setStereo(), QGLFormat::stencil(), and QDomElement::tagName().

```

122 {
123     // Restore standard state
124     QGLViewer::initFromDOMELEMENT(element);
125
126     QGLFormat def = QGLFormat::defaultFormat();
127
128     QDomElement child=element.firstChild().toElement();
129     while (!child.isNull())
130     {
131         if (child.tagName() == "GLFormat")
132         {
133             if (child.hasAttribute("alpha"))
134                 def.setAlpha(child.attribute("alpha").toLower() == "yes");
135             if (child.hasAttribute("depth"))
136                 def.setDepth(child.attribute("depth").toLower() == "yes");
137             if (child.hasAttribute("rgba"))
138                 def.setRgba(child.attribute("rgba").toLower() == "yes");
139             if (child.hasAttribute("stereo"))
140                 def.setStereo(child.attribute("stereo").toLower() == "yes");
141             if (child.hasAttribute("stencil"))
142                 def.setStencil(child.attribute("stencil").toLower() == "yes");
143             if (child.hasAttribute("doubleBuffer"))
144                 def.setDoubleBuffer(child.attribute("doubleBuffer").toLower() == "yes");
145             if (child.hasAttribute("sampleBuffers"))
146                 def.setSampleBuffers(child.attribute("sampleBuffers").toLower() == "yes");
147         }
148         if (child.hasAttribute("directRendering"))
149             def.setDirectRendering(child.attribute("directRendering").toLower() == "yes");
150         if (child.hasAttribute("hasOverlay"))
151             def.setOverlay(child.attribute("hasOverlay").toLower() == "yes");
152         if (def.accum())
153             if (child.hasAttribute("accumBufferSize"))
154                 def.setAccumBufferSize(child.attribute("accumBufferSize").toInt());
155         if (def.alpha())
156             if (child.hasAttribute("alphaBufferSize"))
157                 def.setAlphaBufferSize(child.attribute("alphaBufferSize").toInt());

```

```

157         if(def.depth())
158             if (child.hasAttribute("depthBufferSize"))
159                 def.setDepthBufferSize(child.attribute("depthBufferSize").toInt());
160         if(def.sampleBuffers())
161             if (child.hasAttribute("samples"))
162                 def.setSamples(child.attribute("samples").toInt());
163         if(def.stencil())
164             if (child.hasAttribute("stencilBufferSize"))
165                 def.setStencilBufferSize(child.attribute("stencilBufferSize").toInt());

166         if(def.rgb())
167         {
168             if (child.hasAttribute("redBufferSize"))
169                 def.setRedBufferSize(child.attribute("redBufferSize").toInt());
170             if (child.hasAttribute("greenBufferSize"))
171                 def.setGreenBufferSize(child.attribute("greenBufferSize").toInt());
172             if (child.hasAttribute("blueBufferSize"))
173                 def.setBlueBufferSize(child.attribute("blueBufferSize").toInt());
174         }
175     }
176     child = child.nextSibling().toElement();
177 }
178 QGLFormat::setDefaultFormat(def);
179 }

```

17.71.3.20 void Viewer::keyPressEvent (QKeyEvent * e)

Define a new shortcut .

..

Reimplemented from **QGLViewer**.

Definition at line 393 of file viewer.cpp.

References [index\(\)](#), [QKeyEvent::key\(\)](#), and [QGLViewer::showEntireScene\(\)](#).

```

394 {
395     switch(e->key())
396     {
397         case Qt::Key_F:
398             if(index() < 0 and e->modifiers() == (Qt::AltModifier | Qt::ControlModifier))
399             {
400                 showEntireScene();
401                 return;
402             }
403             break;
404             default:
405                 break;
406     }
407     QGLViewer::keyPressEvent(e);
408 }

```

17.71.3.21 QMainWindow * Viewer::mainWindow ()

Returns the main window associated with the application.

Definition at line 326 of file viewer.cpp.

Referenced by addToolBar(), helpMenu(), and menuBar().

```
327 {  
328     return main_window;  
329 }
```

17.71.3.22 QMenuBar * Viewer::menuBar (bool *clear* = true)

Returns a pointer to the menu bar of the application.

Parameters

clear If true, remove any element from the menu bar but the help.

Definition at line 331 of file viewer.cpp.

References QVariant::isValid(), mainWindow(), and QVariant::value().

```
332 {  
333     const char* property_name;  
334     if (clear)  
335     {  
336         property_name = "menuBar";  
337     }  
338     else  
339     {  
340         property_name = "existingMenuBar";  
341     }  
342     QVariant mb = mainWindow()->property(property_name);  
343     if (mb.isValid())  
344     {  
345         QMenuBar* bar = mb.value<QMenuBar*>();  
346         return bar;  
347     }  
348     return 0;  
349 }
```

17.71.3.23 Model* Viewer::model () [inline]

Function returning a pointer onto the current model.

Definition at line 55 of file viewer.h.

```
55 { return _model; }
```

17.71.3.24 void Viewer::modelRecordingStart () [signal]

Signal sent to the GUI to start recording animations.

Referenced by startRecordingModelAnimation().

17.71.3.25 void Viewer::modelRecordingStop () [signal]

Signal sent to the GUI to stop recording animations.

Referenced by stopRecordingModelAnimation().

17.71.3.26 void Viewer::newViewer (const QGLWidget * shareWidget = 0)

Create a new viewer.

The viewer created is boxed in a widget for easy replacement.

Definition at line 410 of file viewer.cpp.

References createViewer().

```
411 {  
412     emit createViewer(shareWidget);  
413 }
```

17.71.3.27 void Viewer::openOpenGLConfig () [signal]

Open OpenGL config dialog box.

Referenced by configOpenGL().

17.71.3.28 bool Viewer::openScreenshotFormatDialog () [inline, slot]

Open the screenshot format dialog.

See also

QGLViewer::openSnapshotFormatDialog

Definition at line 248 of file viewer.h.

References QGLViewer::openSnapshotFormatDialog().

```
249     { return QGLViewer::openSnapshotFormatDialog(); }
```

17.71.3.29 void Viewer::postDraw () [virtual]

Restore the original state of the OpenGL context.

See also

[Model::postDraw\(Viewer*\)](#)

Reimplemented from QGLViewer.

Definition at line 114 of file viewer.cpp.


```
115 {  
116     if(_model and !block_draw)  
117         _model->postDraw(this);  
118     QGLViewer::postDraw();  
119 }
```

17.71.3.30 virtual void Viewer::postSelection (const QPoint & pos) [inline, virtual]

Handle post selection actions.

Reimplemented from **QGLViewer**.

Definition at line 371 of file viewer.h.

```
372     { if(_model) _model->postSelection(pos, this); }
```

17.71.3.31 virtual void Viewer::preDraw () [inline, virtual]

Initialize the OpenGL context before drawing.

See also

[Model::preDraw\(Viewer*\)](#)

Reimplemented from **QGLViewer**.

Definition at line 337 of file viewer.h.

```
338     { QGLViewer::preDraw(); if( _model and !block_draw ) _model->preDraw(this); }
```

17.71.3.32 void Viewer::saveScreenshot (const QString & fileName, bool overwrite = false) [inline, slot]

Save a screenshot.

See also

QGLViewer::saveSnapshot(const QString&, bool)

Definition at line 208 of file viewer.h.

References **QGLViewer::saveSnapshot()**.

```
209     { QGLViewer::saveSnapshot(fileName, overwrite); }
```

17.71.3.33 void Viewer::saveScreenshot (bool *automatic* = true, bool *overwrite* = false) [inline, slot]

Save a screenshot.

See also

QGLViewer::saveSnapshot(bool,bool)

Definition at line 200 of file viewer.h.

References QGLViewer::saveSnapshot().

```
201    { QGLViewer::saveSnapshot (automatic, overwrite); }
```

17.71.3.34 int Viewer::screenshotCounter () const [inline]

Get the current screenshot counter.

See also

QGLViewer::snapshotCounter

Definition at line 105 of file viewer.h.

References QGLViewer::snapshotCounter().

```
106    { return QGLViewer::snapshotCounter(); }
```

17.71.3.35 const QString& Viewer::screenshotFileName () const [inline]

Get the current screenshot file name.

See also

QGLViewer::snapshotFileName

Definition at line 90 of file viewer.h.

References QGLViewer::snapshotFileName().

```
91    { return QGLViewer::snapshotFileName(); }
```

17.71.3.36 const QString& Viewer::screenshotFormat () const [inline]

Get the current screenshot format.

See also

QGLViewer::snapshotFormat

Definition at line 97 of file viewer.h.

References `QGLViewer::snapshotFormat()`.

```
98 { return QGLViewer::snapshotFormat (); }
```

17.71.3.37 int Viewer::screenshotQuality () [inline]

Get the current screenshot quality.

See also

QGLViewer::snapshotQuality

Definition at line 113 of file viewer.h.

References `QGLViewer::snapshotQuality()`.

```
114 { return QGLViewer::snapshotQuality(); }
```

17.71.3.38 void Viewer::setIndex (int idx) [inline]

Set the creation index to the position in the viewer list.

Definition at line 78 of file viewer.h.

```
78 { _index = idx; }
```

17.71.3.39 void Viewer::setModel (Model * model) [virtual, slot]

This function registers the model creation function to the viewer.

Definition at line 106 of file viewer.cpp.

References `init()`, and `QGLViewer::updateGL()`.

```
107 {  
108     this->_model = m;  
109     if(initialized) this->init();  
110     updateGL();  
111     update();  
112 }
```

17.71.3.40 void Viewer::setScreenshotCounter (int *counter*) [inline, slot]

Set the screenshot counter.

See also

QGLViewer::setSnapshotCounter

Definition at line 232 of file viewer.h.

References QGLViewer::setSnapshotCounter().

```
233 { QGLViewer::setSnapshotCounter(counter); }
```

17.71.3.41 void Viewer::setScreenshotFileName (const QString & *name*) [inline, slot]

Set the screenshot file name.

See also

QGLViewer::setSnapshotFileName

Definition at line 216 of file viewer.h.

References QGLViewer::setSnapshotFileName().

```
217 { QGLViewer::setSnapshotFileName(name); }
```

17.71.3.42 void Viewer::setScreenshotFormat (const QString & *format*) [inline, slot]

Set the screenshot format.

See also

QGLViewer::setSnapshotFormat

Definition at line 224 of file viewer.h.

References QGLViewer::setSnapshotFormat().

```
225 { QGLViewer::setSnapshotFormat(format); }
```

17.71.3.43 void Viewer::setScreenshotQuality (int *quality*) [inline, slot]

Set the screenshot quality.

See also

QGLViewer::setSnapshotQuality

Definition at line 240 of file viewer.h.

References QGLViewer::setSnapshotQuality().

```
241 { QGLViewer::setSnapshotQuality(quality); }
```

17.71.3.44 void Viewer::setStatusMessage (const QString & *msg*)

Change the status message.

On the gui, display the message on the status bar. On batch mode, prints the message to the console.

Definition at line 304 of file viewer.cpp.

```
305 {  
306     bool success = parent()->setProperty("statusMessage", msg);  
307     assert(success);  
308 }
```

17.71.3.45 void Viewer::showEvent (QShowEvent * *event*)

Restore the state (including position) on the show event.

Definition at line 283 of file viewer.cpp.

References QGLViewer::camera(), and QGLViewer::restoreStateFromFile().

```
284 {  
285     QGLViewer::showEvent(event);  
286     if(!initialized)  
287     {  
288         restoreStateFromFile();  
289         lastInitCamera = camera();  
290         initialized = true;  
291     }  
292 }
```

17.71.3.46 void Viewer::startRecordingCameraAnimation () [slot]

Start recording a movie with the various frames.

Definition at line 373 of file viewer.cpp.

References cameraRecordingStart().

```
374 {  
375     emit cameraRecordingStart();  
376 }
```

17.71.3.47 void Viewer::startRecordingModelAnimation () [slot]

Start recording a movie with the various frames.

Definition at line 383 of file viewer.cpp.

References modelRecordingStart().

```
384 {  
385     emit modelRecordingStart();  
386 }
```

17.71.3.48 QString Viewer::statusMessage () const

Get the current status message.

Definition at line 310 of file viewer.cpp.

```
311 {  
312     QString result = parent()->property("statusMessage").toString();  
313     return result;  
314 }
```

17.71.3.49 void Viewer::stepDrawFinished (bool) [signal]

Signal emitted at the end of each simulation step (including the drawing).

17.71.3.50 void Viewer::stopRecordingCameraAnimation () [slot]

Start recording a movie with the various frames.

Definition at line 378 of file viewer.cpp.

References cameraRecordingStop().

```
379 {  
380     emit cameraRecordingStop();  
381 }
```

17.71.3.51 void Viewer::stopRecordingModelAnimation () [slot]

Start recording a movie with the various frames.

Definition at line 388 of file viewer.cpp.

References modelRecordingStop().

```
389 {  
390     emit modelRecordingStop();  
391 }
```

17.71.3.52 void Viewer::updateAll() [slot]

Update first OpenGL, then the view.

Definition at line 367 of file viewer.cpp.

References QGLViewer::updateGL().

```
368 {  
369     updateGL();  
370     update();  
371 }
```

17.71.3.53 bool Viewer::wasInitialized() const [inline]

Returns true if this viewer has already been initialized.

Definition at line 143 of file viewer.h.

```
143 { return initialized; }
```

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

- vvelib/viewer.h
- vvelib/viewer.cpp

17.72 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact > Class Template Reference

Class representing a VV graph.

```
#include <graph/vvbigraph.h>
```

Inheritance diagram for graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >:



Classes

- struct [search_result_t](#)
Type of the result of the search for a vertex in a neighborhood.
- struct [single_neighborhood_t](#)
Type of the neighborhood of a vertex.

Public Types

- typedef std::unordered_map< [vertex1_t](#), typename neighborhood1_t::iterator > [lookup1_t](#)
Type of the lookup hash map for vertices of type 1.
- typedef std::unordered_map< [vertex2_t](#), typename neighborhood2_t::iterator > [lookup2_t](#)
Type of the lookup hash map for vertices of type 2.
- typedef std::list< std::pair< [vertex1_t](#), [single_neighborhood1_t](#) > > [neighborhood1_t](#)
Type of the list of vertices of type 1, together with their neighborhood.
- typedef neighborhood1_t::value_type [neighborhood1_value_type](#)
Shortcut for the value_type of the neighborhood for a vertex of type 1.
- typedef std::list< std::pair< [vertex2_t](#), [single_neighborhood2_t](#) > > [neighborhood2_t](#)
Type of the list of vertices of type 2, together with their neighborhood.
- typedef neighborhood2_t::value_type [neighborhood2_value_type](#)
Shortcut for the value_type of the neighborhood for a vertex of type 2.

- typedef neighborhood1_t::size_type [size_type](#)

Type of a size.

Iterators

- typedef [util::CircIterator](#)< [neighbor_iterator1](#) > [circ_neighbor_iterator1](#)
Iterator used to iterate over the neighbors of a vertex of type 1, but specifying the starting point.
- typedef [util::range](#)< [circ_neighbor_iterator1](#) > [circ_neighbor_iterator1_range](#)
Range of circular iterators for the neighbors of a vertex of type 1.
- typedef [util::CircIterator](#)< [neighbor_iterator2](#) > [circ_neighbor_iterator2](#)
Iterator used to iterate over the neighbors of a vertex of type 2, but specifying the starting point.
- typedef [util::range](#)< [circ_neighbor_iterator2](#) > [circ_neighbor_iterator2_range](#)
Range of circular iterators for the neighbors of a vertex of type 2.
- typedef [util::CircIterator](#)< [const_neighbor_iterator1](#) > [const_circ_neighbor_iterator1](#)
Iterator used to iterate over the neighbors of a vertex of type 1, but specifying the starting point.
- typedef [util::range](#)< [const_circ_neighbor_iterator1](#) > [const_circ_neighbor_iterator1_range](#)
Range of circular iterators for the neighbors of a vertex of type 1.
- typedef [util::CircIterator](#)< [const_neighbor_iterator2](#) > [const_circ_neighbor_iterator2](#)
Iterator used to iterate over the neighbors of a vertex of type 2, but specifying the starting point.
- typedef [util::range](#)< [const_circ_neighbor_iterator2](#) > [const_circ_neighbor_iterator2_range](#)
Range of circular iterators for the neighbors of a vertex of type 2.
- typedef [util::SelectMemberIterator](#)< typename neighborhood1_t::const_iterator, const [vertex1_t](#), &neighborhood1_value_type::first > [const_iterator1](#)
Constant iterator on the vertices of type 1.
- typedef [util::SelectMemberIterator](#)< typename neighborhood2_t::const_iterator, const [vertex2_t](#), &neighborhood2_value_type::first > [const_iterator2](#)
Constant iterator on the vertices of type 1.

- typedef `util::SelectMemberIterator`< typename `edge_list1_t::const_iterator`,
const `vertex2_t`,&`neighbor1_t::target` > `const_neighbor_iterator1`
Constant iterator on the neighbors of a vertex of type 1.
- typedef `util::range`< `const_neighbor_iterator1` > `const_neighbor_iterator1_range`
Constant range of the neighbors of a vertex of type 1.
- typedef `util::SelectMemberIterator`< typename `edge_list2_t::const_iterator`,
const `vertex1_t`,&`neighbor2_t::target` > `const_neighbor_iterator2`
Constant iterator on the neighbors of a vertex of type 2.
- typedef `util::range`< `const_neighbor_iterator2` > `const_neighbor_iterator2_range`
Constant range of the neighbors of a vertex of type 2.
- typedef `util::range`< `const_iterator1` > `const_range_vertex1`
Range of vertex of type 1.
- typedef `util::range`< `const_iterator2` > `const_range_vertex2`
Range of vertex of type 2.
- typedef `in_edges1_t::const_iterator` `ineighbor_iterator1`
Iterator on the incoming neighbors of a vertex of type 1.
- typedef `util::range`< `ineighbor_iterator1` > `ineighbor_iterator1_range`
Range of the incoming neighbors of a vertex of type 1.
- typedef `in_edges2_t::const_iterator` `ineighbor_iterator2`
Iterator on the incoming neighbors of a vertex of type 2.
- typedef `util::range`< `ineighbor_iterator2` > `ineighbor_iterator2_range`
Range of the incoming neighbors of a vertex of type 2.
- typedef `util::SelectMemberIterator`< typename `neighborhood1_t::iterator`,
const `vertex1_t`,&`neighborhood1_value_type::first` > `iterator1`
Iterator on the vertices of type 1.
- typedef `util::SelectMemberIterator`< typename `neighborhood2_t::iterator`,
const `vertex2_t`,&`neighborhood2_value_type::first` > `iterator2`
Iterator on the vertices of type 2.
- typedef `util::SelectMemberIterator`< typename `edge_list1_t::iterator`,
`vertex2_t`,&`neighbor1_t::target` > `neighbor_iterator1`
Iterator on the neighbors of a vertex of type 1.
- typedef `util::range`< `neighbor_iterator1` > `neighbor_iterator1_range`
Range of the neighbors of a vertex of type 1.
- typedef `util::SelectMemberIterator`< typename `edge_list2_t::iterator`,
`vertex1_t`,&`neighbor2_t::target` > `neighbor_iterator2`

Iterator on the neighbors of a vertex of type 2.

- typedef [util::range](#)< [neighbor_iterator2](#) > [neighbor_iterator2_range](#)
Range of the neighbors of a vertex of type 2.
- typedef [util::range](#)< [iterator1](#) > [range_vertex1](#)
Range of vertex of type 1.
- typedef [util::range](#)< [iterator2](#) > [range_vertex2](#)
Range of vertex of type 2.

Smart pointer types

- typedef [Edge](#)< const Edge1Content > [const_edge1_t](#)
Weak pointer on a constant edge from a vertex of type 1 to a vertex of type 2.
- typedef [Edge](#)< const Edge2Content > [const_edge2_t](#)
Weak pointer on a constant edge from a vertex of type 2 to a vertex of type 1.
- typedef [Edge](#)< Edge1Content > [edge1_t](#)
Weak pointer on an edge from a vertex of type 1 to a vertex of type 2.
- typedef [Edge](#)< Edge2Content > [edge2_t](#)
Weak pointer on an edge from a vertex of type 2 to a vertex of type 1.
- typedef [Vertex](#)< Vertex1Content > [vertex1_t](#)
Smart pointer on a vertex of type 1.
- typedef [Vertex](#)< Vertex2Content > [vertex2_t](#)
Smart pointer on a vertex of type 2.

Public Member Functions

- [VVBiGraph](#) (const [VVBiGraph](#) ©)
Copy constructor.
- [VVBiGraph](#) ()
Default constructor.

Vertex set lookup methods

- const [vertex1_t](#) & [any_vertex1](#) () const
Return a vertex of type 1 from the graph.
- const [vertex2_t](#) & [any_vertex2](#) () const
Return a vertex of type 2 from the graph.

- `bool contains (const vertex2_t &v) const`
Test if v is in the graph.
- `bool contains (const vertex1_t &v) const`
Test if v is in the graph.
- `bool empty () const`
Test if there is any vertex in the graph.
- `const vertex1_t &get_vertex1 (size_type idx) const`
Return the vertex of type 1 of index idx .
- `const vertex2_t &get_vertex2 (size_type idx) const`
Return the vertex of type 2 of index idx .
- `size_type nb_vertices1 () const`
Return the number of vertices on the graph.
- `size_type nb_vertices2 () const`
Return the number of vertices on the graph.
- `bool no_vertex1 () const`
Test if there is any vertex in the graph.
- `bool no_vertex2 () const`
Test if there is any vertex in the graph.
- `const vertex2_t &reference (vertex2_t v) const`
Get a reference on the vertex in the graph.
- `const vertex1_t &reference (vertex1_t v) const`
Get a reference on the vertex in the graph.
- `size_type size () const`
Return the number of vertices on the graph.

Neighborhood lookup methods

- `const vertex1_t &anyIn (const vertex2_t &v) const`
Return a vertex in the neighborhood of v .
- `const vertex2_t &anyIn (const vertex1_t &v) const`
Return a vertex in the neighborhood of v .
- `const_edge2_t edge (const vertex2_t &src, const vertex1_t &tgt) const`
Returns the edge from src to tgt .
- `const_edge1_t edge (const vertex1_t &src, const vertex2_t &tgt) const`
Returns the edge from src to tgt .

- [edge2_t edge](#) (const [vertex2_t](#) &src, const [vertex1_t](#) &tgt)
Returns the edge from src to tgt.
- [edge1_t edge](#) (const [vertex1_t](#) &src, const [vertex2_t](#) &tgt)
Returns the edge from src to tgt.
- [bool empty](#) (const [vertex2_t](#) &v) const
Test if a vertex has a neighbor.
- [bool empty](#) (const [vertex1_t](#) &v) const
Test if a vertex has a neighbor.
- [bool flag](#) (const [vertex2_t](#) &src, const [vertex1_t](#) &neighbor)
Flag a neighbor of src for quick retrieval.
- [bool flag](#) (const [vertex1_t](#) &src, const [vertex2_t](#) &neighbor)
Flag a neighbor of src for quick retrieval.
- [const vertex1_t & flagged](#) (const [vertex2_t](#) &src) const
Return the flagged neighbor of src.
- [const vertex2_t & flagged](#) (const [vertex1_t](#) &src) const
Return the flagged neighbor of src.
- [const vertex1_t & iAnyIn](#) (const [vertex2_t](#) &v) const
Returns any incoming neighbor of v.
- [const vertex2_t & iAnyIn](#) (const [vertex1_t](#) &v) const
Returns any incoming neighbor of v.
- [bool iEmpty](#) (const [vertex2_t](#) &v) const
Test if a vertex has an incoming neighbor.
- [bool iEmpty](#) (const [vertex1_t](#) &v) const
Test if a vertex has an incoming neighbor.
- [inneighbor_iterator2_range iNeighbors](#) (const [vertex2_t](#) &v) const
Return the range of incoming neighbors of v.
- [inneighbor_iterator1_range iNeighbors](#) (const [vertex1_t](#) &v) const
Return the range of incoming neighbors of v.
- [size_type iValence](#) (const [vertex2_t](#) &v) const
Returns the number of incoming neighbors of v.
- [size_type iValence](#) (const [vertex1_t](#) &v) const
Returns the number of incoming neighbors of v.
- [circ_neighbor_iterator2_range neighbors](#) (const [vertex2_t](#) &v, const [vertex1_t](#) &n)

Return the range of neighbors of v starting from n .

- `circ_neighbor_iterator1_range neighbors` (const `vertex1_t` & v , const `vertex2_t` & n)

Return the range of neighbors of v starting from n .

- `const_circ_neighbor_iterator2_range neighbors` (const `vertex2_t` & v , const `vertex1_t` & n) const

Return the constant range of neighbors of v starting from n .

- `const_circ_neighbor_iterator1_range neighbors` (const `vertex1_t` & v , const `vertex2_t` & n) const

Return the constant range of neighbors of v starting from n .

- `neighbor_iterator2_range neighbors` (const `vertex2_t` & v)

Return the range of neighbors of v .

- `neighbor_iterator1_range neighbors` (const `vertex1_t` & v)

Return the range of neighbors of v .

- `const_neighbor_iterator2_range neighbors` (const `vertex2_t` & v) const

Return the constant range of neighbors of v .

- `const_neighbor_iterator1_range neighbors` (const `vertex1_t` & v) const

Return the constant range of neighbors of v .

- const `vertex1_t` & `nextTo` (const `vertex2_t` & v , const `vertex1_t` & $neighbor$, unsigned int $n=1$) const

Returns the n th vertex after $neighbor$ in the neighborhood of v .

- const `vertex2_t` & `nextTo` (const `vertex1_t` & v , const `vertex2_t` & $neighbor$, unsigned int $n=1$) const

Returns the n th vertex after $neighbor$ in the neighborhood of v .

- const `vertex1_t` & `prevTo` (const `vertex2_t` & v , const `vertex1_t` & ref , unsigned int $n=1$) const

Returns the n th vertex before ref in the neighborhood of v .

- const `vertex2_t` & `prevTo` (const `vertex1_t` & v , const `vertex2_t` & ref , unsigned int $n=1$) const

Returns the n th vertex before ref in the neighborhood of v .

- const `vertex2_t` & `source` (const `edge2_t` & $edge$) const

Return the source vertex of the edge.

- const `vertex1_t` & `source` (const `edge1_t` & $edge$) const

Return the source vertex of the edge.

- const `vertex1_t` & `target` (const `edge2_t` & $edge$) const

Return the target vertex of the edge.

- `const vertex2_t & target (const edge1_t &edge) const`
Return the target vertex of the edge.
- `size_type valence (const vertex2_t &v) const`
Returns the number of neighbors of v.
- `size_type valence (const vertex1_t &v) const`
Returns the number of neighbors of v.

STL compatibility methods

- `const_iterator1 begin_vertex1 () const`
Returns an iterator on the beginning of the set of vertices of the graph.
- `iterator1 begin_vertex1 ()`
Returns an iterator on the beginning of the set of vertices of the graph.
- `const_iterator2 begin_vertex2 () const`
Returns an iterator on the beginning of the set of vertices of the graph.
- `iterator2 begin_vertex2 ()`
Returns an iterator on the beginning of the set of vertices of the graph.
- `void clear (iterator2 it)`
Clear the neighborhood of a vertex.
- `void clear (iterator1 it)`
Clear the neighborhood of a vertex.
- `size_type count (const vertex2_t &v) const`
Count the number of vertices v in the graph (can be 0 or 1 only).
- `size_type count (const vertex1_t &v) const`
Count the number of vertices v in the graph (can be 0 or 1 only).
- `const_iterator1 end_vertex1 () const`
Returns an iterator on the end of the set of vertices of the graph.
- `iterator1 end_vertex1 ()`
Returns an iterator on the end of the set of vertices of the graph.
- `const_iterator2 end_vertex2 () const`
Returns an iterator on the end of the set of vertices of the graph.
- `iterator2 end_vertex2 ()`
Returns an iterator on the end of the set of vertices of the graph.
- `iterator2 erase (iterator2 pos)`
Erase element at position pos.

- [iterator1 erase](#) ([iterator1](#) pos)
Erase element at position pos.
- void [eraseAllEdges](#) ([iterator2](#) it)
Clear the neighborhood of a vertex.
- void [eraseAllEdges](#) ([iterator1](#) it)
Clear the neighborhood of a vertex.
- [circ_neighbor_iterator2 eraseEdge](#) (const [vertex2_t](#) &v, [circ_neighbor_iterator2](#) pos)
Erase the edge pointed to by the iterator.
- [neighbor_iterator2 eraseEdge](#) (const [vertex2_t](#) &v, [neighbor_iterator2](#) pos)
Erase the edge pointed to by the iterator.
- [circ_neighbor_iterator1 eraseEdge](#) (const [vertex1_t](#) &v, [circ_neighbor_iterator1](#) pos)
Erase the edge pointed to by the iterator.
- [neighbor_iterator1 eraseEdge](#) (const [vertex1_t](#) &v, [neighbor_iterator1](#) pos)
Erase the edge pointed to by the iterator.
- [const_iterator2 find](#) (const [vertex2_t](#) &v) const
Find the vertex v in the graph.
- [const_iterator1 find](#) (const [vertex1_t](#) &v) const
Find the vertex v in the graph.
- [iterator2 find](#) (const [vertex2_t](#) &v)
Find the vertex v in the graph.
- [iterator1 find](#) (const [vertex1_t](#) &v)
Find the vertex v in the graph.
- [const_neighbor_iterator2 findIn](#) (const [vertex2_t](#) &v, const [vertex1_t](#) &n) const
Find the vertex n in the neighborhood of v.
- [const_neighbor_iterator1 findIn](#) (const [vertex1_t](#) &v, const [vertex2_t](#) &n) const
Find the vertex n in the neighborhood of v.
- [neighbor_iterator2 findIn](#) (const [vertex2_t](#) &v, const [vertex1_t](#) &n)
Find the vertex n in the neighborhood of v.
- [neighbor_iterator1 findIn](#) (const [vertex1_t](#) &v, const [vertex2_t](#) &n)
Find the vertex n in the neighborhood of v.
- void [insert](#) ([iterator2](#) first, [iterator2](#) last)
Insert all the vertices from first to last-1 in the graph.

- void [insert](#) (iterator1 first, iterator1 last)
Insert all the vertices from first to last-1 in the graph.
- iterator2 [insert](#) (iterator2 pos, const vertex2_t &v)
Insert a new vertex in the graph.
- iterator1 [insert](#) (iterator1 pos, const vertex1_t &v)
Insert a new vertex in the graph.
- range_vertex1 [vertices1](#) ()
Returns a range of vertices.
- const_range_vertex1 [vertices1](#) () const
Returns a range of vertices.
- range_vertex2 [vertices2](#) ()
Returns a range of vertices.
- const_range_vertex2 [vertices2](#) () const
Returns a range of vertices.

Whole structure methods

- void [clear](#) ()
Clear the graph.
- void [eraseAllEdges](#) ()
Clear all the edges in the graph.
- VVBIGraph & [operator=](#) (const VVBIGraph &other)
Copy the graph into another graph.
- bool [operator==](#) (const VVBIGraph &other) const
Test equality for the graphs.
- template<typename Vertex1Container, typename Vertex2Container >
[VVBIGraph subgraph](#) (const Vertex1Container &vertices1, const Vertex2Container &vertices2) const
- void [swap](#) (VVBIGraph &other)
Swap the content of two graphs.

Neighborhood edition methods

- bool [clear](#) (const vertex2_t &v)
Clear the neighborhood of a vertex.
- bool [clear](#) (const vertex1_t &v)
Clear the neighborhood of a vertex.

- `bool eraseAllEdges (const vertex2_t &v)`
Clear the neighborhood of a vertex.
- `bool eraseAllEdges (const vertex1_t &v)`
Clear the neighborhood of a vertex.
- `size_type eraseEdge (const vertex2_t &src, const vertex1_t &tgt)`
Erase the edge from `src` to `tgt`.
- `size_type eraseEdge (const vertex1_t &src, const vertex2_t &tgt)`
Erase the edge from `src` to `tgt`.
- `edge2_t insertEdge (const vertex2_t &src, const vertex1_t &tgt)`
Insert a new edge in the graph, without ordering.
- `edge1_t insertEdge (const vertex1_t &src, const vertex2_t &tgt)`
Insert a new edge in the graph, without ordering.
- `edge2_t replace (const vertex2_t &v, const vertex1_t &neighbor, const vertex1_t &new_neighbor)`
Replace a vertex by another in a neighborhood.
- `edge1_t replace (const vertex1_t &v, const vertex2_t &neighbor, const vertex2_t &new_neighbor)`
Replace a vertex by another in a neighborhood.
- `edge2_t spliceAfter (const vertex2_t &v, const vertex1_t &neighbor, const vertex1_t &new_neighbor)`
Insert neighbor in the neighborhood of `v` after reference.
- `edge1_t spliceAfter (const vertex1_t &v, const vertex2_t &neighbor, const vertex2_t &new_neighbor)`
Insert neighbor in the neighborhood of `v` after reference.
- `edge2_t spliceBefore (const vertex2_t &v, const vertex1_t &neighbor, const vertex1_t &new_neighbor)`
Insert neighbor in the neighborhood of `v` before reference.
- `edge1_t spliceBefore (const vertex1_t &v, const vertex2_t &neighbor, const vertex2_t &new_neighbor)`
Insert neighbor in the neighborhood of `v` before reference.

Vertex set edition methods

- `size_type erase (const vertex2_t &v)`
Remove a vertex from the graph.
- `size_type erase (const vertex1_t &v)`
Remove a vertex from the graph.

- [iterator2 insert](#) (const [vertex2_t](#) &v)
Insert a new vertex in the graph.
- [iterator1 insert](#) (const [vertex1_t](#) &v)
Insert a new vertex in the graph.

Protected Types

- typedef [search_result_t](#)< const [single_neighborhood1_t](#), [int_const_neighbor_iterator1](#) > [const_neighbor1_found_t](#)
Constant result of a search in the neighborhood.
- typedef [search_result_t](#)< const [single_neighborhood2_t](#), [int_const_neighbor_iterator2](#) > [const_neighbor2_found_t](#)
Constant result of a search in the neighborhood.
- typedef [neighbor1_t::edge_list_t](#) [edge_list1_t](#)
Type of the ordered list of the outgoing edges of a vertex of type 1.
- typedef [neighbor2_t::edge_list_t](#) [edge_list2_t](#)
Type of the ordered list of the outgoing edges of a vertex of type 2.
- typedef [neighbor1_t::base](#) **EdgeCache1**
- typedef [neighbor2_t::base](#) **EdgeCache2**
- typedef [__vvbigraph_set](#)< compact, [vertex2_t](#) >::type [in_edges1_t](#)
Type of the list of sources of the in-edges of a vertex of type 1.
- typedef [__vvbigraph_set](#)< compact, [vertex1_t](#) >::type [in_edges2_t](#)
Type of the list of sources of the in-edges of a vertex of type 2.
- typedef [edge_list1_t::const_iterator](#) [int_const_neighbor_iterator1](#)
Constant iterator on the outgoing edges.
- typedef [edge_list2_t::const_iterator](#) [int_const_neighbor_iterator2](#)
Constant iterator on the outgoing edges.
- typedef [edge_list1_t::iterator](#) [int_neighbor_iterator1](#)
Iterator on the outgoing edges.
- typedef [edge_list2_t::iterator](#) [int_neighbor_iterator2](#)
Iterator on the outgoing edges.
- typedef [search_result_t](#)< [single_neighborhood1_t](#), [int_neighbor_iterator1](#) > [neighbor1_found_t](#)

Result of a search in the neighborhood.

- `typedef neighbor_t< Vertex1Content, Vertex2Content, Edge1Content > neighbor1_t`

Type of a neighbor of a vertex of type 1 (i.e.

- `typedef search_result_t< single_neighborhood2_t, int_neighbor_iterator2 > neighbor2_found_t`

Result of a search in the neighborhood.

- `typedef neighbor_t< Vertex2Content, Vertex1Content, Edge2Content > neighbor2_t`

Type of a neighbor of a vertex of type 1 (i.e.

- `typedef single_neighborhood_t< Vertex1Content, Vertex2Content, Edge1Content > single_neighborhood1_t`

Type of the neighborhood of a vertex of type 1.

- `typedef single_neighborhood_t< Vertex2Content, Vertex1Content, Edge2Content > single_neighborhood2_t`

Type of the neighborhood of a vertex of type 2.

- `typedef single_neighborhood1_t::base VertexCache1`
- `typedef single_neighborhood2_t::base VertexCache2`

Protected Member Functions

- `const_neighbor2_found_t findInVertex (const vertex2_t &v, const vertex1_t &neighbor) const`

Constant version of `findInVertex(const vertex_t&, const vertex_t&)`.

- `const_neighbor1_found_t findInVertex (const vertex1_t &v, const vertex2_t &neighbor) const`

Constant version of `findInVertex(const vertex_t&, const vertex_t&)`.

- `neighbor2_found_t findInVertex (const vertex2_t &v, const vertex1_t &neighbor)`

Find a vertex neighbor in the neighborhood of v.

- `neighbor1_found_t findInVertex (const vertex1_t &v, const vertex2_t &neighbor)`

Find a vertex neighbor in the neighborhood of v.

- `neighborhood2_t::const_iterator findVertex (const vertex2_t &v) const`

Constant version of `findVertex(const vertex2_t&)`.

- `neighborhood1_t::const_iterator findVertex (const vertex1_t &v) const`

Constant version of *findVertex(const vertex1_t&)*.

- neighborhood2_t::iterator *findVertex* (const vertex2_t &v)
Find a vertex in the graph and returns the iterator on it.
- neighborhood1_t::iterator *findVertex* (const vertex1_t &v)
Find a vertex in the graph and returns the iterator on it.
- std::pair< ineighbor_iterator1, bool > *insertInEdge* (const vertex2_t &v, const vertex1_t &new_neighbor)
Insert v in the in_edges of new_neighbor and return true if the insertion was actually done, or false if v was already there.
- std::pair< ineighbor_iterator2, bool > *insertInEdge* (const vertex1_t &v, const vertex2_t &new_neighbor)
Insert v in the in_edges of new_neighbor and return true if the insertion was actually done, or false if v was already there.
- void *undoInEdge* (const vertex2_t &new_neighbor, ineighbor_iterator2 &it)
Undo the insertion of an in edge as a roll-back mechanism if the whole edge insertion fails.
- void *undoInEdge* (const vertex1_t &new_neighbor, ineighbor_iterator1 &it)
Undo the insertion of an in edge as a roll-back mechanism if the whole edge insertion fails.
- void *updateEdgeCache* (const vertex2_t &v, typename edge_list2_t::iterator new_edge_it, const vertex2_t &in_edge)
Update the edge cache information of vertex v in its neighborhood and the in_edge too.
- void *updateEdgeCache* (const vertex1_t &v, typename edge_list1_t::iterator new_edge_it, const vertex1_t &in_edge)
Update the edge cache information of vertex v in its neighborhood and the in_edge too.
- void *updateVertexCache* (const vertex2_t &v, typename neighborhood2_t::iterator it)
Update the vertex cache for vertex v.
- void *updateVertexCache* (const vertex1_t &v, typename neighborhood1_t::iterator it)
Update the vertex cache for vertex v.

Protected Attributes

- `intptr_t graph_id`
Unique id for this graph.
- `lookup1_t lookup1`
Lookup hash map for vertex1's.
- `lookup2_t lookup2`
Lookup hash map for vertex2's.
- `neighborhood1_t neighborhood1`
Main data structure containing the neighborhood of the vertex1's.
- `neighborhood2_t neighborhood2`
Main data structure containing the neighborhood of the vertex2's.

17.72.1 Detailed Description

```
template<typename Vertex1Content, typename Vertex2Content, type-
name Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_
= Edge1Content, bool compact = false> class graph::VVBiGraph< Ver-
tex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >
```

Class representing a VV graph. A VV graph is an oriented rotational graph. That is, the edges of a given vertex have a circular order. That is, if an edge has three edges e_1, e_2, e_3 , there is no first edge, but we can say $e_1 < e_2 < e_3 < e_1$. In that case, ' $<$ ' denote the succession.

`VertexContent` is the type used to hold the vertex data.

`EdgeContent` is the type used to hold the edge data. If no type is specified for `EdgeContent`, then an empty structure is used.

17.72.2 Performance notes

The graph includes two caching mechanisms to improve performances when iterating over the graph and retrieve edge information:

1. while iterating over the graph, the vertex cache the current iterator. So that accessing the neighborhood of the vertex is constant time (i.e. it's dereferencing the cached iterator).
2. while iterating over the neighborhood of a vertex, the target vertex (i.e. the one returned by the iteration) cache the iterator on the neighborhood. So that any operation using that specific edge (i.e. getting edge data, splicing, replacing or erasing) is also constant time.

To make this caching mechanism correct, the cached data are never copied. This means, to make use of the caching mechanism, you have to iterate using constant references (references will end up with compilation errors if used on the graph):

```
forall_named(const vertex1_t &v, S, vertex1)
{
    // Here the vertex v has the vertex cache
    forall(const vertex2_t &n, S.neighbors(v)) // Makes use of the cache
    {
        S.edge(v,n)->pos = Point3d(1,2,3); // Uses the edge cache
        S.edge(n,v)->pos = Point3d(3,2,1); // Do not use any cache
    }
    v->pos = Point3d(0,0,0); // Valid
}
```

Warning

Do not iterate using constant references if you ever remove the vertex you iterate on ! (i.e. remove from the graph if you iterate on the graph or remove from the neighborhood if you iterate on the neighborhood)

Even though this code manipulate constant references, modifying the data pointed by the vertex is allowed. The vertices are coded to allow that. That means if you are using vertices in a dictionary or a set, using their content to compare them, you must not modify them ! The compiler won't guard you against it.

Definition at line 336 of file vvbigraph.h.

17.72.3 Member Typedef Documentation

17.72.3.1 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::CircIterator<neighbor_iterator1> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::circ_neighbor_iterator1`

Iterator used to iterate over the neighbors of a vertex of type 1, but specifying the starting point.

Definition at line 690 of file vvbigraph.h.

17.72.3.2 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<circ_neighbor_iterator1> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::circ_neighbor_iterator1_range`

Range of circular iterators for the neighbors of a vertex of type 1.

Definition at line 700 of file vvbigraph.h.

17.72.3.3 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::CircIterator<neighbor_iterator2> graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::circ_neighbor_iterator2`

Iterator used to iterate over the neighbors of a vertex of type 2, but specifying the starting point.

Definition at line 695 of file vvbigraph.h.

17.72.3.4 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<circ_neighbor_iterator2> graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::circ_neighbor_iterator2_range`

Range of circular iterators for the neighbors of a vertex of type 2.

Definition at line 704 of file vvbigraph.h.

17.72.3.5 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::CircIterator<const_neighbor_iterator1> graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_circ_neighbor_iterator1`

Iterator used to iterate over the neighbors of a vertex of type 1, but specifying the starting point.

Definition at line 772 of file vvbigraph.h.

17.72.3.6 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<const_circ_neighbor_iterator1> graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_circ_neighbor_iterator1_range`

Range of circular iterators for the neighbors of a vertex of type 1.

Definition at line 782 of file vvbigraph.h.

17.72.3.7 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::CircIterator<const_neighbor_iterator2> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_circ_neighbor_iterator2`

Iterator used to iterate over the neighbors of a vertex of type 2, but specifying the starting point.

Definition at line 777 of file vvbigraph.h.

17.72.3.8 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<const_circ_neighbor_iterator2> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_circ_neighbor_iterator2_range`

Range of circular iterators for the neighbors of a vertex of type 2.

Definition at line 786 of file vvbigraph.h.

17.72.3.9 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef Edge<const Edge1Content> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_edge1_t`

Weak pointer on a constant edge from a vertex of type 1 to a vertex of type 2.

Definition at line 382 of file vvbigraph.h.

17.72.3.10 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef Edge<const Edge2Content> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_edge2_t`

Weak pointer on a constant edge from a vertex of type 2 to a vertex of type 1.

Definition at line 387 of file vvbigraph.h.

17.72.3.11 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::SelectMemberIterator<typename neighborhood1_t::const_iterator, const vertex1_t, &neighborhood1_value_type::first> graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_iterator1`

Constant iterator on the vertices of type 1.

Reimplemented in `vvcomplex::VVComplexGraph<CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >`, `vvcomplex::VVComplexGraph<Cell, Vertex >`, and `vvcomplex::VVComplexGraph<RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >`.

Definition at line 656 of file `vvbigraph.h`.

17.72.3.12 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::SelectMemberIterator<typename neighborhood2_t::const_iterator, const vertex2_t, &neighborhood2_value_type::first> graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_iterator2`

Constant iterator on the vertices of type 1.

Reimplemented in `vvcomplex::VVComplexGraph<CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >`, `vvcomplex::VVComplexGraph<Cell, Vertex >`, and `vvcomplex::VVComplexGraph<RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >`.

Definition at line 661 of file `vvbigraph.h`.

17.72.3.13 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef search_result_t<const single_neighborhood1_t, int_const_neighbor_iterator1> graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_neighbor1_found_t [protected]`

Constant result of a search in the neighborhood.

Definition at line 1705 of file `vvbigraph.h`.

17.72.3.14 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef search_result_t<const single_neighborhood2_t, int_const_neighbor_iterator2> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_neighbor2_found_t [protected]`

Constant result of a search in the neighborhood.

Definition at line 1709 of file vvbigraph.h.

17.72.3.15 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::SelectMemberIterator<typename edge_list1_t::const_iterator, const vertex2_t, &neighbor1_t::target> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_neighbor_iterator1`

Constant iterator on the neighbors of a vertex of type 1.

Definition at line 744 of file vvbigraph.h.

17.72.3.16 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<const_neighbor_iterator1> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_neighbor_iterator1_range`

Constant range of the neighbors of a vertex of type 1.

The `first` element of the pair is a constant iterator on the beginning of the range, the `second` element is the past-the-end constant iterator.

Definition at line 758 of file vvbigraph.h.

17.72.3.17 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::SelectMemberIterator<typename edge_list2_t::const_iterator, const vertex1_t, &neighbor2_t::target> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_neighbor_iterator2`

Constant iterator on the neighbors of a vertex of type 2.

Definition at line 749 of file vvbigraph.h.

17.72.3.18 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<const_neighbor_iterator2> graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_neighbor_iterator2_range`

Constant range of the neighbors of a vertex of type 2.

The `first` element of the pair is a constant iterator on the beginning of the range, the `second` element is the past-the-end constant iterator.

Definition at line 766 of file `vvbigraph.h`.

17.72.3.19 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<const_iterator1> graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_range_vertex1`

Range of vertex of type 1.

Definition at line 791 of file `vvbigraph.h`.

17.72.3.20 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<const_iterator2> graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::const_range_vertex2`

Range of vertex of type 2.

Definition at line 795 of file `vvbigraph.h`.

17.72.3.21 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef Edge<Edge1Content> graph::VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::edge1_t`

Weak pointer on an edge from a vertex of type 1 to a vertex of type 2.

Definition at line 372 of file `vvbigraph.h`.

17.72.3.22 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef Edge<Edge2Content> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::edge2_t`

Weak pointer on an edge from a vertex of type 2 to a vertex of type 1.

Definition at line 376 of file vvbigraph.h.

17.72.3.23 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef neighbor1_t::edge_list_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::edge_list1_t [protected]`

Type of the ordered list of the outgoing edges of a vertex of type 1.

Definition at line 510 of file vvbigraph.h.

17.72.3.24 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef neighbor2_t::edge_list_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::edge_list2_t [protected]`

Type of the ordered list of the outgoing edges of a vertex of type 2.

Definition at line 514 of file vvbigraph.h.

17.72.3.25 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef __vvbigraph_set<compact,vertex2_t>::type graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::in_edges1_t [protected]`

Type of the list of sources of the in-edges of a vertex of type 1.

Definition at line 519 of file vvbigraph.h.

17.72.3.26 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef __vvbigraph_set<compact,vertex1_t>::type graph::VVBigraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::in_edges2_t [protected]`

Type of the list of sources of the in-edges of a vertex of type 2.

Definition at line 523 of file vvbigraph.h.

17.72.3.27 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef in_edges1_t::const_iterator graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::ineighbor_iterator1`

Iterator on the incoming neighbors of a vertex of type 1.

An incoming neighbor is a source vertex of an edge whose target is the current vertex.

Definition at line 712 of file vvbigraph.h.

17.72.3.28 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<ineighbor_iterator1> graph::VVBigraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::ineighbor_iterator1_range`

Range of the incoming neighbors of a vertex of type 1.

The `first` element of the pair is an iterator on the beginning of the range, the `second` element is the past-the-end iterator.

See also

`ineighbor_iterator`

Definition at line 729 of file vvbigraph.h.

17.72.3.29 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef in_edges2_t::const_iterator graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::ineighbor_iterator2`

Iterator on the incoming neighbors of a vertex of type 2.

An incoming neighbor is a source vertex of an edge whose target is the current vertex.

Definition at line 719 of file vvbigraph.h.

17.72.3.30 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<ineighbor_iterator2> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::ineighbor_iterator2_range`

Range of the incoming neighbors of a vertex of type 2.

The `first` element of the pair is an iterator on the beginning of the range, the `second` element is the past-the-end iterator.

See also

`ineighbor_iterator`

Definition at line 738 of file vvbigraph.h.

17.72.3.31 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef edge_list1_t::const_iterator graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::int_const_neighbor_iterator1 [protected]`

Constant iterator on the outgoing edges.

Definition at line 1687 of file vvbigraph.h.

17.72.3.32 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef edge_list2_t::const_iterator graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::int_const_neighbor_iterator2 [protected]`

Constant iterator on the outgoing edges.

Definition at line 1691 of file vvbigraph.h.

17.72.3.33 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef edge_list1_t::iterator graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::int_neighbor_iterator1 [protected]`

Iterator on the outgoing edges.

Definition at line 1678 of file vvbigraph.h.

17.72.3.34 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef edge_list2_t::iterator graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::int_neighbor_iterator2 [protected]`

Iterator on the outgoing edges.

Definition at line 1682 of file vvbigraph.h.

17.72.3.35 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::SelectMemberIterator<typename neighborhood1_t::iterator, const vertex1_t, &neighborhood1_value_type::first> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iterator1`

Iterator on the vertices of type 1.

Reimplemented in `vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >`, `vvcomplex::VVComplexGraph< Cell, Vertex >`, and `vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >`.

Definition at line 646 of file vvbigraph.h.

17.72.3.36 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::SelectMemberIterator<typename neighborhood2_t::iterator, const vertex2_t, &neighborhood2_value_type::first> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iterator2`

Iterator on the vertices of type 2.

17.72 graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact > Class Template Reference 723

Reimplemented in [vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >](#), [vvcomplex::VVComplexGraph< Cell, Vertex >](#), and [vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS\(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >\), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >](#).

Definition at line 651 of file `vvbigraph.h`.

17.72.3.37 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef std::unordered_map<vertex1_t, typename neighborhood1_t::iterator> graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1_t`

Type of the lookup hash map for vertices of type 1.

Definition at line 619 of file `vvbigraph.h`.

17.72.3.38 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef std::unordered_map<vertex2_t, typename neighborhood2_t::iterator> graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup2_t`

Type of the lookup hash map for vertices of type 2.

Definition at line 624 of file `vvbigraph.h`.

17.72.3.39 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef search_result_t<single_neighborhood1_t, int_neighbor_iterator1> graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbor1_found_t [protected]`

Result of a search in the neighborhood.

Definition at line 1696 of file `vvbigraph.h`.

17.72.3.40 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef neighbor_t<Vertex1Content, Vertex2Content, Edge1Content> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbor1_t [protected]`

Type of a neighbor of a vertex of type 1 (i.e. it is of type 2)

Definition at line 496 of file vvbigraph.h.

17.72.3.41 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef search_result_t<single_neighborhood2_t, int_neighbor_iterator2> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbor2_found_t [protected]`

Result of a search in the neighborhood.

Definition at line 1700 of file vvbigraph.h.

17.72.3.42 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef neighbor_t<Vertex2Content, Vertex1Content, Edge2Content> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbor2_t [protected]`

Type of a neighbor of a vertex of type 1 (i.e. it is of type 2)

Definition at line 503 of file vvbigraph.h.

17.72.3.43 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::SelectMemberIterator<typename edge_list1_t::iterator, vertex2_t, &neighbor1_t::target> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbor_iterator1`

Iterator on the neighbors of a vertex of type 1.

Definition at line 667 of file vvbigraph.h.

17.72.3.44 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<neighbor_iterator1> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbor_iterator1_range`

Range of the neighbors of a vertex of type 1.

The `first` element of the pair is an iterator on the beginning of the range, the `second` element is the past-the-end iterator.

Definition at line 678 of file `vvbigraph.h`.

17.72.3.45 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::SelectMemberIterator<typename edge_list2_t::iterator, vertex1_t, &neighbor2_t::target> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbor_iterator2`

Iterator on the neighbors of a vertex of type 2.

Definition at line 672 of file `vvbigraph.h`.

17.72.3.46 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<neighbor_iterator2> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbor_iterator2_range`

Range of the neighbors of a vertex of type 2.

The `first` element of the pair is an iterator on the beginning of the range, the `second` element is the past-the-end iterator.

Definition at line 684 of file `vvbigraph.h`.

17.72.3.47 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef std::list<std::pair<vertex1_t, single_neighborhood1_t>> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1_t`

Type of the list of vertices of type 1, together with their neighborhood.

Definition at line 609 of file `vvbigraph.h`.

17.72.3.48 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef neighborhood1_t::value_type graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1_value_type`

Shortcut for the value_type of the neighborhood for a vertex of type 1.

Definition at line 629 of file vvbigraph.h.

17.72.3.49 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef std::list<std::pair<vertex2_t, single_neighborhood2_t> > graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood2_t`

Type of the list of vertices of type 2, together with their neighborhood.

Definition at line 614 of file vvbigraph.h.

17.72.3.50 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef neighborhood2_t::value_type graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood2_value_type`

Shortcut for the value_type of the neighborhood for a vertex of type 2.

Definition at line 633 of file vvbigraph.h.

17.72.3.51 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<iterator1> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::range_vertex1`

Range of vertex of type 1.

Definition at line 800 of file vvbigraph.h.

17.72.3.52 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef util::range<iterator2> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::range_vertex2`

Range of vertex of type 2.

Definition at line 804 of file vvbigraph.h.

17.72.3.53 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef single_neighborhood_t<Vertex1Content, Vertex2Content, Edge1Content> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::single_neighborhood1_t [protected]`

Type of the neighborhood of a vertex of type 1.

Definition at line 594 of file vvbigraph.h.

17.72.3.54 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef single_neighborhood_t<Vertex2Content, Vertex1Content, Edge2Content> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::single_neighborhood2_t [protected]`

Type of the neighborhood of a vertex of type 2.

Definition at line 598 of file vvbigraph.h.

17.72.3.55 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef neighborhood1_t::size_type graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::size_type`

Type of a size.

Reimplemented in `vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >`, `vvcomplex::VVComplexGraph< Cell, Vertex >`, and `vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >`.

Definition at line 638 of file vvbigraph.h.

17.72.3.56 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef Vertex<Vertex1Content> graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::vertex1_t`

Smart pointer on a vertex of type 1.

Definition at line 358 of file vvbigraph.h.

17.72.3.57 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> typedef Vertex<Vertex2Content> graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::vertex2_t`

Smart pointer on a vertex of type 2.

Definition at line 366 of file vvbigraph.h.

17.72.4 Constructor & Destructor Documentation

17.72.4.1 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::VVBigraph () [inline]`

Default constructor.

Creates an empty VV graph

Definition at line 395 of file vvbigraph.h.

```
395 : graph_id(graph::getNextGraphId()) {}
```

17.72.4.2 `template<BIGRAPH_TEMPLATE > graph::VVBigraph< BIGRAPH_TEMPLATE >::VVBigraph (const VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact > ©) [inline]`

Copy constructor.

Maintains the caches in a valid state

Definition at line 3380 of file vvbigraph.h.

```
3381      : graph_id(graph::getNextGraphId())
3382      {
3383          *this = copy;
3384      }
```

17.72.5 Member Function Documentation

17.72.5.1 `template<BIGRAPH_TEMPLATE > const VVBIGraph< BIGRAPH_ARGS >::vertex1_t & graph::VVBIGraph< BIGRAPH_TEMPLATE >::any_vertex1 () const` **[inline]**

Return a vertex of type 1 from the graph.

Returns

A vertex of the graph, or a null vertex if the graph is empty.

Definition at line 2182 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::begin_vertex1()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`, and `graph::Vertex< VertexContent >::null`.

```
2183      {
2184          if(neighborhood1.empty())
2185              return vertex1_t::null;
2186          return *begin_vertex1();
2187      }
```

17.72.5.2 `template<BIGRAPH_TEMPLATE > const VVBIGraph< BIGRAPH_ARGS >::vertex2_t & graph::VVBIGraph< BIGRAPH_TEMPLATE >::any_vertex2 () const` **[inline]**

Return a vertex of type 2 from the graph.

Returns

A vertex of the graph, or a null vertex if the graph is empty.

Definition at line 2191 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::begin_vertex2()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood2`, and `graph::Vertex< VertexContent >::null`.

```

2192     {
2193         if(neighborhood2.empty())
2194             return vertex2_t::null;
2195         return *begin_vertex2();
2196     }

```

17.72.5.3 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const vertex1_t& graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::anyIn (const vertex2_t & v) const`

Return a vertex in the neighborhood of v.

Returns

A vertex in the neighborhood of v, or a null vertex if v is not in the graph or v has no neighborhood.

17.72.5.4 `template<BIGRAPH_TEMPLATE > const VVBiGraph< BIGRAPH_ARGS >::vertex1_t & graph::VVBiGraph< BIGRAPH_TEMPLATE >::anyIn (const vertex1_t & v) const [inline]`

Return a vertex in the neighborhood of v.

Returns

A vertex in the neighborhood of v, or a null vertex if v is not in the graph or v has no neighborhood.

Definition at line 2218 of file vvbigraph.h.

References `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::Vertex< VertexContent >::isNull()`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`, and `graph::Vertex< VertexContent >::null`.

Referenced by `algorithms::TriangleSurface::center3d()`, `algorithms::TriangleSurface::centerPosition()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteJunction()`, and `algorithms::TriangleSurface::position()`.

```

2219     {
2220         if(v.isNull())
2221             return vertex2_t::null;//vertex_t(0);
2222         typename neighborhood1_t::const_iterator it_found = this->findVertex(v);

```



```

2223         if(it_found == neighborhood1.end() || it_found->second.edges.empty())
2224             return vertex2_t::null; //vertex_t(0);
2225         const edge_list1_t& lst = it_found->second.edges;
2226         return lst.front().target;
2227     }

```

17.72.5.5 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const_iterator1 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::begin_vertex1 () const [inline]`

Returns an iterator on the beginning of the set of vertices of the graph.

Definition at line 1497 of file vvbigraph.h.

```

1497 { return neighborhood1.begin(); }

```

17.72.5.6 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> iterator1 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::begin_vertex1 () [inline]`

Returns an iterator on the beginning of the set of vertices of the graph.

Definition at line 1461 of file vvbigraph.h.

Referenced by `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::any_vertex1()`, and `graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::vertices1()`.

```

1461 { return neighborhood1.begin(); }

```

17.72.5.7 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const_iterator2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::begin_vertex2 () const [inline]`

Returns an iterator on the beginning of the set of vertices of the graph.

Definition at line 1501 of file vvbigraph.h.

```

1501 { return neighborhood2.begin(); }

```

17.72.5.8 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> iterator2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::begin_vertex2 () [inline]`

Returns an iterator on the beginning of the set of vertices of the graph.

Definition at line 1465 of file vvbigraph.h.

Referenced by `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::any_vertex2()`, and `graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::vertices2()`.

```
1465 { return neighborhood2.begin(); }
```

17.72.5.9 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> void graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::clear (iterator2 it)`

Clear the neighborhood of a vertex.

All edges from `*it` will be erased.

17.72.5.10 `template<BIGRAPH_TEMPLATE > void graph::VVBIGraph< BIGRAPH_TEMPLATE >::clear (iterator1 it) [inline]`

Clear the neighborhood of a vertex.

All edges from `*it` will be erased.

Definition at line 2024 of file vvbigraph.h.

References `util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::base()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`.

```
2025     {
2026         const vertex1_t& v = *it_found;
2027         for(typename edge_list1_t::iterator it_el = it_found.base()->second.edges.b
egin() ;
2028             it_el != it_found.base()->second.edges.end() ; ++it_el)
2029         {
2030             typename neighborhood2_t::iterator it_found = this->findVertex(it_el->tar
get);
2031             it_found->second.in_edges.erase(v);
2032         }
2033         it_found.base()->second.edges.clear();
2034     }
```

17.72.5.11 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false>
void graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::clear () [inline]`

Clear the graph.

Definition at line 1364 of file vvbigraph.h.

Referenced by graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::clear().

```
1364 { neighborhood1.clear(); neighborhood2.clear(); lookup1.clear(); lookup2.clear();  
    }
```

17.72.5.12 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false>
bool graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::clear (const vertex2_t & v)`

Clear the neighborhood of a vertex.

All edges from *v* will be erased.

17.72.5.13 `template<BIGRAPH_TEMPLATE > bool graph::VVBIGraph< BIGRAPH_TEMPLATE >::clear (const vertex1_t & v) [inline]`

Clear the neighborhood of a vertex.

All edges from *v* will be erased.

Definition at line 2000 of file vvbigraph.h.

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::clear(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1.

Referenced by algorithms::TriangleSurface::clear(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::clear(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell(), and algorithms::TriangleGrowth::init().

```
2001 {  
2002     typename neighborhood1_t::iterator it_found = this->findVertex(v);  
2003     if(it_found != neighborhood1.end() )
```

```

2004     {
2005         clear(it_found);
2006         return true;
2007     }
2008     return false;
2009 }
```

17.72.5.14 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> bool graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::contains (const vertex2_t & v) const`

Test if v is in the graph.

Returns

`true` is v is in the graph.

17.72.5.15 `template<BIGRAPH_TEMPLATE > bool graph::VVBiGraph< BIGRAPH_TEMPLATE >::contains (const vertex1_t & v) const [inline]`

Test if v is in the graph.

Returns

`true` is v is in the graph.

Definition at line 2707 of file `vvbigraph.h`.

References `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, and `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

```

2708     {
2709         return this->findVertex(v) != neighborhood1.end();
2710     }
```

17.72.5.16 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> size_type graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::count (const vertex2_t & v) const`

Count the number of vertices v in the graph (can be 0 or 1 only).

17.72.5.17 `template<BIGRAPH_TEMPLATE > VVBIGraph<
BIGRAPH_ARGS >::size_type graph::VVBIGraph<
BIGRAPH_TEMPLATE >::count (const vertex1_t & v) const
[inline]`

Count the number of vertices `v` in the graph (can be 0 or 1 only).

Definition at line 2070 of file `vvbigraph.h`.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1`.

```
2071     {  
2072         return lookup1.count(v);  
2073     }
```

17.72.5.18 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
const_edge2_t graph::VVBIGraph< Vertex1Content,
Vertex2Content, Edge1Content, Edge2Content_, compact >::edge
(const vertex2_t & src, const vertex1_t & tgt) const`

Returns the edge from `src` to `tgt`.

If the edge does not exists, returns a null edge.

17.72.5.19 `template<BIGRAPH_TEMPLATE > VVBIGraph<
BIGRAPH_ARGS >::const_edge2_t graph::VVBIGraph<
BIGRAPH_TEMPLATE >::edge (const vertex1_t & src, const
vertex2_t & tgt) const [inline]`

Returns the edge from `src` to `tgt`.

If the edge does not exists, returns a null edge.

Definition at line 2948 of file `vvbigraph.h`.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::Vertex< VertexContent >::id()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it`.

```
2950     {  
2951         const_neighbor1_found_t found = findInVertex(src, tgt);  
2952         if (!found)  
2953             return const_edge1_t();  
2954         return const_edge1_t(src.id(), tgt.id(), &*found.it);  
2955     }
```

17.72.5.20 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> edge2_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::edge (const vertex2_t & src, const vertex1_t & tgt)`

Returns the edge from `src` to `tgt`.

If the edge does not exists, returns a null edge.

17.72.5.21 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::edge2_t graph::VVBIGraph< BIGRAPH_TEMPLATE >::edge (const vertex1_t & src, const vertex2_t & tgt) [inline]`

Returns the edge from `src` to `tgt`.

If the edge does not exists, returns a null edge.

Definition at line 2926 of file `vvbigraph.h`.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::Vertex< VertexContent >::id()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::connectFromJunctions()`, and `algorithms::TriangleGrowth::readOBJS()`.

```

2928     {
2929         neighbor1_found_t found = findInVertex(src, tgt);
2930         if(!found)
2931             return edge1_t();
2932         return edge1_t(src.id(), tgt.id(), &*found.it);
2933     }

```

17.72.5.22 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> bool graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::empty (const vertex2_t & v) const`

Test if a vertex has a neighbor.

Returns false if it does. If a vertex is not in the graph, it is considered as having no neighbor. In the same way, the null vertex has no neighbors.

17.72.5.23 `template<BIGRAPH_TEMPLATE > bool graph::VVBIGraph<BIGRAPH_TEMPLATE >::empty (const vertex1_t & v) const [inline]`

Test if a vertex has a neighbor.

Returns false if it does. If a vertex is not in the graph, it is considered as having no neighbor. In the same way, the null vertex has no neighbors.

Definition at line 2267 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::Vertex< VertexContent >::isNull()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

```
2268     {
2269         if (v.isNull())
2270             return true;
2271         typename neighborhood1_t::const_iterator it_found = this->findVertex(v);
2272         if (it_found == neighborhood1.end())
2273             return true;
2274         return it_found->second.edges.empty();
2275     }
```

17.72.5.24 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> bool graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::empty () const [inline]`

Test if there is any vertex in the graph.

Definition at line 942 of file vvbigraph.h.

```
942 { return this->no_vertex1() and this->no_vertex2(); }
```

17.72.5.25 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const_iterator1 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::end_vertex1 () const [inline]`

Returns an iterator on the end of the set of vertices of the graph.

Definition at line 1506 of file vvbigraph.h.

```
1506 { return neighborhood1.end(); }
```

17.72.5.26 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
iterator1 graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::end_vertex1 ()
[inline]`

Returns an iterator on the end of the set of vertices of the graph.

Definition at line 1470 of file vvbigraph.h.

Referenced by `graph::VVBIGraph< typename cell::content_t, typename
junction::content_t >::vertices1()`.

```
1470 { return neighborhood1.end(); }
```

17.72.5.27 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
const_iterator2 graph::VVBIGraph< Vertex1Content,
Vertex2Content, Edge1Content, Edge2Content_, compact
>::end_vertex2 () const [inline]`

Returns an iterator on the end of the set of vertices of the graph.

Definition at line 1510 of file vvbigraph.h.

```
1510 { return neighborhood2.end(); }
```

17.72.5.28 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
iterator2 graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::end_vertex2 ()
[inline]`

Returns an iterator on the end of the set of vertices of the graph.

Definition at line 1474 of file vvbigraph.h.

Referenced by `graph::VVBIGraph< typename cell::content_t, typename
junction::content_t >::vertices2()`.

```
1474 { return neighborhood2.end(); }
```


17.72.5.29 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> iterator2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::erase (iterator2 pos)`

Erase element at position `pos`.

Returns the iterator on the next element (as required for sequence containers)

17.72.5.30 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::iterator2 graph::VVBIGraph< BIGRAPH_TEMPLATE >::erase (iterator1 pos) [inline]`

Erase element at position `pos`.

Returns the iterator on the next element (as required for sequence containers)

Definition at line 1860 of file `vvbigraph.h`.

References `util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::base()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

```
1861     {
1862         eraseAllEdges(it);
1863         lookup1.erase(*it);
1864         return neighborhood1.erase(it.base());
1865     }
```

17.72.5.31 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> size_type graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::erase (const vertex2_t & v)`

Remove a vertex from the graph.

Parameters

`v` vertex to erase

Returns

1 if the vertex was erased, 0 else.

17.72.5.32 `template<BIGRAPH_TEMPLATE > VVBiGraph<
BIGRAPH_ARGS >::size_type graph::VVBiGraph<
BIGRAPH_TEMPLATE >::erase (const vertex1_t & v)
[inline]`

Remove a vertex from the graph.

Parameters

`v` vertex to erase

Returns

1 if the vertex was erased, 0 else.

Definition at line 1878 of file `vvbigraph.h`.

References `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges()`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1`, and `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteJunction()`.

```

1879     {
1880         if(eraseAllEdges(v))
1881         {
1882             typename lookup1_t::iterator it_found = lookup1.find(v);
1883             if(it_found == lookup1.end()) return 0;
1884             typename neighborhood1_t::iterator it = it_found->second;
1885             neighborhood1.erase(it);
1886             lookup1.erase(v);
1887             return 1;
1888         }
1889         return 0;
1890     }

```

17.72.5.33 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
void graph::VVBiGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::eraseAllEdges
(iterator2 it)`

Clear the neighborhood of a vertex.

All edges, to and from `*it` will be erased.

17.72.5.34 `template<BIGRAPH_TEMPLATE > void graph::VVBIGraph< BIGRAPH_TEMPLATE >::eraseAllEdges (iterator1 it)`
[inline]

Clear the neighborhood of a vertex.

All edges, to and from **it* will be erased.

Definition at line 1933 of file vvbigraph.h.

References `util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::base()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup2`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`.

```

1934     {
1935         const vertex1_t& v = *it_found;
1936         // Remove all the edges and their incoming counterpart
1937         for(typename edge_list1_t::iterator it_el = it_found.base()->second.edges.b
begin() ;
1938             it_el != it_found.base()->second.edges.end() ; ++it_el)
1939         {
1940             typename lookup2_t::iterator it_found = lookup2.find(it_el->target);
1941             if(it_found != lookup2.end())
1942                 it_found->second->second.in_edges.erase(v);
1943         }
1944         it_found.base()->second.edges.clear();
1945         it_found.base()->second.flagged = 0;
1946         // Remove the incoming edges
1947         in_edges1_t &in_edges = it_found.base()->second.in_edges;
1948         for( typename in_edges1_t::iterator it = in_edges.begin() ;
1949             it != in_edges.end() ; ++it )
1950         {
1951 #ifndef VVBIGRAPH_NO_EDGE_CACHE
1952             EdgeCache2 *cache = reinterpret_cast<EdgeCache2*>(it->cache);
1953             if(!cache or !cache->eraseEdge())
1954             {
1955 #endif
1956                 neighbor2_found_t found = findInVertex(*it, v);
1957                 found.neighborhood->edges.erase(found.it);
1958                 if(found.neighborhood->flagged && *found.neighborhood->flagged == v) fo
und.neighborhood->flagged = 0;
1959 #ifndef VVBIGRAPH_NO_EDGE_CACHE
1960             }
1961 #endif
1962         }
1963         in_edges.clear();
1964     }

```

17.72.5.35 `template<BIGRAPH_TEMPLATE > void graph::VVBIGraph< BIGRAPH_TEMPLATE >::eraseAllEdges ()` **[inline]**

Clear all the edges in the graph.

Example:

```

vvgraph S;
// Initialize S
vvgraph::size_type s1 = S.size(); // Save the number of vertices
S.eraseAllEdges();
assert(s1 == S.size()); // The number of vertices didn't change
forall_named(const vertex1_t& v, S, vertex1)
{
    assert(S.empty(v)); // No neighbor
}
forall_named(const vertex2_t& v, S, vertex2)
{
    assert(S.empty(v)); // No neighbor
}

```

Definition at line 2050 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood2`.

Referenced by `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::erase()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges()`.

```

2051     {
2052         for(typename neighborhood1_t::iterator it = neighborhood1.begin() ;
2053             it != neighborhood1.end() ; ++it)
2054         {
2055             it->second.edges.clear();
2056             it->second.in_edges.clear();
2057             it->second.flagged = 0;
2058         }
2059         for(typename neighborhood2_t::iterator it = neighborhood2.begin() ;
2060             it != neighborhood2.end() ; ++it)
2061         {
2062             it->second.edges.clear();
2063             it->second.in_edges.clear();
2064             it->second.flagged = 0;
2065         }
2066     }

```

17.72.5.36 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> bool graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges (const vertex2_t & v)`

Clear the neighborhood of a vertex.

All edges, to and from `v` will be erased.

17.72.5.37 `template<BIGRAPH_TEMPLATE > bool graph::VVBIGraph< BIGRAPH_TEMPLATE >::eraseAllEdges (const vertex1_t & v) [inline]`

Clear the neighborhood of a vertex.

All edges, to and from *v* will be erased.

Definition at line 1909 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

```

1910     {
1911         typename neighborhood1_t::iterator it_found = this->findVertex(v);
1912         if(it_found != neighborhood1.end() )
1913         {
1914             eraseAllEdges(it_found);
1915             return true;
1916         }
1917         return false;
1918     }
```

17.72.5.38 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> circ_neighbor_iterator2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge (const vertex2_t & v, circ_neighbor_iterator2 pos)`

Erase the edge pointed to by the iterator.

17.72.5.39 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> neighbor_iterator2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge (const vertex2_t & v, neighbor_iterator2 pos)`

Erase the edge pointed to by the iterator.

17.72.5.40 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::circ_neighbor_iterator2 graph::VVBIGraph< BIGRAPH_TEMPLATE >::eraseEdge (const vertex1_t & v, circ_neighbor_iterator1 pos) [inline]`

Erase the edge pointed to by the iterator.

Definition at line 2658 of file `vvbigraph.h`.

References `util::CircIterator< ForwardIterator >::end()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge()`, and `util::CircIterator< ForwardIterator >::isInitIterator()`.

```

2659     {
2660         neighbor_iterator1 nit = eraseEdge(v, pos.base());
2661         if(pos.isInitIterator(nit))
2662             return pos.end();
2663         return circ_neighbor_iterator1(pos, nit);
2664     }

```

17.72.5.41 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::neighbor_iterator2 graph::VVBIGraph< BIGRAPH_TEMPLATE >::eraseEdge (const vertex1_t & v, neighbor_iterator1 pos) [inline]`

Erase the edge pointed to by the iterator.

Definition at line 2618 of file `vvbigraph.h`.

References `util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::base()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood2`.

```

2619     {
2620         typename neighborhood1_t::iterator it_found = this->findVertex(v);
2621         if(it_found == neighborhood1.end())
2622             return neighbor_iterator1();
2623         const vertex2_t& n = *pos;
2624         typename neighborhood2_t::iterator it_found2 = this->findVertex(n);
2625         if(it_found2 == neighborhood2.end())
2626             return neighbor_iterator1();
2627         size_type result = it_found2->second.in_edges.erase(v);
2628         if(result)
2629         {
2630             typename edge_list1_t::iterator it = pos.base();
2631             return it_found->second.edges.erase(it);
2632         }
2633         return neighbor_iterator1();
2634     }

```

17.72.5.42 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> size_type graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge (const vertex2_t & src, const vertex1_t & tgt)`

Erase the edge from `src` to `tgt`.

Returns

True if successful.

17.72.5.43 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::size_type graph::VVBIGraph< BIGRAPH_TEMPLATE >::eraseEdge (const vertex1_t & src, const vertex2_t & tgt) [inline]`

Erase the edge from src to tgt.

Returns

True if successful.

Definition at line 2678 of file vvbigraph.h.

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it, graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup2, and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood.

Referenced by graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge().

```

2679     {
2680         neighbor1_found_t found = this->findInVertex(v, n);
2681         if(!found)
2682             return false;
2683         if(found.neighborhood->flagged && *found.neighborhood->flagged == n) found.
neighborhood->flagged = 0;
2684         found.neighborhood->edges.erase(found.it);
2685         typename lookup2_t::iterator it_found = lookup2.find(n);
2686         if(it_found != lookup2.end())
2687             it_found->second->second.in_edges.erase(v);
2688         return true;
2689     }
```

17.72.5.44 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const_iterator2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::find (const vertex2_t & v) const [inline]`

Find the vertex v in the graph.

Returns

An iterator on v if found, or end()

Definition at line 1536 of file vvbigraph.h.

```
1536 { return this->findVertex(v); }
```

17.72.5.45 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const_iterator1 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::find (const vertex1_t & v) const [inline]`

Find the vertex *v* in the graph.

Returns

An iterator on *v* if found, or end()

Definition at line 1530 of file vvbigraph.h.

```
1530 { return this->findVertex(v); }
```

17.72.5.46 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> iterator2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::find (const vertex2_t & v) [inline]`

Find the vertex *v* in the graph.

Returns

An iterator on *v* if found, or end()

Definition at line 1523 of file vvbigraph.h.

```
1523 { return this->findVertex(v); }
```

17.72.5.47 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> iterator1 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::find (const vertex1_t & v) [inline]`

Find the vertex *v* in the graph.

Returns

An iterator on v if found, or end()

Definition at line 1517 of file vvbigraph.h.

```
1517 { return this->findVertex(v); }
```

17.72.5.48 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const_neighbor_iterator2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findIn (const vertex2_t & v, const vertex1_t & n) const`

Find the vertex n in the neighborhood of v .

Returns

- An iterator on n in the neighborhood of v , if found
- An iterator on the end of the neighborhood if n is not found in v
- Otherwise, the result is undefined

17.72.5.49 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::const_neighbor_iterator2 graph::VVBIGraph< BIGRAPH_TEMPLATE >::findIn (const vertex1_t & v, const vertex2_t & n) const [inline]`

Find the vertex n in the neighborhood of v .

Returns

- An iterator on n in the neighborhood of v , if found
- An iterator on the end of the neighborhood if n is not found in v
- Otherwise, the result is undefined

Definition at line 2598 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`.

```
2599 {
2600     const_neighbor1_found_t found = findInVertex(v, n);
2601     if(found.neighborhood)
2602         return const_neighbor_iterator1(found.it);
2603     return const_neighbor_iterator1(found.it);
2604 }
```

17.72.5.50 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> neighbor_iterator2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findIn (const vertex2_t & v, const vertex1_t & n)`

Find the vertex *n* in the neighborhood of *v*.

Returns

- An iterator on *n* in the neighborhood of *v*, if found
- An iterator on the end of the neighborhood if *n* is not found in *v*
- Otherwise, the result is undefined

17.72.5.51 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::neighbor_iterator2 graph::VVBIGraph< BIGRAPH_TEMPLATE >::findIn (const vertex1_t & v, const vertex2_t & n) [inline]`

Find the vertex *n* in the neighborhood of *v*.

Returns

- An iterator on *n* in the neighborhood of *v*, if found
- An iterator on the end of the neighborhood if *n* is not found in *v*
- Otherwise, the result is undefined

Definition at line 2578 of file `vvbigraph.h`.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`.

```

2579     {
2580         neighbor1_found_t found = findInVertex(v, n);
2581         if(found.neighborhood)
2582             return neighbor_iterator1(found.it);
2583         return neighbor_iterator1();
2584     }

```

17.72.5.52 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const_neighbor2_found_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex (const vertex2_t & v, const vertex1_t & neighbor) const [protected]`

Constant version of findInVertex(const vertex_t&, const vertex_t&).

17.72.5.53 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::const_neighbor2_found_t graph::VVBIGraph< BIGRAPH_TEMPLATE >::findInVertex (const vertex1_t & v, const vertex2_t & neighbor) const [inline, protected]`

Constant version of findInVertex(const vertex_t&, const vertex_t&).

Definition at line 2516 of file vvbigraph.h.

References graph::Vertex< VertexContent >::cache, graph::Vertex< VertexContent >::cache_source, graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id, graph::Vertex< VertexContent >::id(), graph::Vertex< VertexContent >::isNull(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1.

```

2517     {
2518         if(v.isNull() || n.isNull())
2519             return const_neighbor1_found_t();
2520 #ifndef VVBIGRAPH_NO_EDGE_CACHE
2521         if(n.cache_source == v.id())
2522         {
2523             EdgeCache1 *cache = reinterpret_cast<EdgeCache1*>(n.cache);
2524             if(cache->graph_id() == graph_id)
2525             {
2526                 return const_neighbor1_found_t(cache->it(), cache->neighborhood());
2527             }
2528         }
2529 #endif // VVBIGRAPH_NO_EDGE_CACHE
2530         typename neighborhood1_t::const_iterator it_found = this->findVertex(v);
2531         if(it_found == neighborhood1.end())
2532             return const_neighbor1_found_t();
2533         const edge_list1_t &lst = it_found->second.edges;
2534         for(typename edge_list1_t::const_iterator it = lst.begin() ;
2535             it != lst.end() ; ++it)
2536         {
2537             if(it->target == n)
2538             {
2539                 return const_neighbor1_found_t(it, &it_found->second);
2540             }
2541         }
2542         return const_neighbor1_found_t(lst.end(), &it_found->second, false);
2543     }

```

17.72.5.54 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> neighbor2_found_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex (const vertex2_t & v, const vertex1_t & neighbor) [protected]`

Find a vertex neighbor in the neighborhood of v.

If 'v' is in the graph and 'neighbor' is in its neighborhood, the result is convertible to true, 'neighborhood' points toward the neighborhood of v and 'it' points toward the neighbor itself.

If 'v' is in the graph, but 'neighbor' is not in the neighborhood, the result is convertible to false, 'neighborhood' points toward the neighborhood of v and 'it' is neighborhood->edges.end().

If v is not in the graph, the result is convertible to false and the address of the stored neighborhood is 0.

17.72.5.55 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::neighbor2_found_t graph::VVBIGraph< BIGRAPH_TEMPLATE >::findInVertex (const vertex1_t & v, const vertex2_t & neighbor) [inline, protected]`

Find a vertex neighbor in the neighborhood of v.

If 'v' is in the graph and 'neighbor' is in its neighborhood, the result is convertible to true, 'neighborhood' points toward the neighborhood of v and 'it' points toward the neighbor itself.

If 'v' is in the graph, but 'neighbor' is not in the neighborhood, the result is convertible to false, 'neighborhood' points toward the neighborhood of v and 'it' is neighborhood->edges.end().

If v is not in the graph, the result is convertible to false and the address of the stored neighborhood is 0.

Definition at line 2454 of file vvbigraph.h.

References graph::Vertex< VertexContent >::cache, graph::Vertex< VertexContent >::cache_source, graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id, graph::Vertex< VertexContent >::id(), graph::Vertex< VertexContent >::isNull(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1.

Referenced by graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::edge(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge(), graph::VVBIGraph<

Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findIn(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::flag(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::prevTo(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::replace(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceBefore().

```

2455     {
2456         if(v.isNull() || n.isNull())
2457             return neighbor1_found_t();
2458 #ifndef VVBIGRAPH_NO_EDGE_CACHE
2459         if(n.cache_source == v.id())
2460         {
2461             EdgeCache1 *cache = reinterpret_cast<EdgeCache1*>(n.cache);
2462             if(cache->graph_id() == graph_id)
2463             {
2464                 return neighbor1_found_t(cache->it(), cache->neighborhood());
2465             }
2466         }
2467 #endif // VVBIGRAPH_NO_EDGE_CACHE
2468         typename neighborhood1_t::iterator it_found = this->findVertex(v);
2469         if(it_found == neighborhood1.end())
2470             return neighbor1_found_t();
2471         edge_list1_t &lst = it_found->second.edges;
2472         for(typename edge_list1_t::iterator it = lst.begin() ;
2473             it != lst.end() ; ++it)
2474         {
2475             if(it->target == n)
2476             {
2477                 return neighbor1_found_t(it, &it_found->second);
2478             }
2479         }
2480         return neighbor1_found_t(lst.end(), &it_found->second, false);
2481     }

```

17.72.5.56 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> neighborhood2_t::const_iterator graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex (const vertex2_t & v) const` **[protected]**

Constant version of [findVertex\(const vertex2_t&\)](#).

17.72.5.57 `template<BIGRAPH_TEMPLATE > VVBIGraph<
BIGRAPH_ARGS >::neighborhood2_t::const_iterator
graph::VVBIGraph< BIGRAPH_TEMPLATE >::findVertex (const
vertex1_t & v) const [inline, protected]`

Constant version of [findVertex\(const vertex1_t&\)](#).

Definition at line 2408 of file vvbigraph.h.

References `graph::Vertex< VertexContent >::cache`, `graph::Vertex< VertexContent >::cache_source`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

```

2409     {
2410 #ifndef VVBIGRAPH_NO_VERTEX_CACHE
2411     if(v.cache)
2412     {
2413         if(v.cache_source == 0)
2414         {
2415             VertexCache1* cache = reinterpret_cast<VertexCache1*>(v.cache);
2416             if(cache->graph_id() == graph_id)
2417             {
2418                 return cache->neighborhood();
2419             }
2420         }
2421     }
2422 #endif // VVBIGRAPH_NO_VERTEX_CACHE
2423     typename lookup1_t::const_iterator it_found = lookup1.find(v);
2424     if(it_found != lookup1.end())
2425         return it_found->second;
2426     return neighborhood1.end();
2427 }
```

17.72.5.58 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
neighborhood2_t::iterator graph::VVBIGraph< Vertex1Content,
Vertex2Content, Edge1Content, Edge2Content_, compact
>::findVertex (const vertex2_t & v) [protected]`

Find a vertex in the graph and returns the iterator on it.

Goal is to introduce a new optimization. When iterating over the vertices of the graph, they will cache their own iterator. So accessing their neighborhood would be constant time.

17.72.5.59 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::neighborhood2_t::iterator graph::VVBIGraph< BIGRAPH_TEMPLATE >::findVertex (const vertex1_t & v) [inline, protected]`

Find a vertex in the graph and returns the iterator on it.

Goal is to introduce a new optimization. When iterating over the vertices of the graph, they will cache their own iterator. So accessing their neighborhood would be constant time.

Definition at line 2362 of file vvbigraph.h.

References `graph::Vertex< VertexContent >::cache`, `graph::Vertex< VertexContent >::cache_source`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

Referenced by `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::anyIn()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::clear()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::contains()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::empty()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge()`, `graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::find()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::flagged()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iAnyIn()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iEmpty()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iNeighbors()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertEdge()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iValence()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors()`, `graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::reference()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::replace()`, `graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::source()`, `graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::target()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::valence()`.

```
2363     {
2364 #ifndef VVBIGRAPH_NO_VERTEX_CACHE
```

```

2365         if(v.cache)
2366         {
2367             if(v.cache_source == 0)
2368             {
2369                 VertexCache1* cache = reinterpret_cast<VertexCache1*>(v.cache);
2370                 if(cache->graph_id() == graph_id)
2371                 {
2372                     return cache->neighborhood();
2373                 }
2374             }
2375         }
2376 #endif // VVBIGRAPH_NO_VERTEX_CACHE
2377         typename lookup1_t::const_iterator it_found = lookup1.find(v);
2378         if(it_found != lookup1.end())
2379             return it_found->second;
2380         return neighborhood1.end();
2381     }

```

17.72.5.60 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> bool graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::flag (const vertex2_t & src, const vertex1_t & neighbor)`

Flag a neighbor of `src` for quick retrieval.

17.72.5.61 `template<BIGRAPH_TEMPLATE > bool graph::VVBiGraph< BIGRAPH_TEMPLATE >::flag (const vertex1_t & src, const vertex2_t & neighbor) [inline]`

Flag a neighbor of `src` for quick retrieval.

Definition at line 2969 of file `vvbigraph.h`.

References `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it`, and `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`.

```

2970     {
2971         neighbor1_found_t found = findInVertex(src, neighbor);
2972         if(!found)
2973             return false;
2974         found.neighborhood->flagged = &(found.it->target);
2975         return true;
2976     }

```


17.72.5.62 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const vertex1_t& graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::flagged (const vertex2_t & src) const`

Return the flagged neighbor of `src`.

17.72.5.63 `template<BIGRAPH_TEMPLATE > const VVBIGraph< BIGRAPH_ARGS >::vertex1_t & graph::VVBIGraph< BIGRAPH_TEMPLATE >::flagged (const vertex1_t & src) const [inline]`

Return the flagged neighbor of `src`.

Definition at line 2990 of file `vvbigraph.h`.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`, and `graph::Vertex< VertexContent >::null`.

```

2991     {
2992         typename neighborhood1_t::const_iterator it_found = this->findVertex(src);
2993         if(it_found == neighborhood1.end() or it_found->second.flagged == 0)
2994             return vertex2_t::null;
2995         return *it_found->second.flagged;
2996     }
```

17.72.5.64 `template<BIGRAPH_TEMPLATE > const VVBIGraph< BIGRAPH_ARGS >::vertex1_t & graph::VVBIGraph< BIGRAPH_TEMPLATE >::get_vertex1 (size_type idx) const [inline]`

Return the vertex of type 1 of index `idx`.

Note

This is a convenience function whose complexity is $O(idx)$

Returns

The element `idx` position after the first one (using iterators)

Definition at line 2200 of file `vvbigraph.h`.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

```

2201     {
2202         typename neighborhood1_t::const_iterator it = neighborhood1.begin();
2203         std::advance(it, idx);
2204         return it->first;
2205     }

```

17.72.5.65 `template<BIGRAPH_TEMPLATE > const VVBiGraph< BIGRAPH_ARGS >::vertex2_t & graph::VVBiGraph< BIGRAPH_TEMPLATE >::get_vertex2 (size_type idx) const [inline]`

Return the vertex of type 2 of index *idx*.

Note

This is a convenience function whose complexity is $O(idx)$

Returns

The element *idx* position after the first one (using iterators)

Definition at line 2209 of file `vvbigraph.h`.

References `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood2`.

```

2210     {
2211         typename neighborhood2_t::const_iterator it = neighborhood2.begin();
2212         std::advance(it, idx);
2213         return it->first;
2214     }

```

17.72.5.66 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const vertex1_t& graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iAnyIn (const vertex2_t & v) const`

Returns any incoming neighbor of *v*.

Returns

A vertex in the incoming neighborhood of *v*, or a null vertex if *v* is not in the graph or *v* has no incoming neighbor.

17.72.5.67 `template<BIGRAPH_TEMPLATE > const VVBIGraph<
BIGRAPH_ARGS >::vertex1_t & graph::VVBIGraph<
BIGRAPH_TEMPLATE >::iAnyIn (const vertex1_t & v) const
[inline]`

Returns any incoming neighbor of v.

Returns

A vertex in the incoming neighborhood of v, or a null vertex if v is not in the graph or v has no incoming neighbor.

Definition at line 2290 of file vvbigraph.h.

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex(), graph::Vertex< VertexContent >::isNull(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1, and graph::Vertex< VertexContent >::null.

Referenced by algorithms::TriangleGrowth::readOBJS().

```
2291     {
2292         if (v.isNull())
2293             return vertex2_t::null; //vertex_t(0);
2294         typename neighborhood1_t::const_iterator it_found = this->findVertex(v);
2295         if (it_found == neighborhood1.end() || it_found->second.in_edges.empty())
2296             return vertex2_t::null; //vertex_t(0);
2297         const in_edges1_t& lst = it_found->second.in_edges;
2298         return *lst.begin();
2299     }
```

17.72.5.68 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
bool graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::iEmpty (const vertex2_t
& v) const`

Test if a vertex has an incoming neighbor.

Returns false if it does. If a vertex is not in the graph, it is considered as having no incoming neighbor. In the same way, the null vertex has no incoming neighbors.

17.72.5.69 `template<BIGRAPH_TEMPLATE > bool graph::VVBIGraph<
BIGRAPH_TEMPLATE >::iEmpty (const vertex1_t & v) const
[inline]`

Test if a vertex has an incoming neighbor.

Returns false if it does. If a vertex is not in the graph, it is considered as having no incoming neighbor. In the same way, the null vertex has no incoming neighbors.

Definition at line 2339 of file vvbigraph.h.

References `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::Vertex< VertexContent >::isNull()`, and `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

Referenced by algorithms:`TriangleGrowth::readOBJs()`.

```

2340     {
2341         if(v.isNull())
2342             return true;
2343         typename neighborhood1_t::const_iterator it_found = this->findVertex(v);
2344         if(it_found == neighborhood1.end())
2345             return true;
2346         return it_found->second.in_edges.empty();
2347     }

```

17.72.5.70 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> inneighbor_iterator2_range graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iNeighbors (const vertex2_t & v) const`

Return the range of incoming neighbors of `v`.

Iterating over `iNeighbors` will go through all the vertices having an edge toward `v`.

17.72.5.71 `template<BIGRAPH_TEMPLATE > VVBigraph< BIGRAPH_ARGS >::ineighbor_iterator2_range graph::VVBigraph< BIGRAPH_TEMPLATE >::iNeighbors (const vertex1_t & v) const [inline]`

Return the range of incoming neighbors of `v`.

Iterating over `iNeighbors` will go through all the vertices having an edge toward `v`.

Definition at line 3265 of file vvbigraph.h.

References `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::Vertex< VertexContent >::isNull()`, `util::make_range()`, and `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

Referenced by algorithms:`TriangleGrowth::readOBJs()`.

```

3266     {
3267         inneighbor_iterator1_range result;
3268         if(v.isNull())
3269             return result;
3270         typename neighborhood1_t::const_iterator it_found = this->findVertex(v);
3271         if(it_found == neighborhood1.end())
3272             return result;

```

```

3273         const in_edges1_t &lst = it_found->second.in_edges;
3274         result = util::make_range(ineighbor_iterator1(lst.begin()),
3275                                   ineighbor_iterator1(lst.end()));
3276         return result;
3277     }

```

17.72.5.72 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> void graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insert (iterator2 first, iterator2 last)`

Insert all the vertices from first to last-1 in the graph.

17.72.5.73 `template<BIGRAPH_TEMPLATE > void graph::VVBIGraph< BIGRAPH_TEMPLATE >::insert (iterator1 first, iterator1 last) [inline]`

Insert all the vertices from first to last-1 in the graph.

Definition at line 2163 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insert()`.

```

2164     {
2165         for(iterator1 it = first ; it != last ; ++it)
2166         {
2167             insert(*it);
2168         }
2169     }

```

17.72.5.74 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> iterator2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insert (iterator2 pos, const vertex2_t & v)`

Insert a new vertex in the graph.

The iterator is ignored ...

17.72.5.75 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::iterator2 graph::VVBIGraph< BIGRAPH_TEMPLATE >::insert (iterator1 pos, const vertex1_t & v) [inline]`

Insert a new vertex in the graph.

The iterator is ignored ...

Definition at line 2150 of file vvbigraph.h.

References `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insert()`.

```
2151     {
2152         return this->insert(v);
2153     }
```

17.72.5.76 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> iterator2 graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insert (const vertex2_t & v)`

Insert a new vertex in the graph.

Returns

An iterator on the inserted vertex.

17.72.5.77 `template<BIGRAPH_TEMPLATE > VVBigraph< BIGRAPH_ARGS >::iterator2 graph::VVBigraph< BIGRAPH_TEMPLATE >::insert (const vertex1_t & v) [inline]`

Insert a new vertex in the graph.

Returns

An iterator on the inserted vertex.

Definition at line 2084 of file vvbigraph.h.

References `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`, `util::tie()`, and `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateVertexCache()`.

Referenced by `algorithms::TriangleGrowth::init()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insert()`, and `algorithms::TriangleGrowth::readOBJS()`.

```
2085     {
2086         typename lookup1_t::iterator it_found;
2087         bool inserted;
```

17.72 graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact > Class Template Reference 761

```
2088     util::tie(it_found, inserted) = lookup1.insert(std::make_pair(v, typename n
neighborhood1_t::iterator()));
2089     if(inserted)
2090     {
2091         neighborhood1.push_front(std::make_pair(v, single_neighborhood1_t()));
2092         typename neighborhood1_t::iterator it = neighborhood1.begin();
2093         it_found->second = it;
2094         updateVertexCache(it->first, it);
2095         return it;
2096     }
2097     return it_found->second;
2098
2099     //neighborhood1.push_front(std::make_pair(v, single_neighborhood1_t()));
2100     //typename neighborhood1_t::iterator it = neighborhood1.begin();
2101     //lookup1[v] = it;
2102     //updateVertexCache(it->first, it);
2103     //return it;
2104 }
```

17.72.5.78 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> edge2_t graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertEdge (const vertex2_t & src, const vertex1_t & tgt)`

Insert a new edge in the graph, without ordering.

If `new_neighbor` is already in the neighborhood of `v`, the insertion fails.

Returns

The just created edge if everything succeed, or a null edge.

17.72.5.79 `template<BIGRAPH_TEMPLATE > VVBigraph< BIGRAPH_ARGS >::edge2_t graph::VVBigraph< BIGRAPH_TEMPLATE >::insertEdge (const vertex1_t & src, const vertex2_t & tgt) [inline]`

Insert a new edge in the graph, without ordering.

If `new_neighbor` is already in the neighborhood of `v`, the insertion fails.

Returns

The just created edge if everything succeed, or a null edge.

Definition at line 3096 of file `vvbigraph.h`.

References `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id`,

graph::Vertex< VertexContent >::id(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertInEdge(), graph::Vertex< VertexContent >::isNull(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1, util::tie(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateEdgeCache().

Referenced by algorithms::TriangleGrowth::init(), and algorithms::TriangleGrowth::readOBJS().

```

3098     {
3099         if(v.isNull() || new_neighbor.isNull())
3100             return edge1_t();
3101         typename neighborhood1_t::iterator it_found = this->findVertex(v);
3102         if(it_found == neighborhood1.end())
3103             return edge1_t();
3104         ineighbor_iterator2 it_in;
3105         bool inserted;
3106         util::tie(it_in, inserted) = insertInEdge(v, new_neighbor);
3107         if(!inserted)
3108             return edge1_t();
3109         edge_list1_t &lst = it_found->second.edges;
3110         lst.push_front(neighbor1_t(new_neighbor, Edge1Content(), it_found->second,
graph_id));
3111         typename edge_list1_t::iterator it = lst.begin();
3112         updateEdgeCache(v, it, *it_in);
3113         return edge1_t(v.id(), new_neighbor.id(), &*it);
3114     }

```

17.72.5.80 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> std::pair<ineighbor_iterator1,bool> graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertInEdge (const vertex2_t & v, const vertex1_t & new_neighbor) [protected]`

Insert v in the in_edges of new_neighbor and return true if the insertion was actually done, or false if v was already there.

17.72.5.81 `template<BIGRAPH_TEMPLATE > std::pair< typename VVBIGraph< BIGRAPH_ARGS >::ineighbor_iterator1, bool > graph::VVBIGraph< BIGRAPH_TEMPLATE >::insertInEdge (const vertex1_t & v, const vertex2_t & new_neighbor) [inline, protected]`

Insert v in the in_edges of new_neighbor and return true if the insertion was actually done, or false if v was already there.

Definition at line 2720 of file vvbigraph.h.

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup2.

17.72 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact > Class Template Reference 763

Referenced by graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertEdge(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::replace(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceBefore().

```
2721     {
2722         typename lookup2_t::iterator found = lookup2.find(new_neighbor);
2723         if(found == lookup2.end())
2724             return std::make_pair(ineighbor_iterator2(), false);
2725         return found->second->second.in_edges.insert(v);
2726     }
```

17.72.5.82 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> size_type graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::iValence (const vertex2_t & v) const`

Returns the number of incoming neighbors of v.

The incoming neighbors are the set of vertices with on edge ending on v.

17.72.5.83 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::size_type graph::VVBIGraph< BIGRAPH_TEMPLATE >::iValence (const vertex1_t & v) const [inline]`

Returns the number of incoming neighbors of v.

The incoming neighbors are the set of vertices with on edge ending on v.

Definition at line 2316 of file vvbigraph.h.

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex(), graph::Vertex< VertexContent >::isNull(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1.

Referenced by algorithms::TriangleGrowth::readOBJS().

```
2317     {
2318         if(v.isNull())
2319             return 0;
2320         typename neighborhood1_t::const_iterator it_found = this->findVertex(v);
2321         if(it_found == neighborhood1.end())
2322             return 0;
2323         return it_found->second.in_edges.size();
2324     }
```

17.72.5.84 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
size_type graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::nb_vertices1 () const
[inline]`

Return the number of vertices on the graph.

Definition at line 921 of file vvbigraph.h.

Referenced by `graph::VVBIGraph< typename cell::content_t, typename
junction::content_t >::size()`.

```
921 { return neighborhood1.size(); }
```

17.72.5.85 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
size_type graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::nb_vertices2 () const
[inline]`

Return the number of vertices on the graph.

Definition at line 925 of file vvbigraph.h.

Referenced by `graph::VVBIGraph< typename cell::content_t, typename
junction::content_t >::size()`.

```
925 { return neighborhood2.size(); }
```

17.72.5.86 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
circ_neighbor_iterator2_range graph::VVBIGraph<
Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_,
compact >::neighbors (const vertex2_t & v, const vertex1_t & n)`

Return the range of neighbors of `v` starting from `n`.

17.72.5.87 `template<BIGRAPH_TEMPLATE > VVBIGraph<
BIGRAPH_ARGS >::circ_neighbor_iterator2_range
graph::VVBIGraph< BIGRAPH_TEMPLATE >::neighbors (const
vertex1_t & v, const vertex2_t & n) [inline]`

Return the range of neighbors of `v` starting from `n`.

Definition at line 3233 of file vvbigraph.h.

17.72 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact > Class Template Reference 765

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex(), graph::Vertex< VertexContent >::isNull(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it, util::make_range(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood.

```

3234     {
3235         circ_neighbor_iterator1_range result;
3236         if (v.isNull())
3237             return result;
3238         neighbor1_found_t found = findInVertex(v, n);
3239         if (!found)
3240             return result;
3241         edge_list1_t &lst = found.neighborhood->edges;
3242         result = util::make_range(circ_neighbor_iterator1(lst.begin(), lst.end(), f
ound.it),
3243                                 circ_neighbor_iterator1(lst.begin(), lst.end()));
3244         return result;
3245     }

```

17.72.5.88 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const_circ_neighbor_iterator2_range graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors (const vertex2_t & v, const vertex1_t & n) const`

Return the constant range of neighbors of v starting from n.

17.72.5.89 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::const_circ_neighbor_iterator2_range graph::VVBIGraph< BIGRAPH_TEMPLATE >::neighbors (const vertex1_t & v, const vertex2_t & n) const [inline]`

Return the constant range of neighbors of v starting from n.

Definition at line 3201 of file vvbigraph.h.

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex(), graph::Vertex< VertexContent >::isNull(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it, util::make_range(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood.

```

3202     {

```

```

3203         const_circ_neighbor_iterator1_range result;
3204         if(v.isNull())
3205             return result;
3206         const_neighbor1_found_t found = findInVertex(v, n);
3207         if(!found)
3208             return result;
3209         const edge_list1_t &lst = found.neighborhood->edges;
3210         result = util::make_range(const_circ_neighbor_iterator1(lst.begin(), lst.en
3211 d()), found.it),
3212                                     const_circ_neighbor_iterator1(lst.begin(), lst.en
3213 d()));
3212         return result;
3213     }

```

17.72.5.90 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> neighbor_iterator2_range graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors (const vertex2_t & v)`

Return the range of neighbors of v.

17.72.5.91 `template<BIGRAPH_TEMPLATE > VVBiGraph< BIGRAPH_ARGS >::neighbor_iterator2_range graph::VVBiGraph< BIGRAPH_TEMPLATE >::neighbors (const vertex1_t & v) [inline]`

Return the range of neighbors of v.

Definition at line 3171 of file vvbigraph.h.

References `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::Vertex< VertexContent >::isNull()`, `util::make_range()`, and `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

```

3172     {
3173         neighbor_iterator1_range result;
3174         if(v.isNull())
3175             return result;
3176         typename neighborhood1_t::iterator it_found = this->findVertex(v);
3177         if(it_found == neighborhood1.end())
3178             return result;
3179         edge_list1_t &lst = it_found->second.edges;
3180         result = util::make_range(neighbor_iterator1(lst.begin()),
3181 neighbor_iterator1(lst.end()));
3181         return result;
3182     }

```

17.72.5.92 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const_neighbor_iterator2_range graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighbors (const vertex2_t & v) const`

Return the constant range of neighbors of v.

17.72.5.93 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::const_neighbor_iterator2_range graph::VVBIGraph< BIGRAPH_TEMPLATE >::neighbors (const vertex1_t & v) const [inline]`

Return the constant range of neighbors of v.

Definition at line 3140 of file vvbigraph.h.

References graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex(), graph::Vertex< VertexContent >::isNull(), util::make_range(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1.

Referenced by vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >) >::border(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::calcQuads(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >) >::connectFromJunctions(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >) >::divideCell(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCellContour(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawSimplifiedCell(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawWalledCell(), vvcomplex::findCenter(), vvcomplex::FindOppositeWall(), and algorithms::TriangleGrowth::readOBJs().

```

3141     {
3142         const_neighbor_iterator1_range result;
3143         if (v.isNull())
3144             return result;
3145         typename neighborhood1_t::const_iterator it_found = this->findVertex(v);
3146         if (it_found == neighborhood1.end())
3147             return result;
3148         const edge_list1_t &lst = it_found->second.edges;

```

```

3149         result = util::make_range(const_neighbor_iterator1(lst.begin()),
3150                                   const_neighbor_iterator1(lst.end()));
3151     return result;
3152 }

```

17.72.5.94 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const vertex1_t& graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::nextTo(const vertex2_t & v, const vertex1_t & neighbor, unsigned int n = 1) const`

Returns the `n`th vertex after `neighbor` in the neighborhood of `v`.

Returns

The vertex found or a null vertex if there is any problem.

17.72.5.95 `template<BIGRAPH_TEMPLATE > const VVBiGraph< BIGRAPH_ARGS >::vertex1_t & graph::VVBiGraph< BIGRAPH_TEMPLATE >::nextTo(const vertex1_t & v, const vertex2_t & neighbor, unsigned int n = 1) const` [`inline`]

Returns the `n`th vertex after `neighbor` in the neighborhood of `v`.

Returns

The vertex found or a null vertex if there is any problem.

Definition at line 2846 of file `vvbigraph.h`.

References `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`, and `graph::Vertex< VertexContent >::null`.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::calcQuads()`, `algorithms::TriangleSurface::center3d()`, `algorithms::TriangleSurface::centerPosition()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::connectFromJunctions()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteJunction()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawCell()`, `tissue::Tissue<`

Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawWalledCell(), vvcomplex::FindOppositeWall(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::mergeCells(), algorithms::TriangleSurface::position(), algorithms::TriangleGrowth::readOBJS(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::splitWall(), and vvcomplex::testDivisionOnVertices().

```

2849     {
2850         const_neighbor1_found_t found = findInVertex(v, ref);
2851         if(!found)
2852             return vertex2_t::null;//vertex_t(0);
2853         typename edge_list1_t::const_iterator it = found.it;
2854         const edge_list1_t& lst = found.neighborhood->edges;
2855         for(unsigned int i = 0 ; i < n ; ++i)
2856         {
2857             ++it;
2858             if(it == lst.end())
2859                 it = lst.begin();
2860         }
2861         return it->target;
2862     }

```

17.72.5.96 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> bool graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::no_vertex1 () const [inline]`

Test if there is any vertex in the graph.

Definition at line 934 of file vvbigraph.h.

Referenced by graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::empty().

```

934 { return neighborhood1.empty(); }

```

17.72.5.97 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> bool graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::no_vertex2 () const [inline]`

Test if there is any vertex in the graph.

Definition at line 938 of file vvbigraph.h.

Referenced by `graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::empty()`.

```
938 { return neighborhood2.empty(); }
```

17.72.5.98 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS > & graph::VVBIGraph< BIGRAPH_TEMPLATE >::operator= (const VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact > & other) [inline]`

Copy the graph into another graph.

Definition at line 3491 of file `vvbigraph.h`.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup2`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood2`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateEdgeCache()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateVertexCache()`.

```
3492 {
3493     // Copy the structure
3494     neighborhood1 = other.neighborhood1;
3495     neighborhood2 = other.neighborhood2;
3496     lookup1.clear();
3497     lookup2.clear();
3498     for(typename neighborhood1_t::iterator it1 = neighborhood1.begin() ;
3499         it1 != neighborhood1.end() ; ++it1)
3500         lookup1[it1->first] = it1;
3501     for(typename neighborhood2_t::iterator it2 = neighborhood2.begin() ;
3502         it2 != neighborhood2.end() ; ++it2)
3503         lookup2[it2->first] = it2;
3504     // And reconstruct the cache for vertex type 1
3505     for(typename neighborhood1_t::iterator it1 = neighborhood1.begin() ;
3506         it1 != neighborhood1.end() ; ++it1)
3507     {
3508         const vertex1_t& src = it1->first;
3509         updateVertexCache(src, it1);
3510 #ifndef VVBIGRAPH_NO_EDGE_CACHE
3511         edge_list1_t& lst = it1->second.edges;
3512         for(typename edge_list1_t::iterator it1_e = lst.begin() ;
3513             it1_e != lst.end() ; ++it1_e)
3514         {
3515             // Find the incoming vertex corresponding to this edge
3516             typename lookup2_t::iterator it_found = lookup2.find(it1_e->target);
3517             const vertex1_t& iv = *(it_found->second->second.in_edges.find(src));
3518             updateEdgeCache(src, it1_e, iv);
3519             it1_e->set(&it1->second, graph_id);

```



```

3520     }
3521 #endif
3522     }
3523     // And reconstruct the cache for vertex type 2
3524     for(typename neighborhood2_t::iterator it2 = neighborhood2.begin() ;
3525         it2 != neighborhood2.end() ; ++it2)
3526     {
3527         const vertex2_t& src = it2->first;
3528         updateVertexCache(src, it2);
3529 #ifndef VVBIGRAPH_NO_EDGE_CACHE
3530         edge_list2_t& lst = it2->second.edges;
3531         for(typename edge_list2_t::iterator it2_e = lst.begin() ;
3532             it2_e != lst.end() ; ++it2_e)
3533         {
3534             // Find the incoming vertex corresponding to this edge
3535             typename lookup1_t::iterator it_found = lookup1.find(it2_e->target);
3536             const vertex2_t& iv = *(it_found->second->second.in_edges.find(src));
3537             updateEdgeCache(src, it2_e, iv);
3538             it2_e->set(&it2->second, graph_id);
3539         }
3540 #endif
3541     }
3542     return *this;
3543 }
```

17.72.5.99 `template<BIGRAPH_TEMPLATE> bool graph::VVBIGraph<BIGRAPH_TEMPLATE>::operator==(const VVBIGraph<Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact> &other) const` [`inline`]

Test equality for the graphs.

Two graphs are equal if they shared the same vertices and their neighborhood are equivalent.

Note

Current implementation do not consider invariance of neighborhood by rotation.

Definition at line 3405 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup2`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood2`.

```

3406     {
3407         if(neighborhood1.size() != other.neighborhood1.size() or
3408            neighborhood2.size() != other.neighborhood2.size())
3409             return false;
3410         for(typename neighborhood1_t::const_iterator it1 = neighborhood1.begin() ;
3411             it1 != neighborhood1.end() ; ++it1)
```

```

3412     {
3413         const vertex1_t& v1 = it1->first;
3414         typename lookup1_t::const_iterator it_found = other.lookup1.find(v1);
3415         if(it_found == other.lookup1.end())
3416             return false;
3417         if(it1->second.edges.empty())
3418         {
3419             if(!it_found->second->second.edges.empty())
3420                 return false;
3421         }
3422         else
3423         {
3424             const edge_list1_t& lst = it1->second.edges;
3425             const edge_list1_t& olst = it_found->second->second.edges;
3426             if(lst.size() != olst.size()) return false;
3427             const vertex2_t& v2 = lst.begin()->target;
3428             bool found = false;
3429             for(typename edge_list1_t::const_iterator it_olst = olst.begin() ;
3430                 it_olst != olst.end() ; ++it_olst)
3431             {
3432                 if(it_olst->target == v2)
3433                 {
3434                     found = true;
3435                     for(typename edge_list1_t::const_iterator it_lst = lst.begin() ;
3436                         it_lst != lst.end() ; ++it_lst)
3437                     {
3438                         if(it_lst->target != it_olst->target) return false;
3439                         ++it_olst;
3440                         if(it_olst == olst.end()) it_olst = olst.begin();
3441                     }
3442                     break;
3443                 }
3444             }
3445             if(!found) return false;
3446         }
3447     }
3448     for(typename neighborhood2_t::const_iterator it2 = neighborhood2.begin() ;
3449         it2 != neighborhood2.end() ; ++it2)
3450     {
3451         const vertex2_t& v2 = it2->first;
3452         typename lookup2_t::const_iterator it_found = other.lookup2.find(v2);
3453         if(it_found == other.lookup2.end())
3454             return false;
3455         if(it2->second.edges.empty())
3456         {
3457             if(!it_found->second->second.edges.empty())
3458                 return false;
3459         }
3460         else
3461         {
3462             const edge_list2_t& lst = it2->second.edges;
3463             const edge_list2_t& olst = it_found->second->second.edges;
3464             if(lst.size() != olst.size()) return false;
3465             const vertex1_t& v1 = lst.begin()->target;
3466             bool found = false;
3467             for(typename edge_list2_t::const_iterator it_olst = olst.begin() ;
3468                 it_olst != olst.end() ; ++it_olst)
3469             {
3470                 if(it_olst->target == v1)
3471                 {
3472                     found = true;
3473                     for(typename edge_list2_t::const_iterator it_lst = lst.begin() ;

```

```

3474             it_lst != lst.end() ; ++it_lst)
3475         {
3476             if(it_lst->target != it_olst->target) return false;
3477             ++it_olst;
3478             if(it_olst == olst.end()) it_olst = olst.begin();
3479         }
3480         break;
3481     }
3482 }
3483 if(!found) return false;
3484 }
3485 }
3486 return true;
3487 }
```

17.72.5.100 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const vertex1_t & graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::prevTo (const vertex2_t & v, const vertex1_t & ref, unsigned int n = 1) const`

Returns the nth vertex before ref in the neighborhood of v.

If ref is not in the neighborhood of v, return a null vertex.

17.72.5.101 `template<BIGRAPH_TEMPLATE > const VVBIGraph< BIGRAPH_ARGS >::vertex1_t & graph::VVBIGraph< BIGRAPH_TEMPLATE >::prevTo (const vertex1_t & v, const vertex2_t & ref, unsigned int n = 1) const [inline]`

Returns the nth vertex before ref in the neighborhood of v.

If ref is not in the neighborhood of v, return a null vertex.

Definition at line 2886 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`, and `graph::Vertex< VertexContent >::null`.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::calcQuads()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >) >::connectFromJunctions()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >) >::deleteJunction()`, `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellCon-`

tent, compact, LeafClass >::drawCell(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::drawWalledCell(), vvcomplex::findCenter(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::mergeCells(), and algorithms::TriangleGrowth::readOBJS().

```

2889     {
2890         const_neighbor1_found_t found = findInVertex(v, ref);
2891         if(!found)
2892             return vertex2_t::null;//vertex_t(0);
2893         typename edge_list1_t::const_iterator it = found.it;
2894         const edge_list1_t& lst = found.neighborhood->edges;
2895         for(unsigned int i = 0 ; i < n ; ++i)
2896         {
2897             if(it == lst.begin())
2898                 it = lst.end();
2899             --it;
2900         }
2901         return it->target;
2902     }

```

17.72.5.102 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const vertex2_t& graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::reference (vertex2_t v) const [inline]`

Get a reference on the vertex in the graph.

If the vertex do not exist in the graph, the result is totally undefined and the program might even crash.

Definition at line 910 of file vvbigraph.h.

```

911     {
912         typename neighborhood2_t::const_iterator found = this->
findVertex(v);
913         if(found != neighborhood2.end())
914             return found->first;
915         return vertex2_t::null;//vertex_t(0);
916     }

```

17.72.5.103 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const vertex1_t& graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::reference (vertex1_t v) const [inline]`

Get a reference on the vertex in the graph.

If the vertex do not exist in the graph, the result is totally undefined and the program might even crash.

Definition at line 897 of file vvbigraph.h.

Referenced by algorithms::TriangleSurface::center3d(), algorithms::TriangleSurface::centerPosition(), and algorithms::TriangleGrowth::init().

```

898         {
899             typename neighborhood1_t::const_iterator found = this->
findVertex(v);
900             if(found != neighborhood1.end())
901                 return found->first;
902             return vertex1_t::null;//vertex_t(0);
903         }

```

17.72.5.104 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> edge2_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::replace (const vertex2_t & v, const vertex1_t & neighbor, const vertex1_t & new_neighbor)`

Replace a vertex by another in a neighborhood.

Parameters

- ← *v* [Vertex](#) whose neighborhood is changed.
- ← *neighbor* [Vertex](#) to replace
- ← *new_neighbor* [Vertex](#) replacing neighbor

If *new_neighbor* is already in the neighborhood of *v*, then the operation fails and nothing is changed.

Returns

The edge between *v* and *new_neighbor* or a null edge if anything goes wrong.

17.72.5.105 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::edge2_t graph::VVBIGraph< BIGRAPH_TEMPLATE >::replace (const vertex1_t & v, const vertex2_t & neighbor, const vertex2_t & new_neighbor) [inline]`

Replace a vertex by another in a neighborhood.

Parameters

- ← *v* [Vertex](#) whose neighborhood is changed.

← *neighbor* [Vertex](#) to replace
 ← *new_neighbor* [Vertex](#) replacing *neighbor*

If *new_neighbor* is already in the neighborhood of *v*, then the operation fails and nothing is changed.

Returns

The edge between *v* and *new_neighbor* or a null edge if anything goes wrong.

Definition at line 2796 of file `vvbigraph.h`.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::Vertex< VertexContent >::id()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertInEdge()`, `graph::Vertex< VertexContent >::isNull()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it`, `util::tie()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateEdgeCache()`.

```

2799     {
2800         if(new_neighbor.isNull() || (neighbor == new_neighbor))
2801             return edge1_t();
2802         neighborhood1_found_t found = findInVertex(v, neighbor);
2803         if(!found)
2804             return edge1_t();
2805         typename neighborhood2_t::iterator n_found = this->findVertex(neighbor);
2806         neighborhood2_iterator it_in;
2807         bool inserted;
2808         util::tie(it_in, inserted) = insertInEdge(v, new_neighbor);
2809         if(!inserted)
2810             return edge1_t();
2811         n_found->second.in_edges.erase(v);
2812         found.it->target = new_neighbor;
2813         found.it->clear_edge();
2814         updateEdgeCache(v, found.it, *it_in);
2815         if(n_found->second.flagged and *n_found->second.flagged == neighbor) n_foun
d->second.flagged = 0;
2816         return edge1_t(v.id(), new_neighbor.id(), &*found.it);
2817     }

```

17.72.5.106 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> size_type graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::size () const [inline]`

Return the number of vertices on the graph.

Definition at line 929 of file `vvbigraph.h`.

```
929 { return this->nb_vertices1() + this->nb_vertices2(); }
```

17.72.5.107 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const vertex2_t & graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::source (const edge2_t & edge) const` **[inline]**

Return the source vertex of the edge.

Definition at line 1177 of file vvbigraph.h.

```
1178         {
1179             typename neighborhood2_t::const_iterator found = this->
findVertex(vertex2_t(edge.source()));
1180             if(found != neighborhood2.end())
1181                 return found->first;
1182             return vertex2_t::null;//vertex_t(0);
1183         }
```

17.72.5.108 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const vertex1_t & graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::source (const edge1_t & edge) const` **[inline]**

Return the source vertex of the edge.

Definition at line 1166 of file vvbigraph.h.

```
1167         {
1168             typename neighborhood1_t::const_iterator found = this->
findVertex(vertex1_t(edge.source()));
1169             if(found != neighborhood1.end())
1170                 return found->first;
1171             return vertex1_t::null;//vertex_t(0);
1172         }
```

17.72.5.109 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> edge2_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter (const vertex2_t & v, const vertex1_t & neighbor, const vertex1_t & new_neighbor)`

Insert neighbor in the neighborhood of v after reference.

If new_neighbor is already in the neighborhood of v, the insertion fails.

Returns

The just created edge if everything succeed, or a null edge.

17.72.5.110 `template<BIGRAPH_TEMPLATE > VVBiGraph<
BIGRAPH_ARGS >::edge2_t graph::VVBiGraph<
BIGRAPH_TEMPLATE >::spliceAfter (const vertex1_t & v,
const vertex2_t & neighbor, const vertex2_t & new_neighbor)
[inline]`

Insert neighbor in the neighborhood of v after reference.

If new_neighbor is already in the neighborhood of v, the insertion fails.

Returns

The just created edge if everything succeed, or a null edge.

Definition at line 3010 of file vvbigraph.h.

References `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id`, `graph::Vertex< VertexContent >::id()`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertInEdge()`, `graph::Vertex< VertexContent >::isNull()`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`, `util::tie()`, and `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateEdgeCache()`.

Referenced by `algorithms::TriangleGrowth::init()`, and `algorithms::TriangleGrowth::readOBJS()`.

```

3013     {
3014         if(new_neighbor.isNull())
3015             return edge1_t();
3016         neighbor1_found_t found = findInVertex(v, neighbor);
3017         if(!found)
3018             return edge1_t();
3019         inneighbor_iterator2 it_in;
3020         bool inserted;
3021         util::tie(it_in, inserted) = insertInEdge(v, new_neighbor);
3022         if(!inserted)
3023             return edge1_t();
3024         ++found.it;
3025         typename edge_list1_t::iterator new_edge_it = found.neighborhood->edges.ins
ert(found.it, neighbor1_t(new_neighbor, Edge1Content(), *found.neighborhood,
graph_id));
3026         updateEdgeCache(v, new_edge_it, *it_in);
3027         return edge1_t(v.id(), new_neighbor.id(), &*new_edge_it);
3028     }

```


17.72.5.111 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> edge2_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceBefore (const vertex2_t & v, const vertex1_t & neighbor, const vertex1_t & new_neighbor)`

Insert neighbor in the neighborhood of v before reference.

If `new_neighbor` is already in the neighborhood of v, the insertion fails.

Returns

The just created edge if everything succeed, or a null edge.

17.72.5.112 `template<BIGRAPH_TEMPLATE > VVBIGraph< BIGRAPH_ARGS >::edge2_t graph::VVBIGraph< BIGRAPH_TEMPLATE >::spliceBefore (const vertex1_t & v, const vertex2_t & neighbor, const vertex2_t & new_neighbor) [inline]`

Insert neighbor in the neighborhood of v before reference.

If `new_neighbor` is already in the neighborhood of v, the insertion fails.

Returns

The just created edge if everything succeed, or a null edge.

Definition at line 3054 of file `vvbigraph.h`.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id`, `graph::Vertex< VertexContent >::id()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertInEdge()`, `graph::Vertex< VertexContent >::isNull()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::it`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`, `util::tie()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateEdgeCache()`.

Referenced by algorithms::TriangleGrowth::readOBJS().

```

3057     {
3058         if(new_neighbor.isNull())
3059             return edge1_t();
3060         neighbor1_found_t found = findInVertex(v, neighbor);
3061         if(!found)
3062             return edge1_t();

```

```

3063         ineighbor_iterator2 it_in;
3064         bool inserted;
3065         util::tie(it_in, inserted) = insertInEdge(v, new_neighbor);
3066         if(!inserted)
3067             return edge1_t();
3068         typename edge_list1_t::iterator new_edge_it = found.neighborhood->edges.ins
ert(found.it, neighbor1_t(new_neighbor, Edge1Content(), *found.neighborhood,
graph_id));
3069         updateEdgeCache(v, new_edge_it, *it_in);
3070         return edge1_t(v.id(), new_neighbor.id(), &*new_edge_it);
3071     }

```

17.72.5.113 `template<BIGRAPH_TEMPLATE > void graph::VVBiGraph< BIGRAPH_TEMPLATE >::swap (VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact > & other) [inline]`

Swap the content of two graphs.

Definition at line 3546 of file vvbigraph.h.

References `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup2`, `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`, and `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood2`.

```

3547     {
3548         neighborhood1.swap(other.neighborhood1);
3549         neighborhood2.swap(other.neighborhood2);
3550         lookup1.swap(other.lookup1);
3551         lookup2.swap(other.lookup2);
3552         std::swap(graph_id, other.graph_id);
3553     }

```

17.72.5.114 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const vertex1_t& graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::target (const edge2_t & edge) const [inline]`

Return the target vertex of the edge.

Definition at line 1199 of file vvbigraph.h.

```

1200     {
1201         typename neighborhood1_t::const_iterator found = this->
findVertex(vertex1_t(edge.target()));

```

```

1202             if(found != neighborhood1.end())
1203                 return found->first;
1204             return vertex1_t::null;//vertex_t(0);
1205         }

```

17.72.5.115 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const vertex2_t& graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::target (const edge1_t & edge) const` **[inline]**

Return the target vertex of the edge.

Definition at line 1188 of file vvbigraph.h.

```

1189         {
1190             typename neighborhood2_t::const_iterator found = this->
findVertex(vertex2_t(edge.target()));
1191             if(found != neighborhood2.end())
1192                 return found->first;
1193             return vertex2_t::null;//vertex_t(0);
1194         }

```

17.72.5.116 `template<BIGRAPH_TEMPLATE > void graph::VVBIGraph< BIGRAPH_TEMPLATE >::undoInEdge (const vertex2_t & new_neighbor, inneighbor_iterator2 & it) [inline, protected]`

Undo the insertion of an in edge as a roll-back mechanism if the whole edge insertion fails.

Definition at line 2739 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup2`.

```

2741     {
2742         typename lookup2_t::iterator found = lookup2.find(new_neighbor);
2743         if(found == lookup2.end())
2744             return;
2745         found->second->second.in_edges.erase(it);
2746     }

```

17.72.5.117 `template<BIGRAPH_TEMPLATE > void graph::VVBIGraph< BIGRAPH_TEMPLATE >::undoInEdge (const vertex1_t & new_neighbor, inneighbor_iterator1 & it) [inline, protected]`

Undo the insertion of an in edge as a roll-back mechanism if the whole edge insertion fails.

Definition at line 2749 of file vvbigraph.h.

References `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1`.

```

2751     {
2752         typename lookup1_t::iterator found = lookup1.find(new_neighbor);
2753         if(found == lookup1.end())
2754             return;
2755         found->second->second.in_edges.erase(it);
2756     }

```

17.72.5.118 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> void graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateEdgeCache (const vertex2_t & v, typename edge_list2_t::iterator new_edge_it, const vertex2_t & in_edge) [protected]`

Update the edge cache information of vertex v in its neighborhood and the in_edge too.

17.72.5.119 `template<BIGRAPH_TEMPLATE > void graph::VVBigraph< BIGRAPH_TEMPLATE >::updateEdgeCache (const vertex1_t & v, typename edge_list1_t::iterator new_edge_it, const vertex1_t & in_edge) [inline, protected]`

Update the edge cache information of vertex v in its neighborhood and the in_edge too.

Definition at line 2759 of file vvbigraph.h.

References `graph::Vertex< VertexContent >::cache`, `graph::Vertex< VertexContent >::cache_source`, and `graph::Vertex< VertexContent >::id()`.

Referenced by `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertEdge()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::operator=()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::replace()`, `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter()`, and `graph::VVBigraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceBefore()`.

```

2762     {
2763         #ifndef VVBIGRAPH_NO_EDGE_CACHE
2764             // Caching in the neighbor
2765             neighbor1_t& neighbor = *new_edge_it;
2766             neighbor.setIt(new_edge_it);
2767             neighbor.target.cache_source = v.id();
2768             neighbor.target.cache = reinterpret_cast<void*>(&neighbor);
2769
2770             // Caching in the incoming edge source

```

```
2771         in_edge.cache_source = v.id();
2772         in_edge.cache = reinterpret_cast<void*>(&neighbor);
2773 #endif // VVBIGRAPH_NO_EDGE_CACHE
2774     }
```

17.72.5.120 `template<BIGRAPH_TEMPLATE > void graph::VVBIGraph< BIGRAPH_TEMPLATE >::updateVertexCache (const vertex2_t & v, typename neighborhood2_t::iterator it) [inline, protected]`

Update the vertex cache for vertex v.

Definition at line 2140 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id`.

```
2142     {
2143 #ifndef VVBIGRAPH_NO_VERTEX_CACHE
2144         it->second.update(graph_id, v, it);
2145 #endif
2146     }
```

17.72.5.121 `template<BIGRAPH_TEMPLATE > void graph::VVBIGraph< BIGRAPH_TEMPLATE >::updateVertexCache (const vertex1_t & v, typename neighborhood1_t::iterator it) [inline, protected]`

Update the vertex cache for vertex v.

Definition at line 2131 of file vvbigraph.h.

References `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id`.

Referenced by `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insert()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::operator=()`.

```
2133     {
2134 #ifndef VVBIGRAPH_NO_VERTEX_CACHE
2135         it->second.update(graph_id, v, it);
2136 #endif
2137     }
```

17.72.5.122 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> size_type graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::valence (const vertex2_t & v) const`

Returns the number of neighbors of v .

If v is not in the graph, the behavior is undefined.

17.72.5.123 `template<BIGRAPH_TEMPLATE > VVBiGraph< BIGRAPH_ARGS >::size_type graph::VVBiGraph< BIGRAPH_TEMPLATE >::valence (const vertex1_t & v) const [inline]`

Returns the number of neighbors of v .

If v is not in the graph, the behavior is undefined.

Definition at line 2244 of file vvbigraph.h.

References `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::Vertex< VertexContent >::isNull()`, and `graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::border()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteJunction()`, `vvcomplex::findCenter()`, `algorithms::TriangleGrowth::init()`, and `algorithms::TriangleGrowth::readOBJS()`.

```

2245     {
2246         if(v.isNull())
2247             return 0;
2248         typename neighborhood1_t::const_iterator it_found = this->findVertex(v);
2249         if(it_found == neighborhood1.end())
2250             return 0;
2251         return it_found->second.edges.size();
2252     }

```

17.72.5.124 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> range_vertex1 graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::vertices1 () [inline]`

Returns a range of vertices.

Definition at line 1488 of file vvbigraph.h.

```
1488 { return util::make_range(begin_vertex1(), end_vertex1()); }
```

17.72.5.125 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const_range_vertex1 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::vertices1 () const [inline]`

Returns a range of vertices.

Definition at line 1479 of file vvbigraph.h.

```
1479 { return util::make_range(begin_vertex1(), end_vertex1()); }
```

17.72.5.126 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> range_vertex2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::vertices2 () [inline]`

Returns a range of vertices.

Definition at line 1492 of file vvbigraph.h.

```
1492 { return util::make_range(begin_vertex2(), end_vertex2()); }
```

17.72.5.127 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> const_range_vertex2 graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::vertices2 () const [inline]`

Returns a range of vertices.

Definition at line 1483 of file vvbigraph.h.

```
1483 { return util::make_range(begin_vertex2(), end_vertex2()); }
```

17.72.6 Member Data Documentation

17.72.6.1 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> intptr_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::graph_id [protected]`

Unique id for this graph.

Definition at line 1847 of file vvbigraph.h.

Referenced by `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findInVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertEdge()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::operator=()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceAfter()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::spliceBefore()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::swap()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::updateVertexCache()`.

17.72.6.2 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> lookup1_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup1 [protected]`

Lookup hash map for vertex1's.

Definition at line 1838 of file vvbigraph.h.

Referenced by `graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::clear()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::count()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::erase()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::findVertex()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insert()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::operator=()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::operator==()`, `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::swap()`, and `graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::undoInEdge()`.

17.72.6.3 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> lookup2_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::lookup2 [protected]`

Lookup hash map for vertex2's.

Definition at line 1842 of file vvbigraph.h.

Referenced by graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::clear(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::insertInEdge(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::operator=(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::operator==(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::swap(), and graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::undoInEdge().

17.72.6.4 `template<typename Vertex1Content, typename Vertex2Content, typename Edge1Content = _EmptyBiEdgeContent, typename Edge2Content_ = Edge1Content, bool compact = false> neighborhood1_t graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::neighborhood1 [protected]`

Main data structure containing the neighborhood of the vertex1's.

Definition at line 1829 of file vvbigraph.h.

Referenced by graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::any_vertex1(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::anyIn(), graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::begin_vertex1(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::clear(), graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::clear(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::contains(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::empty(), graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::end_vertex1(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::erase(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseEdge(), graph::VVBIGraph< Vertex1Content, Vertex2Content,

```

Edge1Content, Edge2Content_, compact >::findInVertex(), graph::VVBIGraph<
Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, com-
pact >::findVertex(), graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::flagged(), graph::VVBIGraph<
Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact
>::get_vertex1(), graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::iAnyIn(), graph::VVBIGraph<
Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, com-
pact >::iEmpty(), graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::iNeighbors(), graph::VVBIGraph<
Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact
>::insert(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content,
Edge2Content_, compact >::insertEdge(), graph::VVBIGraph< Vertex1Content,
Vertex2Content, Edge1Content, Edge2Content_, compact >::iValence(),
graph::VVBIGraph< typename cell::content_t, typename junction::content_t >::nb_-
vertices1(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content,
Edge2Content_, compact >::neighbors(), graph::VVBIGraph< typename
cell::content_t, typename junction::content_t >::no_vertex1(), graph::VVBIGraph<
Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, com-
pact >::operator=(), graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::operator==(), graph::VVBIGraph<
typename cell::content_t, typename junction::content_t >::reference(),
graph::VVBIGraph< typename cell::content_t, typename junction::content_t
>::source(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content,
Edge2Content_, compact >::swap(), graph::VVBIGraph< typename cell::content_t,
typename junction::content_t >::target(), and graph::VVBIGraph< Vertex1Content,
Vertex2Content, Edge1Content, Edge2Content_, compact >::valence().

```

17.72.6.5 `template<typename Vertex1Content, typename Vertex2Content,
typename Edge1Content = _EmptyBiEdgeContent, typename
Edge2Content_ = Edge1Content, bool compact = false>
neighborhood2_t graph::VVBIGraph< Vertex1Content,
Vertex2Content, Edge1Content, Edge2Content_, compact
>::neighborhood2 [protected]`

Main data structure containing the neighborhood of the vertex2's.

Definition at line 1833 of file vvbigraph.h.

```

Referenced by graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::any_vertex2(), graph::VVBIGraph<
typename cell::content_t, typename junction::content_t >::begin_vertex2(),
graph::VVBIGraph< typename cell::content_t, typename junction::content_t
>::clear(), graph::VVBIGraph< typename cell::content_t, typename
junction::content_t >::end_vertex2(), graph::VVBIGraph< Vertex1Content,
Vertex2Content, Edge1Content, Edge2Content_, compact >::eraseAllEdges(),
graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content,
Edge2Content_, compact >::eraseEdge(), graph::VVBIGraph< Vertex1Content,
Vertex2Content, Edge1Content, Edge2Content_, compact >::get_vertex2(),
graph::VVBIGraph< typename cell::content_t, typename junction::content_t

```

```
t >::nb_vertices2(), graph::VVBIGraph< typename cell::content_t, type-
name junction::content_t >::no_vertex2(), graph::VVBIGraph< Ver-
tex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact
>::operator=(), graph::VVBIGraph< Vertex1Content, Vertex2Content,
Edge1Content, Edge2Content_, compact >::operator==((), graph::VVBIGraph<
typename cell::content_t, typename junction::content_t >::reference(),
graph::VVBIGraph< typename cell::content_t, typename junction::content_t
>::source(), graph::VVBIGraph< Vertex1Content, Vertex2Content, Edge1Content,
Edge2Content_, compact >::swap(), and graph::VVBIGraph< typename
cell::content_t, typename junction::content_t >::target().
```

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

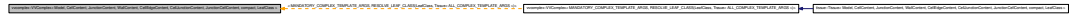
- vvelib/graph/[vbigraph.h](#)

17.73 **vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference**

Class handling the representation and development of 2D cell complex.

```
#include <algorithms/complex.h>
```

Inheritance diagram for vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >:



Public Types

- typedef [graph::Vertex](#)< CellContent > [cell](#)
Type of a cell vertex.
- typedef [cell_graph::arc_t](#) [cell_arc](#)
Type of an arc in the cell graph.
- typedef CellContent [cell_content](#)
- typedef [cell_graph::edge_t](#) [cell_edge](#)
Type of an edge in the cell graph.
- typedef [graph::VVGraph](#)< CellContent, CellEdgeContent, compact > [cell_graph](#)
Type of the cell graph.
- typedef CellJunctionContent [cell_junction_content](#)
- typedef [complex_graph::edge1_t](#) [cell_junction_edge](#)
Type of the edges from cell to junction in the cell complex graph.
- typedef [VVComplexGraph](#)< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact > [complex_graph](#)
Type of the cell complex graph.
- typedef [cell_graph::const_edge_t](#) [const_cell_edge](#)
Type of an edge in the cell graph.
- typedef [complex_graph::const_edge1_t](#) [const_cell_junction_edge](#)
Type of the edges from cell to junction in the cell complex graph.

17.73 `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference` 791

- typedef `complex_graph::const_edge2_t` `const_junction_cell_edge`
Type of the edges from junction to cell in the cell complex graph.

- typedef `wall_graph::const_edge_t` `const_wall`
Type of an edge in the wall graph;.

- typedef `complex_graph::division_data` `division_data`
Type of the division data used for the complex.

- typedef `complex_graph::division_result_t` `division_result_t`
- typedef `graph::Vertex< JunctionContent >` `junction`
Type of a junction vertex.

- typedef `JunctionCellContent` `junction_cell_content`
- typedef `complex_graph::edge2_t` `junction_cell_edge`
Type of the edges from junction to cell in the cell complex graph.

- typedef `JunctionContent` `junction_content`
- typedef `Model` `model_t`
- typedef `wall_graph::edge_t` `wall`
Type of an edge in the wall graph;.

- typedef `wall_graph::arc_t` `wall_arc`
Type of an arc in the wall graph;.

- typedef `WallContent` `wall_content`
- typedef `graph::VVGraph< JunctionContent, WallContent, compact >` `wall_graph`
Type of the wall graph.

Public Member Functions

- template<typename JunctionContainer >
`bool addCell (const cell &c, const JunctionContainer &junctions)`
Add a new cell to the complex.

- `bool addCell (const cell &c, const junction &j1, const junction &j2, const junction &j3, const junction &j4, const junction &j5, const junction &j6, const junction &j7, const junction &j8, const junction &j9)`
Add a new cell with nine vertices to the complex.

- `bool addCell (const cell &c, const junction &j1, const junction &j2, const junction &j3, const junction &j4, const junction &j5, const junction &j6, const junction &j7, const junction &j8)`
Add a new cell with height vertices to the complex.

- `bool addCell (const cell &c, const junction &j1, const junction &j2, const junction &j3, const junction &j4, const junction &j5, const junction &j6, const junction &j7)`
Add a new cell with seven vertices to the complex.
- `bool addCell (const cell &c, const junction &j1, const junction &j2, const junction &j3, const junction &j4, const junction &j5, const junction &j6)`
Add a new cell with six vertices to the complex.
- `bool addCell (const cell &c, const junction &j1, const junction &j2, const junction &j3, const junction &j4, const junction &j5)`
Add a new cell with five vertices to the complex.
- `bool addCell (const cell &c, const junction &j1, const junction &j2, const junction &j3, const junction &j4)`
Add a new cell with four vertices to the complex.
- `bool addCell (const cell &c, const junction &j1, const junction &j2, const junction &j3)`
Add a new cell with three vertices to the complex.
- `virtual bool addCellExtra (const cell &c)`
Method to override to add a cell in extra graphs.
- `template<typename JunctionContainer >`
`bool addCellUnsorted (const cell &c, const JunctionContainer &j)`
Add a new cell to the complex.
- `const cell & adjacentCell (const cell &c, const junction &j) const`
Returns the cell adjacent the c sharing the wall (j, S.nextTo(c,j)).
- `std::pair< cell, cell > adjacentCells (const junction &j1, const junction &j2) const`
Returns the cell(s) sharing the wall (j1,j2).
- `bool border (const junction &j1, const junction &j2) const`
True if the wall from j1 to j2 is at the border of the cell complex.
- `bool border (const cell &c, const junction &j) const`
True if the wall on the cell c starting at junction j is on the border.
- `bool border (const junction &j) const`
True if the junction is on the border.
- `bool border (const cell &c) const`
True if a cell has an border face.

- virtual void `clear ()`
Erase the graphs of the complex.
- virtual void `clearOldGraphs ()`
Called after the division, erased the old graphs maintained during the division itself.
- void `deleteCell (const cell &c)`
Delete a cell from the graph.
- virtual void `deleteCellInGraphs (const cell &c, const std::vector< junction > &deleted)`
*Function to override **and call** to handle cell deletion if other graphs are maintained.*
- bool `deleteJunction (const junction &j)`
Erase a junction.
- virtual void `deleteJunctionExtra (const junction &)`
Called after a junction have been erased.
- template<typename AlgoParameter >
`division_result_t divideCell (const cell &to_divide, const AlgoParameter ¶ms, const cell &cell_kept)`
Divide a cell.
- `division_result_t divideCell (const cell &to_divide, const division_data ¶ms, const cell &cell_kept)`
Divide a cell.
- template<typename AlgoParameter, typename CellContainer >
`division_result_t divideCell (const cell &c, const AlgoParameter ¶ms, const CellContainer &to_keep)`
Divide a cell with the cells to keep defined by any kind of container.
- template<typename AlgoParameter >
`division_result_t divideCell (const cell &c, const AlgoParameter ¶ms, const std::vector< cell > &to_keep)`
Divide a cell of the complex.
- template<typename CellContainer >
`division_result_t divideCell (const cell &c, const division_data &ddata, const CellContainer &to_keep)`
Divide a cell with the cells to keep defined by any kind of container.
- `division_result_t divideCell (const cell &c, const division_data &ddata)`
- `division_result_t divideCell (const cell &c, const division_data &ddata, const std::vector< cell > &to_keep)`

Divide a cell of the complex.

- `const junction & interfaceWall (const cell &c1, const cell &c2) const`
- `std::pair< junction, junction > interfaceWallSpan (const cell &c1, const cell &c2) const`
- `std::pair< junction, junction > mergeCells (const cell &c1, const cell &c2)`

Merge two adjacent cells.

- `void reconnectCellGraph (const cell &c1, const cell &cr, const cell &c)`
- `virtual void reconnectGraphs (const cell &c, const cell &c1, const cell &cr, const junction &u1, const junction &u2, const junction &v1, const junction &v2, const junction &nu, const junction &nv)`

Virtual method to reconnect the graphs after cell division.

- `void reconnectWallGraph (const junction &u1, const junction &u2, const junction &v1, const junction &v2, const junction &nu, const junction nv)`
- `virtual void saveSubgraphs (const std::vector< cell > &cells_to_keep, const std::set< junction > &junctions_to_keep)`

Virtual method to redefine to save the status of the graphs maintained by an extension.

- `bool serialize (storage::VVEStorage &)`

Serialization method.

- `const junction & splitWall (const cell &c, const junction &j1, const junction &x=junction(0))`

Split a wall in two and return the new junction.

- `const junction & splitWall (const junction &j1, const junction &j2, const junction &x=junction(0))`

Split a wall in two and return the new junction.

- `virtual void splitWallExtra (const junction &, const junction &, const junction &)`

Called after a wall was split was done.

- `VVComplex (const VVComplex &other)`

Copy constructor.

- `VVComplex (Model *mod)`

Constructor.

- `virtual ~VVComplex ()`

Virtual destructor allow for inheritance.

- [cell_graph C](#)
Cell graph.
- [Model * model](#)
Model linked to this complex.
- [cell_graph OldC](#)
Saved cell graph.
- [complex_graph OldS](#)
Saved complex graph.
- [wall_graph OldW](#)
Saved wall graph.
- [complex_graph S](#)
Cell complex graph.
- [bool save_parameters](#)
If true, the parameters are also serialized.
- [wall_graph W](#)
Wall graph.

Protected Member Functions

- `void connectFromJunctions (const cell &old_c, const cell &c, const cell &cc)`
Find the connections of cell c in the cell graph, when it's correctly connected in the vv graph.

17.73.1 Detailed Description

```
template<typename Model, typename CellContent, typename JunctionContent,
typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent,
typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false,
typename LeafClass = template_utils::this_class> class vvcomplex::VVComplex< Model, CellContent,
JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >
```

Class handling the representation and development of 2D cell complex.

Definition at line 1944 of file `complex.h`.

17.73.2 Member Typedef Documentation

17.73.2.1 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> typedef graph::Vertex<CellContent> vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cell`

Type of a cell vertex.

Definition at line 1962 of file complex.h.

17.73.2.2 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> typedef cell_graph::arc_t vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cell_arc`

Type of an arc in the cell graph.

Definition at line 1992 of file complex.h.

17.73.2.3 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> typedef cell_graph::edge_t vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cell_edge`

Type of an edge in the cell graph.

Definition at line 1987 of file complex.h.

17.73 `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference` 797

17.73.2.4 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> typedef graph::VVGraph< CellContent, CellEdgeContent, compact> vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cell_graph`

Type of the cell graph.

Definition at line 1972 of file `complex.h`.

17.73.2.5 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> typedef complex_graph::edge1_t vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::cell_junction_edge`

Type of the edges from cell to junction in the cell complex graph.

Definition at line 2017 of file `complex.h`.

17.73.2.6 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> typedef VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact> vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::complex_graph`

Type of the cell complex graph.

Definition at line 1982 of file `complex.h`.

17.73.2.7 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent
= graph::_EmptyEdgeContent, bool compact = false,
typename LeafClass = template_utils::this_class> typedef
cell_graph::const_edge_t vvcomplex::VVComplex< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::const_cell_edge`

Type of an edge in the cell graph.

Definition at line 1997 of file complex.h.

17.73.2.8 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent
= graph::_EmptyEdgeContent, bool compact = false,
typename LeafClass = template_utils::this_class> typedef
complex_graph::const_edge1_t vvcomplex::VVComplex< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::const_cell_junction_edge`

Type of the edges from cell to junction in the cell complex graph.

Definition at line 2022 of file complex.h.

17.73.2.9 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent
= graph::_EmptyEdgeContent, bool compact = false,
typename LeafClass = template_utils::this_class> typedef
complex_graph::const_edge2_t vvcomplex::VVComplex< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::const_junction_cell_edge`

Type of the edges from junction to cell in the cell complex graph.

Definition at line 2032 of file complex.h.

17.73 `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >` **Class Template Reference** 799

17.73.2.10 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> typedef wall_graph::const_edge_t vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::const_wall`

Type of an edge in the wall graph;.

Definition at line 2012 of file complex.h.

17.73.2.11 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> typedef complex_graph::division_data vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::division_data`

Type of the division data used for the complex.

Reimplemented in [tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >](#).

Definition at line 2037 of file complex.h.

17.73.2.12 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> typedef graph::Vertex<JunctionContent> vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::junction`

Type of a junction vertex.

Definition at line 1967 of file complex.h.

17.73.2.13 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent
= graph::_EmptyEdgeContent, bool compact = false,
typename LeafClass = template_utils::this_class> typedef
complex_graph::edge2_t vvcomplex::VVComplex< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::junction_cell_edge`

Type of the edges from junction to cell in the cell complex graph.

Definition at line 2027 of file complex.h.

17.73.2.14 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> typedef wall_graph::edge_t
vvcomplex::VVComplex< Model, CellContent, JunctionContent,
WallContent, CellEdgeContent, CellJunctionContent,
JunctionCellContent, compact, LeafClass >::wall`

Type of an edge in the wall graph;.

Definition at line 2002 of file complex.h.

17.73.2.15 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> typedef wall_graph::arc_t
vvcomplex::VVComplex< Model, CellContent, JunctionContent,
WallContent, CellEdgeContent, CellJunctionContent,
JunctionCellContent, compact, LeafClass >::wall_arc`

Type of an arc in the wall graph;.

Definition at line 2007 of file complex.h.

17.73 `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >` **Class Template Reference** 801

17.73.2.16 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> typedef graph::VVGraph<JunctionContent, WallContent, compact> vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::wall_graph`

Type of the wall graph.

Definition at line 1977 of file complex.h.

17.73.3 Constructor & Destructor Documentation

17.73.3.1 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::VVComplex (Model * mod) [inline]`

Constructor.

Definition at line 2081 of file complex.h.

```
2082         : model(mod)
2083         , save_parameters(false)
2084         { }
```

17.73.3.2 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::VVComplex (const VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > & other) [inline]`

Copy constructor.

Definition at line 2089 of file complex.h.

```

2090         : model (other.model)
2091         , C (other.C)
2092         , W (other.W)
2093         , S (other.S)
2094         , save_parameters (other.save_parameters)
2095         { }
```

17.73.3.3 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> virtual vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::~~VVComplex () [inline, virtual]`

Virtual destructor allow for inheritance.

Definition at line 2100 of file complex.h.

```

2100 {}
```


17.73.4.1 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> template<typename
JunctionContainer > bool vvcomplex::VVComplex< Model,
CellContent, JunctionContent, WallContent, CellEdgeContent,
CellJunctionContent, JunctionCellContent, compact, LeafClass
>::addCell (const cell & c, const JunctionContainer & junctions)
[inline]`

Add a new cell to the complex.

Parameters

c Vertex holding the cell center

j Container holding the counter-clockwise ordered cycle defining the cell.

Note

c must not be in the complex already or result is undefined. However, all or part of the junctions may be in the vv graph already.

Returns

false if the addition fails

Definition at line 2272 of file complex.h.

```

2273     {
2274         if(!S.addCell(c, junctions) ) return false;
2275         return this->addCellExtra(c);
2276     }
```

17.73.4.2 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3, const junction & j4, const junction & j5, const junction & j6, const junction & j7, const junction & j8, const junction & j9) [inline]`

Add a new cell with nine vertices to the complex.

Definition at line 2241 of file complex.h.

```

2244     {
2245         std::vector<junction> junctions(9, junction(0));
2246         junctions[0] = j1;
2247         junctions[1] = j2;
2248         junctions[2] = j3;
2249         junctions[3] = j4;
2250         junctions[4] = j5;
2251         junctions[5] = j6;
2252         junctions[6] = j7;
2253         junctions[7] = j8;
2254         junctions[8] = j9;
2255         return this->addCell(c, junctions);
2256     }

```

17.73.4.3 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3, const junction & j4, const junction & j5, const junction & j6, const junction & j7, const junction & j8) [inline]`

Add a new cell with eight vertices to the complex.

Definition at line 2222 of file complex.h.

```

2225     {
2226         std::vector<junction> junctions(8, junction(0));

```

17.73 vvcomplex::VComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference 805

```

2227         junctions[0] = j1;
2228         junctions[1] = j2;
2229         junctions[2] = j3;
2230         junctions[3] = j4;
2231         junctions[4] = j5;
2232         junctions[5] = j6;
2233         junctions[6] = j7;
2234         junctions[7] = j8;
2235         return this->addCell(c, junctions);
2236     }

```

17.73.4.4 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> bool vvcomplex::VComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3, const junction & j4, const junction & j5, const junction & j6, const junction & j7) [inline]`

Add a new cell with seven vertices to the complex.

Definition at line 2205 of file complex.h.

```

2207     {
2208         std::vector<junction> junctions(7, junction(0));
2209         junctions[0] = j1;
2210         junctions[1] = j2;
2211         junctions[2] = j3;
2212         junctions[3] = j4;
2213         junctions[4] = j5;
2214         junctions[5] = j6;
2215         junctions[6] = j7;
2216         return this->addCell(c, junctions);
2217     }

```

17.73.4.5 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3, const junction & j4, const junction & j5, const junction & j6) [inline]`

Add a new cell with six vertices to the complex.

Definition at line 2189 of file complex.h.

```

2191     {
2192         std::vector<junction> junctions(6, junction(0));
2193         junctions[0] = j1;
2194         junctions[1] = j2;
2195         junctions[2] = j3;
2196         junctions[3] = j4;
2197         junctions[4] = j5;
2198         junctions[5] = j6;
2199         return this->addCell(c, junctions);
2200     }

```

17.73.4.6 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3, const junction & j4, const junction & j5) [inline]`

Add a new cell with five vertices to the complex.

Definition at line 2174 of file complex.h.

```

2176     {
2177         std::vector<junction> junctions(5, junction(0));
2178         junctions[0] = j1;
2179         junctions[1] = j2;
2180         junctions[2] = j3;
2181         junctions[3] = j4;
2182         junctions[4] = j5;
2183         return this->addCell(c, junctions);

```

17.73.4.7 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3, const junction & j4) [inline]`

Add a new cell with four vertices to the complex.

Definition at line 2160 of file complex.h.

```

2162     {
2163         std::vector<junction> junctions(4, junction(0));
2164         junctions[0] = j1;
2165         junctions[1] = j2;
2166         junctions[2] = j3;
2167         junctions[3] = j4;
2168         return this->addCell(c, junctions);
2169     }

```

17.73.4.8 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3) [inline]`

Add a new cell with three vertices to the complex.

Definition at line 2148 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::addCell()`, and `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::addCellUnsorted()`.

```

2149     {
2150         std::vector<junction> junctions(3, junction(0));
2151         junctions[0] = j1;
2152         junctions[1] = j2;
2153         junctions[2] = j3;
2154         return this->addCell(c, junctions);
2155     }

```

17.73.4.9 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> virtual bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::addCellExtra (const cell & c) [inline, virtual]`

Method to override to add a cell in extra graphs.

Note

Do not forget to call this method again if you extend the complex.

Definition at line 2283 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::addCell()`.

```

2284     {
2285         bool result = this->addCell_cell_graph(c) && this->addCell_wall_graph(c);
2286         return result;
2287     }

```

17.73.4.10 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> template<typename JunctionContainer > bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::addCellUnsorted (const cell & c, const JunctionContainer & j) [inline]`

Add a new cell to the complex.

17.73 vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference 809

This function add one cell to the complex. The set of junctions is first sorted in counterclockwise order around the position of the cell *c* and then the [addCell\(\)](#) method is called. For the sorting to work, the cell center need to be correctly positioned with its normal and the junctions just need to be positioned.

Parameters

c First cell of the complex

j Set of junctions of the cell

Definition at line 2422 of file complex.h.

```
2423     {
2424         Point3d normal = model->normal(c);
2425         Point3d cpos, ref, u1, u2;
2426         cpos = model->position(c);
2427         normal.normalize();
2428         bool ref_found = false;
2429         std::set<AngledJunction> sorted;
2430         forall(const junction& v,j)
2431         {
2432             if(ref_found)
2433             {
2434                 const Point3d& pos = model->position(v);
2435                 Point3d u = pos - cpos;
2436                 Point3d proj = normalized(u - (u*normal)*normal);
2437                 double cosa = proj*u1;
2438                 double sina = (u1^proj)*normal;
2439                 double angle = std::atan2(sina, cosa);
2440                 sorted.insert(AngledJunction(v, angle));
2441             }
2442             else
2443             {
2444                 ref = model->position(v);
2445                 u1 = normalized(ref - cpos);
2446                 sorted.insert(AngledJunction(v, 0));
2447                 ref_found = true;
2448             }
2449         }
2450         std::vector<junction> sorted_list;
2451         forall(const AngledJunction& aj,sorted)
2452         {
2453             sorted_list.push_back(aj.v);
2454         }
2455         return this->addCell(c, sorted_list);
2456     }
```

17.73.4.11 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent
= graph::_EmptyEdgeContent, bool compact = false,
typename LeafClass = template_utils::this_class> const cell&
vvcomplex::VVComplex< Model, CellContent, JunctionContent,
WallContent, CellEdgeContent, CellJunctionContent,
JunctionCellContent, compact, LeafClass >::adjacentCell (const cell
& c, const junction & j) const [inline]`

Returns the cell adjacent the c sharing the wall (j, S.nextTo(c,j)).

Definition at line 2743 of file complex.h.

```
2744     {
2745         return S.adjacentCell(c, j);
2746     }
```

17.73.4.12 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> std::pair<cell,cell>
vvcomplex::VVComplex< Model, CellContent, JunctionContent,
WallContent, CellEdgeContent, CellJunctionContent,
JunctionCellContent, compact, LeafClass >::adjacentCells (const
junction & j1, const junction & j2) const [inline]`

Returns the cell(s) sharing the wall (j1,j2).

Returns

a pair of cells (c1,c2) such that in c1, j1 is before j2 and in c2, j2 is before j1.

Definition at line 2754 of file complex.h.

```
2755     {
2756         return S.adjacentCells(j1, j2);
2757     }
```


17.73 vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference 811

17.73.4.13 ~~template<typename Model, typename CellContent,~~
~~typename JunctionContent, typename WallContent =~~
~~graph::_EmptyEdgeContent, typename CellEdgeContent =~~
~~graph::_EmptyEdgeContent, typename CellJunctionContent =~~
~~graph::_EmptyEdgeContent, typename JunctionCellContent~~
~~= graph::_EmptyEdgeContent, bool compact = false,~~
~~typename LeafClass = template_utils::this_class> bool~~
~~vvcomplex::VVComplex< Model, CellContent, JunctionContent,~~
~~WallContent, CellEdgeContent, CellJunctionContent,~~
~~JunctionCellContent, compact, LeafClass >::border (const junction~~
~~&j1, const junction &j2) const [inline]~~

True if the wall from j1 to j2 is at the border of the cell complex.

Definition at line 2876 of file complex.h.

```
2877     {
2878         return S.border(j1, j2);
2879     }
```

17.73.4.14 ~~template<typename Model, typename CellContent,~~
~~typename JunctionContent, typename WallContent =~~
~~graph::_EmptyEdgeContent, typename CellEdgeContent =~~
~~graph::_EmptyEdgeContent, typename CellJunctionContent =~~
~~graph::_EmptyEdgeContent, typename JunctionCellContent~~
~~= graph::_EmptyEdgeContent, bool compact = false,~~
~~typename LeafClass = template_utils::this_class> bool~~
~~vvcomplex::VVComplex< Model, CellContent, JunctionContent,~~
~~WallContent, CellEdgeContent, CellJunctionContent,~~
~~JunctionCellContent, compact, LeafClass >::border (const cell &c,~~
~~const junction &j) const [inline]~~

True if the wall on the cell c starting at junction j is on the border.

Definition at line 2868 of file complex.h.

```
2869     {
2870         return S.border(c, j);
2871     }
```

17.73.4.15 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::border (const junction & j) const [inline]`

True if the junction is on the border.

Definition at line 2860 of file complex.h.

```
2861     {
2862         return S.valence(j) != W.valence(j);
2863     }
```

17.73.4.16 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::border (const cell & c) const [inline]`

True if a cell has an border face.

Definition at line 2846 of file complex.h.

```
2847     {
2848         size_t nb_walls = 0;
2849         forall(const junction& j, S.neighbors(c))
2850         {
2851             if(W.valence(j) > 2)
2852                 nb_walls++;
2853         }
2854         return C.empty(c) or nb_walls != C.valence(c);
2855     }
```

17.73 vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference **813**

17.73.4.17 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> virtual void vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::clear () [inline, virtual]`

Erase the graphs of the complex.

Definition at line 2135 of file complex.h.

```
2136     {
2137         S.clear();
2138         C.clear();
2139         W.clear();
2140         OldS.clear();
2141         OldC.clear();
2142         OldW.clear();
2143     }
```

17.73.4.18 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> virtual void vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::clearOldGraphs () [inline, virtual]`

Called after the division, erased the old graphs maintained during the division itself.

Definition at line 2645 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell()`.

```
2646     {
2647         OldC.clear();
2648         OldW.clear();
2649     }
```

17.73.4.19 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> void vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::connectFromJunctions(const cell & old_c, const cell & c, const cell & cc) [inline, protected]`

Find the connections of cell *c* in the cell graph, when it's correctly connected in the vv graph.

Warning

For internal use only

Definition at line 2899 of file complex.h.

```

2900     {
2901         C.insertEdge(c, cc);
2902         // Connect c to its neighbors
2903         cell pn(0);
2904         forall( const junction& jn , S.neighbors(c))
2905         {
2906             const junction& jp = S.prevTo(c, jn);
2907             const cell& n = S.nextTo(jn, c);
2908             // If we are not at a border ...
2909             if(n != c and S.edge(jp, n))
2910             {
2911                 if(n != cc)
2912                 {
2913                     // Two cases:
2914                     // 1 - old_c is in the neighborhood of n -> we replace it by the
2915                     // current
2916                     // 2 - cc is in the neighborhood of n -> we figure if we should
2917                     // place it before or after
2918                     cell_edge e = C.replace(n, old_c, c);
2919                     if(e.isNull()) // c was not in the neighborhood of n
2920                     {
2921                         const cell& cc1 = S.nextTo(jp, c);
2922                         if(cc1 == cc)
2923                         {
2924                             C.spliceBefore(n, cc, c);
2925                         }
2926                         else
2927                         {
2928                             C.spliceAfter(n, cc, c);
2929                         }
2930                     }
2931                     if(pn.isNull())
2932                     {
2933                         C.insertEdge(c, n);
2934                     }

```

17.73 vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference 815

```

2935         else
2936         {
2937             C.spliceAfter(c, pn, n);
2938         }
2939     }
2940     pn = n;
2941 }
2942 }
2943 }
```

17.73.4.20 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> void vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::deleteCell (const cell & c) [inline]`

Delete a cell from the graph.

Definition at line 2683 of file complex.h.

```

2684     {
2685         const std::vector<junction>& deleted = S.deleteCell(c);
2686         this->deleteCellInGraphs(c, deleted);
2687     }
```

17.73.4.21 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> virtual void vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::deleteCellInGraphs (const cell & c, const std::vector< junction > & deleted) [inline, virtual]`

Function to override **and call** to handle cell deletion if other graphs are maintained.

Definition at line 2693 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteCell()`.

```

2694     {
2695         C.erase(c);
2696         forall(const junction& j,deleted)
2697             W.erase(j);
2698     }

```

17.73.4.22 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::deleteJunction (const junction &j) [inline]`

Erase a junction.

This is possible only if the junction do not change the structure of the cell graph.

Definition at line 2816 of file complex.h.

```

2817     {
2818         size_t nb_cells = S.valence(j);
2819         size_t nb_junctions = W.valence(j);
2820         if(nb_cells > 2)
2821             return false;
2822         if(nb_cells == 2 and nb_junctions != 2)
2823             return false;
2824         if(nb_cells > 0)
2825         {
2826             const cell& c = S.anyIn(j);
2827             const junction& pj = S.prevTo(c, j);
2828             const junction& nj = S.nextTo(c, j);
2829             W.replace(pj, j, nj);
2830             W.replace(nj, j, pj);
2831         }
2832         W.erase(j);
2833         S.erase(j);
2834         this->deleteJunctionExtra(j);
2835         return true;
2836     }

```

17.73 `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >` **Class Template Reference** 817

17.73.4.23 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> virtual void vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::deleteJunctionExtra (const junction &) [inline, virtual]`

Called after a junction have been erased.

Definition at line 2841 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteJunction()`.

2841 {}

17.73.4.24 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> template<typename AlgoParameter > division_result_t vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell (const cell & to_divide, const AlgoParameter & params, const cell & cell_kept) [inline]`

Divide a cell.

Convenience function where a single cell is to be kept

Reimplemented in [tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >](#).

Definition at line 2671 of file complex.h.

```
2674     {
2675         std::vector<cell> k;
2676         k.push_back(cell_kept);
2677         return this->divideCell(to_divide, params, k);
2678     }
```

17.73.4.25 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> division_result_t vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell (const cell & to_divide, const division_data & params, const cell & cell_kept) [inline]`

Divide a cell.

Convenience function where a single cell is to be kept

Reimplemented in [tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >](#).

Definition at line 2656 of file complex.h.

```

2659     {
2660         std::vector<cell> k;
2661         k.push_back(cell_kept);
2662         return this->divideCell(to_divide, params, k);
2663     }

```

17.73.4.26 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> template<typename AlgoParameter, typename CellContainer > division_result_t vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell (const cell & c, const AlgoParameter & params, const CellContainer & to_keep) [inline]`

Divide a cell with the cells to keep defined by any kind of container.

Reimplemented in [tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >](#).

Definition at line 2620 of file complex.h.

```

2623     {

```


17.73 vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference 819

```

2624         std::vector<cell> tk(to_keep.begin(), to_keep.end());
2625         return this->divideCell(c, params, tk);
2626     }

```

17.73.4.27 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename AlgoParameter > division_result_t vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell (const cell & c, const AlgoParameter & params, const std::vector< cell > & to_keep)`
[inline]

Divide a cell of the complex.

Parameters

c Cell to divide

params Parameter of the algorithm to use. The type of the params will decide which algorithm to use.

to_keep Set of cells to keep to update the data in the new cells

See also

[updateFromOld](#)

Definition at line 2556 of file complex.h.

```

2559     {
2560         // Exit if division fails
2561         const division_data& result = findDivisionPoints(c, static_cast<leaf_class>(&(*this), params);
2562         if(!result)
2563         {
2564             OldS.clear();
2565             this->clearOldGraphs();
2566             cerr << "Error, division failed" << endl;
2567             return division_result_t();
2568         }
2569         return divideCell(c, result, to_keep);
2570     }
2571

```

17.73.4.28 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> template<typename CellContainer > division_result_t vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell (const cell & c, const division_data & ddata, const CellContainer & to_keep) [inline]`

Divide a cell with the cells to keep defined by any kind of container.

Reimplemented in [tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >](#).

Definition at line 2537 of file complex.h.

```

2540     {
2541         std::vector<cell> tk(to_keep.begin(), to_keep.end());
2542         return this->divideCell(c, ddata, tk);
2543     }
```

17.73.4.29 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> division_result_t vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell (const cell & c, const division_data & ddata, const std::vector< cell > & to_keep) [inline]`

Divide a cell of the complex.

In the `updateFromOld` call, `cl` will be the cell having the wall (u-v) and `cr` the cell having the wall (v-u). In the same way, `cl` has the vertices of `c` from `v` to `u` and `cr` from `u` to `v`.

Parameters

c Cell to divide

ddata Specification of where the division points should appear

to_keep Set of cells to keep to update the data in the new cells

[updateFromOld](#)

Definition at line 2473 of file complex.h.

Referenced by `tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::divideCell()`, and `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell()`.

```

2476     {
2477         std::set<junction> junctions_to_keep;
2478
2479         forall(const cell& k,to_keep)
2480         {
2481             forall( const junction& j , S.neighbors(k))
2482             {
2483                 junctions_to_keep.insert(j);
2484             }
2485         }
2486
2487         OldS = S.subgraph(to_keep, junctions_to_keep);
2488         this->saveSubgraphs(to_keep, junctions_to_keep);
2489
2490         division_result_t res = S.divideCell(c, ddata);
2491         if(!res)
2492             return res;
2493
2494         // Set the position of u
2495         if(!ddata.divide_at_ul)
2496         {
2497             const Point3d& pu1 = model->position(res.u1);
2498             const Point3d& pu2 = model->position(res.u2);
2499             double ru = ((ddata.pu-pu1)*(pu2-pu1)) / normsq(pu2-pu1);
2500             model->setPositionHint(res.u, res.u1, res.u2, ru);
2501             model->setPosition(res.u, ddata.pu);
2502         }
2503
2504         // Set the position of v
2505         if(!ddata.divide_at_vl)
2506         {
2507             const Point3d& pv1 = model->position(res.v1);
2508             const Point3d& pv2 = model->position(res.v2);
2509             double rv = ((ddata.pv-pv1)*(pv2-pv1)) / normsq(pv2-pv1);
2510             model->setPositionHint(res.v, res.v1, res.v2, rv);
2511             model->setPosition(res.v, ddata.pv);
2512         }
2513
2514         // Set new centers, clear, and update concentrations from parent
2515         //FindCenter(cl, static_cast<leaf_class*>(*this));
2516         //FindCenter(cr, static_cast<leaf_class*>(*this));
2517
2518         // Connect the other graphs
2519         this->reconnectGraphs(c, res.cl, res.cr, res.u1, res.u2, res.v1, res.v2, res
s.u, res.v);
2520
2521         model->updateFromOld(res.cl, res.cr, c, ddata, static_cast<leaf_class*>(*th
is));
2522         OldS.clear();

```

```

2523         this->clearOldGraphs();
2524         return res;
2525     }

```

17.73.4.30 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> const junction& vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::interfaceWall (const cell & c1, const cell & c2) const [inline]`

Returns

the junction in c1 starting the wall common between c1 and c2.

Definition at line 2763 of file complex.h.

```

2764     {
2765         return S.interfaceWall(c1, c2);
2766     }

```

17.73.4.31 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> std::pair<junction, junction> vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::interfaceWallSpan (const cell & c1, const cell & c2) const [inline]`

Returns

the pair of junctions in c1 such that the interface between c1 and c2 starts at the first junction and end at the second. In case the c1 is entirely contained in c2 (or the other way around), both junctions will be the same.

Definition at line 2774 of file complex.h.

```

2775     {
2776         return S.interfaceWallSpan(c1, c2);
2777     }

```

17.73 `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >` **Class Template Reference** **823**

17.73.4.32 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> std::pair<junction, junction> vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::mergeCells (const cell & c1, const cell & c2) [inline]`

Merge two adjacent cells.

Only c1 is kept, c2 is being deleted.

Returns

the junctions at the start and end of the wall that has been removed

Definition at line 2708 of file complex.h.

```

2709     {
2710         std::vector<junction> deleted_junctions;
2711         std::pair<junction, junction> result = S.mergeCells(c1, c2, &deleted_junctio
ns);
2712
2713         const junction& j1 = result.first;
2714         const junction& j2 = result.second;
2715
2716         // Remove junctions from W
2717         forall(const junction& j, deleted_junctions)
2718         {
2719             W.erase(j);
2720         }
2721         W.spliceAfter(j1, S.prevTo(c1, j1), j2);
2722         W.spliceBefore(j2, S.nextTo(c1, j2), j1);
2723
2724         // Update cell graph
2725         const cell& after = C.nextTo(c1, c2);
2726         forall(const cell& c, C.neighbors(c2, C.nextTo(c2, c1)))
2727         {
2728             if(c == c1) break;
2729             C.spliceBefore(c1, after, c);
2730             C.replace(c, c2, c1);
2731         }
2732
2733         // Erase c2 from C and S
2734         C.erase(c2);
2735
2736         return result;
2737     }

```

17.73.4.33 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> virtual void vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::reconnectGraphs (const cell & c, const cell & cl, const cell & cr, const junction & u1, const junction & u2, const junction & v1, const junction & v2, const junction & nu, const junction & nv) [inline, virtual]`

Virtual method to reconnect the graphs after cell division.

This method input all the parameters for the cell division. The division itself already occurred only for the complex graph at that point. This function maintain the consistency with other graphs.

Note

Don't forget to call this method from your overloaded version.

Definition at line 2607 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell()`.

```

2611     {
2612         reconnectWallGraph(u1, u2, v1, v2, nu, nv);
2613         reconnectCellGraph(cl, cr, c);
2614     }

```

17.73.4.34 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> virtual void vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::saveSubgraphs (const std::vector< cell > & cells_to_keep, const std::set< junction > & junctions_to_keep) [inline, virtual]`

Virtual method to redefine to save the status of the graphs maintained by an extension.

17.73 `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference` 825

Note

Don't forget to call this method.

Definition at line 2634 of file `complex.h`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell()`.

```
2636     {
2637         OldC = C.subgraph(cells_to_keep);
2638         OldW = W.subgraph(junctions_to_keep);
2639     }
```

17.73.4.35 `template<typename Model , typename CellContent , typename JunctionContent , typename WallContent , typename CellEdgeContent , typename CellJunctionContent , typename JunctionCellContent , bool compact, typename LeafClass > bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::serialize (storage::VVEStorage & store) [inline]`

Serialization method.

By default, only the three main graphs are serialized: S, T and W. No parameter and no old graph are saved.

Warning

You need to include `<storage/complex.h>` to use this serialization method

Definition at line 31 of file `complex.h`.

References `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::C`, `storage::VVEStorage::field()`, `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::S`, `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::save_parameters`, and `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::W`.

```
32     {
33         if(!store.field("S", S))
34         {
35             return false;
36         }
37         if(!store.field("C", C))
```

```

38     {
39         return false;
40     }
41     if(!store.field("W", W))
42     {
43         return false;
44     }
45     if(save_parameters)
46     {
47     }
48     return true;
49 }

```

17.73.4.36 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> const junction& vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::splitWall (const cell & c, const junction & j1, const junction & x = junction (0)) [inline]`

Split a wall in two and return the new junction.

Definition at line 2793 of file complex.h.

```

2794     {
2795         const junction& j2 = S.nextTo(c, j1);
2796         const junction& j = algorithms::insert(j1, j2, W, x);
2797         S.splitWall(c, j1, j);
2798         this->splitWallExtra(j1, j2, j);
2799         return j;
2800     }

```

17.73.4.37 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> const junction& vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::splitWall (const junction & j1, const junction & j2, const junction & x = junction (0)) [inline]`

Split a wall in two and return the new junction.

17.73 vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference 827

Definition at line 2782 of file complex.h.

```
2783     {
2784         const junction& j = algorithms::insert(j1, j2, W, x);
2785         S.splitWall(j1, j2, j);
2786         this->splitWallExtra(j1, j2, j);
2787         return j;
2788     }
```

17.73.4.38 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> virtual void vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::splitWallExtra (const junction &, const junction &, const junction &) [inline, virtual]`

Called after a wall was split was done.

The argument is the junction inserted. Getting the original wall is easily done using `W.anyIn(j)` and `W.nextTo(W.anyIn(j))`.

Definition at line 2808 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::splitWall()`.

```
2808 { }
```

17.73.5 Member Data Documentation

17.73.5.1 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> cell_graph vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::C`

Cell graph.

Definition at line 2047 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::border()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::clear()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::connectFromJunctions()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteCellInGraphs()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::mergeCells()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::saveSubgraphs()`, and `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::serialize()`.

17.73.5.2 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> Model* vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::model`

[Model](#) linked to this complex.

Definition at line 2042 of file `complex.h`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::addCellUnsorted()`, and `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell()`.

17.73 `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >` **Class Template Reference** 829

17.73.5.3 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> cell_graph vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::OldC`

Saved cell graph.

Only during cell division!

Definition at line 2051 of file `complex.h`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::clear()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::clearOldGraphs()`, and `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::saveSubgraphs()`.

17.73.5.4 `template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> complex_graph vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::OldS`

Saved complex graph.

Only during cell division!

Definition at line 2069 of file `complex.h`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::clear()`, and `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell()`.

17.73.5.5 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent
= graph::_EmptyEdgeContent, bool compact = false,
typename LeafClass = template_utils::this_class> wall_graph
vvcomplex::VVComplex< Model, CellContent, JunctionContent,
WallContent, CellEdgeContent, CellJunctionContent,
JunctionCellContent, compact, LeafClass >::OldW`

Saved wall graph.

Only during cell division!

Definition at line 2060 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_-
TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_-
COMPLEX_TEMPLATE_ARGS >)>::clear(), vvcomplex::VVComplex<
MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_-
CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS
>)>::clearOldGraphs(), and vvcomplex::VVComplex< MANDATORY_-
COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue<
ALL_COMPLEX_TEMPLATE_ARGS >)>::saveSubgraphs().`

17.73.5.6 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent =
graph::_EmptyEdgeContent, bool compact = false, typename
LeafClass = template_utils::this_class> complex_graph
vvcomplex::VVComplex< Model, CellContent, JunctionContent,
WallContent, CellEdgeContent, CellJunctionContent,
JunctionCellContent, compact, LeafClass >::S`

Cell complex graph.

Definition at line 2065 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_-
TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_-
COMPLEX_TEMPLATE_ARGS >)>::addCell(), vvcomplex::VVComplex<
MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_-
CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS
>)>::adjacentCell(), vvcomplex::VVComplex< MANDATORY_COMPLEX_-
TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_-
COMPLEX_TEMPLATE_ARGS >)>::adjacentCells(), vvcomplex::VVComplex<
MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_-
CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::border(),`

17.73 vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass > Class Template Reference 831

```

vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::clear(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::connectFromJunctions(),
vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteCell(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteJunction(),
vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell(), vvcomplex::findCenter(), vvcomplex::FindOppositeWall(),
vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::interfaceWall(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::interfaceWallSpan(),
vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::mergeCells(), vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::serialize(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::splitWall(), and vvcomplex::testDivisionOnVertices().

```

17.73.5.7 template<typename Model, typename CellContent, typename JunctionContent, typename WallContent = graph::_EmptyEdgeContent, typename CellEdgeContent = graph::_EmptyEdgeContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false, typename LeafClass = template_utils::this_class> bool vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::save_parameters

If true, the parameters are also serialized.

Not that this parameter will never be serialized ...

Definition at line 2076 of file complex.h.

Referenced by vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::serialize().

17.73.5.8 `template<typename Model, typename CellContent,
typename JunctionContent, typename WallContent =
graph::_EmptyEdgeContent, typename CellEdgeContent =
graph::_EmptyEdgeContent, typename CellJunctionContent =
graph::_EmptyEdgeContent, typename JunctionCellContent
= graph::_EmptyEdgeContent, bool compact = false,
typename LeafClass = template_utils::this_class> wall_graph
vvcomplex::VVComplex< Model, CellContent, JunctionContent,
WallContent, CellEdgeContent, CellJunctionContent,
JunctionCellContent, compact, LeafClass >::W`

Wall graph.

Definition at line 2056 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::border()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::clear()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteCellInGraphs()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteJunction()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::mergeCells()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::saveSubgraphs()`, `vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::serialize()`, and `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::splitWall()`.

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

- [vvelib/algorithms/complex.h](#)
- [vvelib/storage/complex.h](#)

17.74 `vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >` Class Template

Reference

17.74 `vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >` Class Template
Reference

Definition of the bipartite graph with specialized methods for the complex graph.

```
#include <algorithms/complex.h>
```

Inheritance diagram for `vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >`:



Classes

- class `division_result_t`
Describe the result of a cell division.

Public Types

- typedef `ParentClass::vertex1_t` `cell`
- typedef `ParentClass::edge1_t` `cell_junction_edge`
- typedef `ParentClass::const_edge1_t` `const_cell_junction_edge`
- typedef `ParentClass::const_iterator1` `const_iterator1`
Constant iterator on the vertices of type 1.
- typedef `ParentClass::const_iterator2` `const_iterator2`
Constant iterator on the vertices of type 1.
- typedef `ParentClass::const_edge2_t` `const_junction_cell_edge`
- typedef `ParentClass::const_range_vertex1` `const_range_cell`
- typedef `ParentClass::const_range_vertex2` `const_range_junction`
- typedef `DivisionData< JunctionContent >` `division_data`
Structure describing a cell division.
- typedef `ParentClass::iterator1` `iterator1`
Iterator on the vertices of type 1.
- typedef `ParentClass::iterator2` `iterator2`
Iterator on the vertices of type 2.
- typedef `ParentClass::vertex2_t` `junction`

- typedef [ParentClass::edge2_t](#) **junction_cell_edge**
- typedef [graph::VVBiGraph](#)< COMPLEX_ARGS > **ParentClass**
- typedef [ParentClass::range_vertex1](#) **range_cell**
- typedef [ParentClass::range_vertex2](#) **range_junction**
- typedef [ParentClass::size_type](#) **size_type**

Type of a size.

Public Member Functions

- template<typename JunctionContainer >
bool **addCell** (const [cell](#) &c, const JunctionContainer &junctions)
- bool **addCell** (const [cell](#) &c, const [junction](#) &j1, const [junction](#) &j2, const [junction](#) &j3, const [junction](#) &j4, const [junction](#) &j5, const [junction](#) &j6, const [junction](#) &j7, const [junction](#) &j8, const [junction](#) &j9)
Add a new cell with nine vertices to the complex.
- bool **addCell** (const [cell](#) &c, const [junction](#) &j1, const [junction](#) &j2, const [junction](#) &j3, const [junction](#) &j4, const [junction](#) &j5, const [junction](#) &j6, const [junction](#) &j7, const [junction](#) &j8)
Add a new cell with height vertices to the complex.
- bool **addCell** (const [cell](#) &c, const [junction](#) &j1, const [junction](#) &j2, const [junction](#) &j3, const [junction](#) &j4, const [junction](#) &j5, const [junction](#) &j6, const [junction](#) &j7)
Add a new cell with seven vertices to the complex.
- bool **addCell** (const [cell](#) &c, const [junction](#) &j1, const [junction](#) &j2, const [junction](#) &j3, const [junction](#) &j4, const [junction](#) &j5, const [junction](#) &j6)
Add a new cell with six vertices to the complex.
- bool **addCell** (const [cell](#) &c, const [junction](#) &j1, const [junction](#) &j2, const [junction](#) &j3, const [junction](#) &j4, const [junction](#) &j5)
Add a new cell with five vertices to the complex.
- bool **addCell** (const [cell](#) &c, const [junction](#) &j1, const [junction](#) &j2, const [junction](#) &j3, const [junction](#) &j4)
Add a new cell with four vertices to the complex.
- bool **addCell** (const [cell](#) &c, const [junction](#) &j1, const [junction](#) &j2, const [junction](#) &j3)
Add a new cell with three vertices to the complex.
- const [cell](#) & **adjacentCell** (const [cell](#) &c, const [junction](#) &j) const
Returns the cell adjacent the c sharing the wall (j, S.nextTo(c,j)).

17.74 vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact > Class Template

Reference

835

- `std::pair< cell, cell > adjacentCells (const junction &j1, const junction &j2)`
`const`
Returns the cell(s) sharing the wall (j1,j2).
- `const cell & any_cell () const`
Returns a cell from the complex graph.
- `const junction & any_junction () const`
Retrns a junction from the complex graph.
- `bool border (const junction &j1, const junction &j2) const`
- `bool border (const cell &c) const`
Test if a junction is at the border of the complex graph.
- `bool border (const junction &j) const`
Test if a junction is at the border of the complex graph.
- `bool border (const cell &c, const junction &j) const`
Test if the wall of the cell c starting at j is at the border of the complex graph.
- `std::vector< junction > contour () const`
- `std::vector< junction > deleteCell (const cell &c)`
Delete a cell and all the junctions that become orphans.
- `division_result_t divideCell (const cell &c, const division_data &ddata)`
Divide a cell.
- `const cell & get_cell (size_type idx) const`
Returns the cell number idx.
- `const junction & get_junction (size_type idx) const`
Returns the junction number idx.
- `const junction & interfaceWall (const cell &c1, const cell &c2) const`
- `std::pair< junction, junction > interfaceWallSpan (const cell &c1, const cell &c2) const`
- `std::pair< junction, junction > mergeCells (const cell &c1, const cell &c2, std::vector< junction > *deleted_junctions=0)`
Merge two adjacent cells.
- `size_t nb_cells () const`
Returns the number of cells in the graph.
- `size_t nb_junctions () const`
Returns the number of junctions in the graph.

- `std::vector< junction > nextContourVertex` (const `junction` &first, const `cell` &c, const `junction` &j, `std::vector< junction > acc`) const
- `bool no_cell ()` const
True if there are no cells.
- `bool no_junction ()` const
True if there are no junctions.
- `const junction & splitWall` (const `cell` &c, const `junction` &j1, const `junction` &x=`junction`(0))
Split a wall in two and return the new junction.
- `const junction & splitWall` (const `junction` &j1, const `junction` &j2, const `junction` &x=`junction`(0))
Split a wall in two and return the new junction.
- `VVComplexGraph` (const `ParentClass` &other)

STL iterators

- `const_iterator1 begin_cells ()` const
- `iterator1 begin_cells ()`
- `const_iterator2 begin_junctions ()` const
- `iterator2 begin_junctions ()`
- `const_range_cell cells ()` const
- `range_cell cells ()`
- `const_iterator1 end_cells ()` const
- `iterator1 end_cells ()`
- `const_iterator2 end_junctions ()` const
- `iterator2 end_junctions ()`
- `const_range_junction junctions ()` const
- `range_junction junctions ()`

17.74.1 Detailed Description

```
template<typename CellContent, typename JunctionContent, typename
CellJunctionContent = graph::_EmptyEdgeContent, typename Junction-
CellContent = graph::_EmptyEdgeContent, bool compact = false> class
vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunction-
Content, JunctionCellContent, compact >
```

Definition of the bipartite graph with specialized methods for the complex graph.

Definition at line 860 of file `complex.h`.

17.74.2 Member Typedef Documentation

17.74.2.1 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> typedef ParentClass::const_iterator1 vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::const_iterator1`

Constant iterator on the vertices of type 1.

Reimplemented from [graph::VVBIGraph< COMPLEX_ARGS >](#).

Definition at line 865 of file complex.h.

17.74.2.2 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> typedef ParentClass::const_iterator2 vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::const_iterator2`

Constant iterator on the vertices of type 1.

Reimplemented from [graph::VVBIGraph< COMPLEX_ARGS >](#).

Definition at line 866 of file complex.h.

17.74.2.3 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> typedef DivisionData<JunctionContent> vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::division_data`

Structure describing a cell division.

Definition at line 1349 of file complex.h.

17.74.2.4 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> typedef ParentClass::iterator1 vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::iterator1`

Iterator on the vertices of type 1.

Reimplemented from [graph::VVBIGraph< COMPLEX_ARGS >](#).

Definition at line 863 of file complex.h.

```
17.74.2.5  template<typename CellContent, typename JunctionContent,
            typename CellJunctionContent = graph::_EmptyEdgeContent,
            typename JunctionCellContent = graph::_EmptyEdgeContent,
            bool compact = false> typedef ParentClass::iterator2
            vvcomplex::VVComplexGraph< CellContent, JunctionContent,
            CellJunctionContent, JunctionCellContent, compact >::iterator2
```

Iterator on the vertices of type 2.

Reimplemented from [graph::VVBIGraph< COMPLEX_ARGS >](#).

Definition at line 864 of file complex.h.

```
17.74.2.6  template<typename CellContent, typename JunctionContent,
            typename CellJunctionContent = graph::_EmptyEdgeContent,
            typename JunctionCellContent = graph::_EmptyEdgeContent,
            bool compact = false> typedef ParentClass::size_type
            vvcomplex::VVComplexGraph< CellContent, JunctionContent,
            CellJunctionContent, JunctionCellContent, compact >::size_type
```

Type of a size.

Reimplemented from [graph::VVBIGraph< COMPLEX_ARGS >](#).

Definition at line 873 of file complex.h.

17.74.3 Member Function Documentation

```
17.74.3.1  template<typename CellContent, typename JunctionContent,
            typename CellJunctionContent = graph::_EmptyEdgeContent,
            typename JunctionCellContent = graph::_EmptyEdgeContent, bool
            compact = false> bool vvcomplex::VVComplexGraph< CellContent,
            JunctionContent, CellJunctionContent, JunctionCellContent,
            compact >::addCell (const cell & c, const junction & j1, const
            junction & j2, const junction & j3, const junction & j4, const
            junction & j5, const junction & j6, const junction & j7, const
            junction & j8, const junction & j9)  [inline]
```

Add a new cell with nine vertices to the complex.

Definition at line 1040 of file complex.h.

```
1043      {
1044          std::vector<junction> junctions(9, junction(0));
1045          junctions[0] = j1;
1046          junctions[1] = j2;
1047          junctions[2] = j3;
1048          junctions[3] = j4;
1049          junctions[4] = j5;
```

17.74 vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact > Class Template

839

Reference

```
1050     junctions[5] = j6;
1051     junctions[6] = j7;
1052     junctions[7] = j8;
1053     junctions[8] = j9;
1054     return this->addCell(c, junctions);
1055 }
```

17.74.3.2 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> bool vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3, const junction & j4, const junction & j5, const junction & j6, const junction & j7, const junction & j8) [inline]`

Add a new cell with height vertices to the complex.

Definition at line 1021 of file complex.h.

```
1024     {
1025         std::vector<junction> junctions(8, junction(0));
1026         junctions[0] = j1;
1027         junctions[1] = j2;
1028         junctions[2] = j3;
1029         junctions[3] = j4;
1030         junctions[4] = j5;
1031         junctions[5] = j6;
1032         junctions[6] = j7;
1033         junctions[7] = j8;
1034         return this->addCell(c, junctions);
1035     }
```

17.74.3.3 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> bool vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3, const junction & j4, const junction & j5, const junction & j6, const junction & j7) [inline]`

Add a new cell with seven vertices to the complex.

Definition at line 1004 of file complex.h.

```
1006     {
1007         std::vector<junction> junctions(7, junction(0));
1008         junctions[0] = j1;
1009         junctions[1] = j2;
1010         junctions[2] = j3;
```

```

1011     junctions[3] = j4;
1012     junctions[4] = j5;
1013     junctions[5] = j6;
1014     junctions[6] = j7;
1015     return this->addCell(c, junctions);
1016 }

```

17.74.3.4 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> bool vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3, const junction & j4, const junction & j5, const junction & j6) [inline]`

Add a new cell with six vertices to the complex.

Definition at line 988 of file complex.h.

```

990     {
991         std::vector<junction> junctions(6, junction(0));
992         junctions[0] = j1;
993         junctions[1] = j2;
994         junctions[2] = j3;
995         junctions[3] = j4;
996         junctions[4] = j5;
997         junctions[5] = j6;
998         return this->addCell(c, junctions);
999     }

```

17.74.3.5 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> bool vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3, const junction & j4, const junction & j5) [inline]`

Add a new cell with five vertices to the complex.

Definition at line 973 of file complex.h.

```

975     {
976         std::vector<junction> junctions(5, junction(0));
977         junctions[0] = j1;
978         junctions[1] = j2;
979         junctions[2] = j3;
980         junctions[3] = j4;
981         junctions[4] = j5;
982         return this->addCell(c, junctions);
983     }

```

17.74 vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact > Class Template

Reference

841

17.74.3.6 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> bool vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3, const junction & j4) [inline]`

Add a new cell with four vertices to the complex.

Definition at line 959 of file complex.h.

```
961     {
962         std::vector<junction> junctions(4, junction(0));
963         junctions[0] = j1;
964         junctions[1] = j2;
965         junctions[2] = j3;
966         junctions[3] = j4;
967         return this->addCell(c, junctions);
968     }
```

17.74.3.7 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> bool vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::addCell (const cell & c, const junction & j1, const junction & j2, const junction & j3) [inline]`

Add a new cell with three vertices to the complex.

Definition at line 947 of file complex.h.

Referenced by vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::addCell(), and vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::addCell().

```
948     {
949         std::vector<junction> junctions(3, junction(0));
950         junctions[0] = j1;
951         junctions[1] = j2;
952         junctions[2] = j3;
953         return this->addCell(c, junctions);
954     }
```

17.74.3.8 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> const cell& vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::adjacentCell (const cell & c, const junction & j) const [inline]`

Returns the cell adjacent the c sharing the wall (j, S.nextTo(c,j)).

Definition at line 1177 of file complex.h.

Referenced by vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::adjacentCell(), vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::divideCell(), and vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::splitWall().

```

1178     {
1179         const junction& nj = this->nextTo(c, j);
1180         const cell& adj = this->prevTo(j, c);
1181         if(this->prevTo(adj, j) == nj)
1182             return adj;
1183         return cell::null;
1184     }

```

17.74.3.9 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> std::pair<cell,cell> vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::adjacentCells (const junction & j1, const junction & j2) const [inline]`

Returns the cell(s) sharing the wall (j1,j2).

Returns

a pair of cells (c1,c2) such that in c1, j1 is before j2 and in c2, j2 is before j1.

Definition at line 1192 of file complex.h.

Referenced by vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::adjacentCells().

17.74 vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact > Class Template

843

Reference

```
1193 {
1194     vvassert(j1 != j2);
1195     forall(const cell& c , neighbors(j1))
1196     {
1197         if(nextTo(c, j1) == j2)
1198         {
1199             const cell& c2 = prevTo(j1, c);
1200             if(prevTo(c2, j1) == j2)
1201                 return std::make_pair(c,c2);
1202             return std::make_pair(c, cell::null);
1203         }
1204         else if(prevTo(c, j1) == j2)
1205         {
1206             const cell& c1 = nextTo(j1, c);
1207             if(nextTo(c1, j1) == j2)
1208                 return std::make_pair(c1, c);
1209             return std::make_pair(cell::null, c);
1210         }
1211     }
1212     return std::make_pair(cell::null, cell::null);
1213 }
```

17.74.3.10 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> const cell& vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::any_cell () const [inline]`

Returns a cell from the complex graph.

Definition at line 890 of file complex.h.

```
890 { return this->any_vertex1(); }
```

17.74.3.11 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> const junction& vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::any_junction () const [inline]`

Retrns a junction from the complex graph.

Definition at line 894 of file complex.h.

```
894 { return this->any_vertex2(); }
```

17.74.3.12 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> bool vvcomplex::VVComplexGraph<CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::border (const cell & c) const [inline]`

Test if a junction is at the border of the complex graph.

Definition at line 1153 of file complex.h.

```

1154     {
1155         forall( const junction& j , neighbors(c))
1156         {
1157             if(border(c, j)) return true;
1158         }
1159         return false;
1160     }

```

17.74.3.13 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> bool vvcomplex::VVComplexGraph<CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::border (const junction & j) const [inline]`

Test if a junction is at the border of the complex graph.

Definition at line 1141 of file complex.h.

```

1142     {
1143         forall( const cell& c , this->neighbors(j))
1144         {
1145             if(border(c, j)) return true;
1146         }
1147         return false;
1148     }

```

17.74.3.14 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> bool vvcomplex::VVComplexGraph<CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::border (const cell & c, const junction & j) const [inline]`

Test if the wall of the cell c starting at j is at the border of the complex graph.

Definition at line 1132 of file complex.h.

17.74 vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact > Class Template

Reference

845

Referenced by vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::border(), and vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::border().

```
1133     {
1134         const cell& c2 = this->prevTo(j, c);
1135         return this->nextTo(c, j) != this->prevTo(c2, j);
1136     }
```

17.74.3.15 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> std::vector<junction> vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::deleteCell(const cell &c) [inline]`

Delete a cell and all the junctions that become orphans.

Returns

the list of junctions deleted

Definition at line 1293 of file complex.h.

Referenced by vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteCell().

```
1294     {
1295         typename ParentClass::neighbor_iterator1_range p = neighbors(c);
1296         std::vector<junction> js(p.begin(), p.end());
1297         std::vector<junction> deleted;
1298         erase(c);
1299         forall(const junction& j, js) if(empty(j))
1300         {
1301             erase(j);
1302             deleted.push_back(j);
1303         }
1304         return deleted;
1305     }
```

17.74.3.16 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> division_result_t vvcomplex::VVComplexGraph<CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::divideCell (const cell & c, const division_data & ddata) [inline]`

Divide a cell.

Definition at line 1405 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::divideCell()`.

```

1407     {
1408         //Point3d pu = ddata.pu, pv = ddata.pv;
1409         junction u1 = ddata.u1, v1 = ddata.v1;
1410         junction u2 = this->nextTo(c, u1), v2 = this->nextTo(c, v1);
1411
1412         junction u(0), v(0);
1413
1414         // First, find the neighbor cells
1415         const cell& nu = adjacentCell(c, u1);
1416         const cell& nv = adjacentCell(c, v1);
1417
1418         // Insert a new vertex on the new wall
1419         if(ddata.divide_at_u1)
1420             u = u1;
1421         else
1422         {
1423             u = junction();
1424             this->insert(u);
1425             this->spliceAfter(c, u1, u);
1426             this->insertEdge(u, c);
1427         }
1428
1429         // Insert a new vertex on the other side
1430         if(ddata.divide_at_v1)
1431             v = v1;
1432         else
1433         {
1434             v = junction();
1435             this->insert(v);
1436             this->spliceAfter(c, v1, v);
1437             this->insertEdge(v, c);
1438         }
1439
1440         // If necessary, adjust the connectivity of the adjacent cells
1441         // Then compute the position of the new vertices and adjust the one of
1442         // the cells around
1443         if(!ddata.divide_at_u1)
1444         {
1445             if(nu)
1446             {
1447                 this->spliceBefore(nu, u1, u);
1448                 this->insertEdge(u, nu);
1449             }
1450         }

```

17.74 vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact > Class Template

Reference

847

```
1451
1452     if(!ddata.divide_at_v1)
1453     {
1454         if(nv)
1455         {
1456             this->spliceBefore(nv, v1, v);
1457             this->insertEdge(v, nv);
1458         }
1459     }
1460
1461     // Create the two new cell centers
1462     cell cl, cr;
1463     this->insert(cl);
1464     this->insert(cr);
1465
1466     // Connect new junctions
1467     this->insertEdge(cl, u);
1468     this->insertEdge(cl, v);
1469     this->insertEdge(cr, u);
1470     this->insertEdge(cr, v);
1471
1472     this->spliceAfter(u, c, cl);
1473     this->replace(u, c, cr);
1474     this->spliceBefore(v, c, cl);
1475     this->replace(v, c, cr);
1476
1477     // Connect the new centers
1478     forall( const junction& x , this->neighbors(c, this->nextTo(c, u)))
1479     {
1480         if(x == v) break;
1481         this->spliceBefore(cr, v, x);
1482         this->replace(x, c, cr);
1483     }
1484
1485     forall( const junction& y , this->neighbors(c, this->nextTo(c, v)))
1486     {
1487         if(y == u) break;
1488         this->spliceBefore(cl, u, y);
1489         this->replace(y, c, cl);
1490     }
1491
1492     // Remove c
1493     this->erase(c);
1494
1495     return division_result_t(cl, cr, u1, u2, u, v1, v2, v);
1496 }
```

17.74.3.17 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> const cell& vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::get_cell (size_type idx) const [inline]`

Returns the cell number idx.

Definition at line 938 of file complex.h.

```
938 { return this->get_vertex1(idx); }
```

17.74.3.18 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> const junction& vvcomplex::VVComplexGraph<CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::get_junction (size_type idx) const [inline]`

Returns the junction number *idx*.

Definition at line 942 of file complex.h.

```
942 { return this->get_vertex2(idx); }
```

17.74.3.19 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> const junction& vvcomplex::VVComplexGraph<CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::interfaceWall (const cell & c1, const cell & c2) const [inline]`

Returns

the junction in *c1* starting the wall common between *c1* and *c2*.

Definition at line 1251 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::interfaceWall()`.

```
1252 {
1253     forall( const junction& j, neighbors(c1))
1254     {
1255         if(prevTo(j, c1) == c2)
1256         {
1257             if(nextTo(c1, j) == prevTo(c2, j))
1258             {
1259                 return startInterfaceWall(j, c1, c2);
1260             }
1261         }
1262     }
1263     return junction::null;
1264 }
```

17.74 vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact > Class Template

Reference

849

17.74.3.20 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> std::pair<junction, junction> vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::interfaceWallSpan (const cell & c1, const cell & c2) const [inline]`

Returns

the pair of junctions in c1 such that the interface between c1 and c2 starts at the first junction and end at the second. In case the c1 is entirely contained in c2 (or the other way around), both junctions will be the same.

Definition at line 1272 of file complex.h.

Referenced by vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::interfaceWallSpan(), and vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::mergeCells().

```
1273     {
1274         forall(const junction& j, neighbors(c1))
1275         {
1276             if(prevTo(j, c1) == c2)
1277             {
1278                 if(nextTo(c1, j) == prevTo(c2, j))
1279                 {
1280                     return std::make_pair(startInterfaceWall(j, c1, c2),
1281                                           endInterfaceWall(j, c1, c2));
1282                 }
1283             }
1284         }
1285         return std::make_pair(junction::null, junction::null);
1286     }
```

17.74.3.21 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> std::pair<junction, junction> vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::mergeCells (const cell & c1, const cell & c2, std::vector<junction > * deleted_junctions = 0) [inline]`

Merge two adjacent cells.

Only c1 is kept, c2 is being deleted.

Returns

the junctions at the start and end of the wall that has been removed

Definition at line 1506 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::mergeCells()`.

```

1507     {
1508         std::pair<junction, junction> result = interfaceWallSpan(c1, c2);
1509         if(!result.first)
1510             return result;
1511         junction j1 = result.first;
1512         junction j2 = result.second;
1513         std::vector<junction> local_deleted_junctions;
1514         if(deleted_junctions == 0) deleted_junctions = &local_deleted_junctions;
1515         forall(const junction& j, neighbors(c1, nextTo(c1, j1)))
1516         {
1517             if(j == j2)
1518                 break;
1519             deleted_junctions->push_back(j);
1520         }
1521
1522         // Remove junctions from c1 and W
1523         forall(const junction& j, *deleted_junctions)
1524         {
1525             eraseEdge(c1, j);
1526         }
1527
1528         // Update S
1529         forall(const junction& j, neighbors(c2, nextTo(c2, j1)))
1530         {
1531             if(j == j2) break;
1532             spliceBefore(c1, j2, j);
1533             replace(j, c2, c1);
1534         }
1535
1536         // Erase c2 from C and S
1537         erase(c2);
1538
1539         return result;
1540     }

```

17.74.3.22 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> size_t vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::nb_cells() const [inline]`

Returns the number of cells in the graph.

Definition at line 920 of file complex.h.

```

920 { return this->nb_vertices1(); }

```


17.74 vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact > Class Template

Reference

851

17.74.3.23 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> size_t vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::nb_junctions () const [inline]`

Returns the number of junctions in the graph.

Definition at line 924 of file complex.h.

```
924 { return this->nb_vertices2(); }
```

17.74.3.24 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> bool vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::no_cell () const [inline]`

True if there are no cells.

Definition at line 929 of file complex.h.

```
929 { return this->no_vertex1(); }
```

17.74.3.25 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> bool vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::no_junction () const [inline]`

True if there are no junctions.

Definition at line 933 of file complex.h.

```
933 { return this->no_vertex2(); }
```

17.74.3.26 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> const junction& vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::splitWall (const cell & c, const junction & jI, const junction & x = junction (0)) [inline]`

Split a wall in two and return the new junction.

Definition at line 1332 of file complex.h.

```

1333     {
1334         const cell& c2 = adjacentCell(c, j1);
1335         const junction& j = *insert(x?x:junction());
1336         spliceAfter(c, j1, j);
1337         insertEdge(j, c);
1338         if(c2)
1339         {
1340             spliceBefore(c2, j1, j);
1341             insertEdge(j, c2);
1342         }
1343         return j;
1344     }

```

17.74.3.27 `template<typename CellContent, typename JunctionContent, typename CellJunctionContent = graph::_EmptyEdgeContent, typename JunctionCellContent = graph::_EmptyEdgeContent, bool compact = false> const junction& vvcomplex::VVComplexGraph<CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::splitWall(const junction &j1, const junction &j2, const junction &x = junction(0)) [inline]`

Split a wall in two and return the new junction.

Definition at line 1310 of file complex.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::splitWall()`.

```

1311     {
1312         const junction& j = *insert(x?x:junction());
1313         forall(const cell& c, neighbors(j1))
1314         {
1315             if(nextTo(c, j1) == j2)
1316             {
1317                 spliceAfter(c, j1, j);
1318                 insertEdge(j, c);
1319             }
1320             else if(prevTo(c, j1) == j2)
1321             {
1322                 spliceBefore(c, j1, j);
1323                 insertEdge(j, c);
1324             }
1325         }
1326         return j;
1327     }

```

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

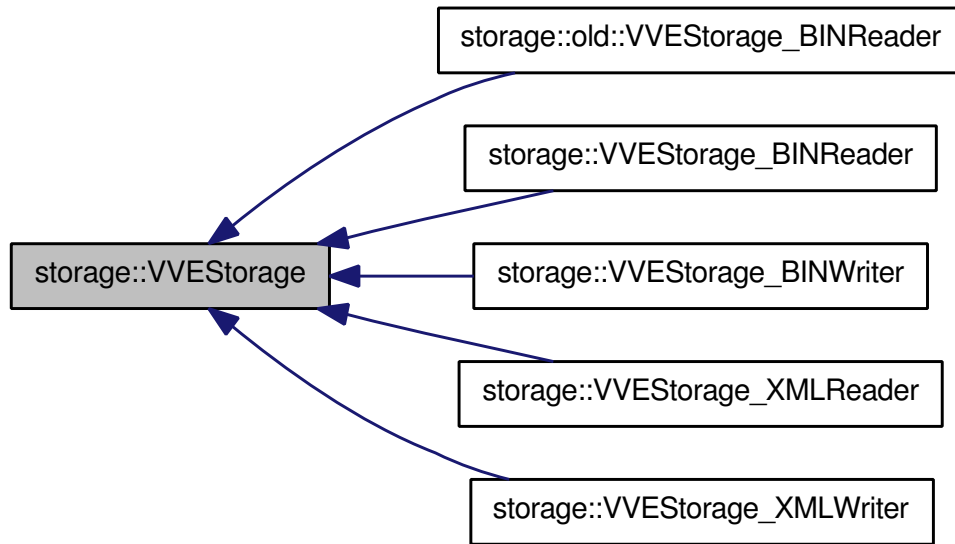
- [vvelib/algorithms/complex.h](#)

17.75 storage::VVEStorage Class Reference

Abstract base class defining the interactions with the persistence module.

```
#include <storage/storage.h>
```

Inheritance diagram for storage::VVEStorage:



Public Types

- typedef std::map< size_t, ReferencePtr > **ref_map**
- typedef std::map< **QString**, ref_map > **ref_map_type**
- typedef ptrdiff_t [reference_t](#)

Type of a reference identifier.

Public Member Functions

- virtual bool [checkNextField](#) (const **QString** &name)=0
Test if there is a field with matching name and type.
- virtual void [clear](#) ()
Clear the internal structures of the object.
- virtual void [clearLastError](#) ()
Remove any error message.
- virtual bool [endCompound](#) ()=0

End the current compound.

- virtual bool **endReference** ()=0
Stop the current reference.
- template<typename T , typename Model >
bool **field** (const **QString** &name, T &value, **Model** *model)
Create a new field, using the model to serialize it.
- template<typename T >
bool **field** (const **QString** &name, T &value)
Create a new field.
- virtual **QString** **filename** () const =0
Name of the file being processed.
- const **QString** & **fileVersion** () const
Version of the file.
- virtual bool **ignore** (const **QString** &)
Instruct a reader to ignore the next field if called name.
- int **lastError** () const
Retrieve the identifier for the last error.
- **QString** **lastErrorString** () const
Retrieve a string describing the last error that occurred, if any.
- virtual bool **reader** ()=0
- template<typename Ptr >
bool **reference** (const **QString** &name, const **QString** &type, Ptr &value, **Model** *model)
Create a reference to a field, using the model to serialize it.
- template<typename Ptr >
bool **reference** (const **QString** &name, const **QString** &type, Ptr &value)
Create a reference to a field.
- virtual bool **serialize** (const **QString** &filename, **Model** *model)=0
Load or save depending on the object.
- virtual void **setLastError** (int error, const **QString** &error_str)
Change the text of the last error.
- virtual bool **setOption** (int, int)
Change an option.

- virtual bool [startCompound](#) (const **QString** &name)=0
Start a new compound.
- virtual int [startReference](#) (const **QString** &name, const **QString** &ref_type, [reference_t](#) ref)=0
Start a reference.
- int [version](#) () const
Integer representing the version of the file for the model.
- virtual bool [writer](#) ()=0
- virtual [~VVEStorage](#) ()
Pure virtual destructor to allow for inheritance.

Fields with standard types

- virtual bool [field](#) (const **QString** &name, std::string &value)=0
Create a std::string field.
- virtual bool [field](#) (const **QString** &name, **QString** &value)=0
*Create a **QString** field.*
- virtual bool [field](#) (const **QString** &name, long double &value)=0
Create a long double field.
- virtual bool [field](#) (const **QString** &name, double &value)=0
Create a double field.
- virtual bool [field](#) (const **QString** &name, float &value)=0
Create a float field.
- virtual bool [field](#) (const **QString** &name, unsigned long long &value)=0
Create a unsigned long long integer field.
- virtual bool [field](#) (const **QString** &name, unsigned long &value)=0
Create a unsigned long integer field.
- virtual bool [field](#) (const **QString** &name, unsigned int &value)=0
Create an unsigned integer field.
- virtual bool [field](#) (const **QString** &name, unsigned short &value)=0
Create a unsigned short integer field.
- virtual bool [field](#) (const **QString** &name, unsigned char &value)=0
Create a unsigned char field.
- virtual bool [field](#) (const **QString** &name, long long &value)=0
Create a long long integer field.

- virtual bool [field](#) (const **QString** &name, long &value)=0
Create a long integer field.
- virtual bool [field](#) (const **QString** &name, int &value)=0
Create an integer field.
- virtual bool [field](#) (const **QString** &name, short &value)=0
Create a short integer field.
- virtual bool [field](#) (const **QString** &name, signed char &value)=0
Create a signed char field.
- virtual bool [field](#) (const **QString** &name, char &value)=0
Create a char field.
- virtual bool [field](#) (const **QString** &name, wchar_t &value)=0
Create a wchar_t field.
- virtual bool [field](#) (const **QString** &name, bool &value)=0
Create a boolean field.

Public Attributes

- **ref_map_type** references

Protected Member Functions

- template<typename Ptr >
bool **find_reference** (const **QString** &type, int ref, Ptr &value)

Protected Attributes

- **QString** **file_version**
- int **last_error**
- **QString** **last_error_string**
- int **version_number**

17.75.1 Detailed Description

Abstract base class defining the interactions with the persistence module.

Definition at line 387 of file storage.h.

17.75.2 Member Typedef Documentation

17.75.2.1 typedef ptrdiff_t storage::VVEStorage::reference_t

Type of a reference identifier.

Reimplemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINWriter](#), [storage::VVEStorage_BINReader](#), [storage::VVEStorage_XMLWriter](#), and [storage::VVEStorage_XMLReader](#).

Definition at line 391 of file storage.h.

17.75.3 Constructor & Destructor Documentation

17.75.3.1 virtual storage::VVEStorage::~~VVEStorage () [inline, virtual]

Pure virtual destructor to allow for inheritance.

Definition at line 405 of file storage.h.

```
405 {}
```

17.75.4 Member Function Documentation

17.75.4.1 virtual bool storage::VVEStorage::checkNextField (const QString & name) [pure virtual]

Test if there is a field with matching name and type.

For writer, this should always return False.

Implemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINWriter](#), [storage::VVEStorage_BINReader](#), [storage::VVEStorage_XMLWriter](#), and [storage::VVEStorage_XMLReader](#).

17.75.4.2 void storage::VVEStorage::clear () [virtual]

Clear the internal structures of the object.

Definition at line 21 of file storage.cpp.

```
22 {  
23     references.clear();  
24 }
```

17.75.4.3 virtual void storage::VVEStorage::clearLastError () [inline, virtual]

Remove any error message.

Definition at line 645 of file storage.h.

References `storage::NO__ERROR`.

```
645 { last_error = NO__ERROR; last_error_string = QString(); }
```

17.75.4.4 **virtual bool storage::VVEStorage::endCompound () [pure virtual]**

End the current compound.

Implemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINWriter](#), [storage::VVEStorage_BINReader](#), [storage::VVEStorage_XMLWriter](#), and [storage::VVEStorage_XMLReader](#).

Referenced by `field()`.

17.75.4.5 **virtual bool storage::VVEStorage::endReference () [pure virtual]**

Stop the current reference.

Implemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINWriter](#), [storage::VVEStorage_BINReader](#), [storage::VVEStorage_XMLWriter](#), and [storage::VVEStorage_XMLReader](#).

Referenced by `reference()`.

17.75.4.6 **template<typename T , typename Model > bool storage::VVEStorage::field (const QString & name, T & value, Model * model) [inline]**

Create a new field, using the model to serialize it.

Parameters

- name* Name of the field
- value* Variable to associate
- model* [Model](#) used for serialization

Returns

Always true when writing, or False if reading and the field does not exist

Definition at line 550 of file storage.h.

References `endCompound()`, `QString::isEmpty()`, `Model::serialize()`, and `startCompound()`.


```
551     {
552         bool result = false;
553         bool no_name = name.isEmpty();
554         if(startCompound(name))
555         {
556             result = model->serialize(*this, value);
557             result &= endCompound();
558         }
559         return result;
560     }
```

17.75.4.7 template<typename T> bool storage::VVEStorage::field (const QString & name, T & value) [inline]

Create a new field.

Parameters

name Name of the field
value Variable to associate

Returns

Always true when writing, or False if reading and the field does not exist

Definition at line 526 of file storage.h.

References endCompound(), storage::serialization(), and startCompound().

```
527     {
528         bool result = false;
529         if(startCompound(name))
530         {
531             result = serialization(*this, value);
532             if(!result)
533                 return false;
534             result &= endCompound();
535         }
536         return result;
537     }
```

17.75.4.8 virtual bool storage::VVEStorage::field (const QString & name, std::string & value) [pure virtual]

Create a std::string field.

17.75.4.9 virtual bool storage::VVEStorage::field (const QString & name, QString & value) [pure virtual]

Create a QString field.

17.75.4.10 `virtual bool storage::VVEStorage::field (const QString & name,
long double & value) [pure virtual]`

Create a long double field.

17.75.4.11 `virtual bool storage::VVEStorage::field (const QString & name,
double & value) [pure virtual]`

Create a double field.

17.75.4.12 `virtual bool storage::VVEStorage::field (const QString & name,
float & value) [pure virtual]`

Create a float field.

17.75.4.13 `virtual bool storage::VVEStorage::field (const QString & name,
unsigned long long & value) [pure virtual]`

Create a unsigned long long integer field.

17.75.4.14 `virtual bool storage::VVEStorage::field (const QString & name,
unsigned long & value) [pure virtual]`

Create a unsigned long integer field.

17.75.4.15 `virtual bool storage::VVEStorage::field (const QString & name,
unsigned int & value) [pure virtual]`

Create an unsigned integer field.

17.75.4.16 `virtual bool storage::VVEStorage::field (const QString & name,
unsigned short & value) [pure virtual]`

Create a unsigned short integer field.

17.75.4.17 `virtual bool storage::VVEStorage::field (const QString & name,
unsigned char & value) [pure virtual]`

Create a unsigned char field.

17.75.4.18 `virtual bool storage::VVEStorage::field (const QString & name,
long long & value) [pure virtual]`

Create a long long integer field.

17.75.4.19 virtual bool storage::VVEStorage::field (const QString & *name*, long & *value*) [pure virtual]

Create a long integer field.

17.75.4.20 virtual bool storage::VVEStorage::field (const QString & *name*, int & *value*) [pure virtual]

Create an integer field.

17.75.4.21 virtual bool storage::VVEStorage::field (const QString & *name*, short & *value*) [pure virtual]

Create a short integer field.

17.75.4.22 virtual bool storage::VVEStorage::field (const QString & *name*, signed char & *value*) [pure virtual]

Create a signed char field.

17.75.4.23 virtual bool storage::VVEStorage::field (const QString & *name*, char & *value*) [pure virtual]

Create a char field.

17.75.4.24 virtual bool storage::VVEStorage::field (const QString & *name*, wchar_t & *value*) [pure virtual]

Create a wchar_t field.

17.75.4.25 virtual bool storage::VVEStorage::field (const QString & *name*, bool & *value*) [pure virtual]

Create a boolean field.

Referenced by reference(), vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >::serialize(), cell_system::CellSystem< TissueClass, RealModel >::serialize(), cell_system::CellSystemJunction::serialize(), and cell_system::CellSystemCell::serialize().

17.75.4.26 virtual QString storage::VVEStorage::filename () const [pure virtual]

Name of the file being processed.

Implemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINWriter](#), [storage::VVEStorage_BINReader](#), [storage::VVEStorage_XMLWriter](#), and [storage::VVEStorage_XMLReader](#).

17.75.4.27 `const QString& storage::VVEStorage::fileVersion () const [inline]`

Version of the file.

Definition at line 658 of file storage.h.

```
658 { return file_version; }
```

17.75.4.28 `virtual bool storage::VVEStorage::ignore (const QString &) [inline, virtual]`

Instruct a reader to ignore the next field if called `name`.

This method is useful only for truly serial reader (like the BIN one)

Reimplemented in [storage::VVEStorage_BINReader](#).

Definition at line 432 of file storage.h.

```
432 { return true; }
```

17.75.4.29 `int storage::VVEStorage::lastError () const [inline]`

Retrieve the identifier for the last error.

See also

[STORAGE_ERROR](#)

Definition at line 633 of file storage.h.

```
633 { return last_error; }
```

17.75.4.30 `QString storage::VVEStorage::lastErrorString () const [inline]`

Retrieve a string describing the last error that occurred, if any.

Definition at line 626 of file storage.h.

```
626 { return last_error_string; }
```

17.75.4.31 virtual bool storage::VVEStorage::reader() [pure virtual]**Returns**

True if the object is a storage reader

Implemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINWriter](#), [storage::VVEStorage_BINReader](#), [storage::VVEStorage_XMLWriter](#), and [storage::VVEStorage_XMLReader](#).

Referenced by [reference\(\)](#).

17.75.4.32 template<typename Ptr> bool storage::VVEStorage::reference(const QString & name, const QString & type, Ptr & value, Model * model) [inline]

Create a reference to a field, using the model to serialize it.

The value is supposed to be a pointer on the variable, not the variable itself.

Note

This could be typically a smart pointer.

Definition at line 598 of file [storage.h](#).

References [endReference\(\)](#), [field\(\)](#), [reader\(\)](#), and [startReference\(\)](#).

```

599     {
600         int ref = startReference(name, type, (char*)(&*value) - (char*)0);
601         if(ref < 0)
602             return false;
603         if(!find_reference(type, ref, value))
604         {
605             if(reader())
606                 value = create_object(value);
607             if(!field("", *value, model))
608             {
609                 return false;
610             }
611             references[type][ref] = make_reference(value);
612         }
613         return endReference();
614     }

```

17.75.4.33 template<typename Ptr> bool storage::VVEStorage::reference(const QString & name, const QString & type, Ptr & value) [inline]

Create a reference to a field.

The value is supposed to be a pointer on the variable, not the variable itself.

Note

This could be typically a smart pointer.

Definition at line 571 of file storage.h.

References `endReference()`, `field()`, `reader()`, and `startReference()`.

```

572     {
573         int ref = startReference(name, type, (char*)&*value) - (char*)0);
574         if(ref < 0)
575             return false;
576         if(!find_reference(type, ref, value))
577         {
578             if(reader())
579                 value = create_object(value);
580             if(!field("", *value))
581             {
582                 return false;
583             }
584             references[type][ref] = make_reference(value);
585         }
586         return endReference();
587     }

```

17.75.4.34 **virtual bool storage::VVEStorage::serialize (const QString & filename, Model *model) [pure virtual]**

Load or save depending on the object.

Implemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINWriter](#), [storage::VVEStorage_BINReader](#), [storage::VVEStorage_XMLWriter](#), and [storage::VVEStorage_XMLReader](#).

17.75.4.35 **virtual void storage::VVEStorage::setLastError (int error, const QString &error_str) [inline, virtual]**

Change the text of the last error.

Any specific implementation might alter the text.

Reimplemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINReader](#), and [storage::VVEStorage_XMLReader](#).

Definition at line 640 of file storage.h.

References `storage::NO__ERROR`.

Referenced by `storage::VVEStorage_XMLWriter::endCompound()`, `storage::VVEStorage_BINWriter::endCompound()`, `storage::VVEStorage_XMLWriter::endReference()`, `storage::VVEStorage_XMLWriter::serialize()`, and `storage::VVEStorage_BINWriter::serialize()`.

```

640 { if(last_error == NO__ERROR) { last_error = error; last_error_string = error_str
    ; } }

```

17.75.4.36 virtual bool storage::VVEStorage::setOption (int, int) [inline, virtual]

Change an option.

The options are implementation-dependent. If the option is not supported, the function returns 0.

Reimplemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINReader](#), [storage::VVEStorage_XMLWriter](#), and [storage::VVEStorage_XMLReader](#).

Definition at line 413 of file storage.h.

```
413 { return false; }
```

17.75.4.37 virtual bool storage::VVEStorage::startCompound (const QString & name) [pure virtual]

Start a new compound.

Useful to organize complicated data structure

When writing, always returns true. When reading, returns true only if the compound exists.

Implemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINWriter](#), [storage::VVEStorage_BINReader](#), [storage::VVEStorage_XMLWriter](#), and [storage::VVEStorage_XMLReader](#).

Referenced by field().

17.75.4.38 virtual int storage::VVEStorage::startReference (const QString & name, const QString & ref_type, reference_t ref) [pure virtual]

Start a reference.

All operations preceding the call to stopReference should be stored in a section reserved for references.

References should be loaded only on need.

When writing, always returns true. When reading, returns true only if the reference exists. If the reference does not exist, it will create it, so the reader has to set the fields.

Returns

the reference identifier or -1 if no reference is available.

Implemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINWriter](#), [storage::VVEStorage_BINReader](#), [storage::VVEStorage_XMLWriter](#), and [storage::VVEStorage_XMLReader](#).

Referenced by reference().

17.75.4.39 int storage::VVEStorage::version () const [inline]

Integer representing the version of the file for the model.

This number is defined for a given model and will never appear in a file.

Definition at line 653 of file storage.h.

```
653 { return version_number; }
```

17.75.4.40 virtual bool storage::VVEStorage::writer () [pure virtual]**Returns**

True is the object is a storage writer

Implemented in [storage::old::VVEStorage_BINReader](#), [storage::VVEStorage_BINWriter](#), [storage::VVEStorage_BINReader](#), [storage::VVEStorage_XMLWriter](#), and [storage::VVEStorage_XMLReader](#).

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

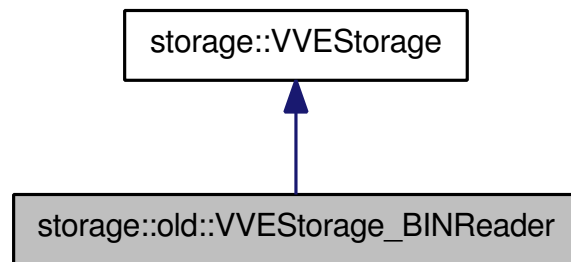
- [vvelib/storage/storage.h](#)
- [vvelib/storage/storage.cpp](#)

17.76 storage::old::VVEStorage_BINReader Class Reference

Implementation reading an BIN file.

```
#include <storage/storage.h>
```

Inheritance diagram for storage::old::VVEStorage_BINReader:



Public Types

- typedef ptrdiff_t [reference_t](#)
Type of a reference identifier.

Public Member Functions

- bool [checkNextField](#) (const **QString** &name)
Test if there is a field with matching name and type.
- virtual bool [endCompound](#) ()
End the current compound.
- virtual bool [endReference](#) ()
Stop the current reference.
- virtual **QString** [filename](#) () const
Name of the file being processed.
- virtual bool [reader](#) ()
- virtual bool [serialize](#) (const **QString** &filename, **Model** *model)
Actually read the file.
- virtual void [setLastError](#) (int error, const **QString** &err)
Change the text of the last error.

- virtual bool [setOption](#) (int option, int value)
Change an option.
- virtual bool [startCompound](#) (const **QString** &name)
Start a new compound.
- virtual int [startReference](#) (const **QString** &name, const **QString** &ref_type, [reference_t](#) ref)
Start a reference.
- virtual bool [writer](#) ()

Protected Member Functions

- bool **checkFileStatus** ()
- void **elementUsed** ()
- const **QString** & **nextElementName** ()
- unsigned char **nextElementType** ()
- template<typename Read , typename Store >
bool **readData** (**QFile** *file, Store &value)
- template<typename T >
bool **readValue** (const **QString** &name, T &value)

Protected Attributes

- **QFile** * **file**
- bool **has_next_element_name**
- **QString** **next_element_name**
- int **type_check**
- NUMBER_CLASS **type_class** [NB_STORAGE_TYPES]
- TYPES **type_conversion** [NB_STORAGE_TYPES]
- size_t **type_size** [NB_STORAGE_TYPES]

17.76.1 Detailed Description

Implementation reading an BIN file.

Definition at line 439 of file storage_bin.cpp.

17.76.2 Member Typedef Documentation

17.76.2.1 typedef ptrdiff_t storage::old::VVEStorage_BINReader::reference_t

Type of a reference identifier.

Reimplemented from [storage::VVEStorage](#).

Definition at line 443 of file storage_bin.cpp.

17.76.3 Member Function Documentation

17.76.3.1 bool storage::old::VVEStorage_BINReader::checkNextField (const QString & name) [virtual]

Test if there is a field with matching name and type.

For writer, this should always return False.

Implements [storage::VVEStorage](#).

Definition at line 771 of file storage_bin.cpp.

```
772     {  
773         const QString& next_name = nextElementName();  
774         return next_name == name;  
775     }
```

17.76.3.2 bool storage::old::VVEStorage_BINReader::endCompound () [virtual]

End the current compound.

Implements [storage::VVEStorage](#).

Definition at line 612 of file storage_bin.cpp.

```
613     {  
614         return true;  
615     }
```

17.76.3.3 bool storage::old::VVEStorage_BINReader::endReference () [virtual]

Stop the current reference.

Implements [storage::VVEStorage](#).

Definition at line 651 of file storage_bin.cpp.

```
652     {  
653         return true;  
654     }
```

17.76.3.4 virtual QString storage::old::VVEStorage_BINReader::filename () const [inline, virtual]

Name of the file being processed.

Implements [storage::VVEStorage](#).

Definition at line 474 of file storage_bin.cpp.

References [QFile::fileName\(\)](#).

```

475     {
476         if(file)
477             return file->fileName();
478         else
479             return "";
480     }

```

17.76.3.5 virtual bool storage::old::VVEStorage_BINReader::reader () [inline, virtual]

Returns

True is the object is a storage reader

Implements [storage::VVEStorage](#).

Definition at line 461 of file storage_bin.cpp.

```

461 { return true; }

```

17.76.3.6 bool storage::old::VVEStorage_BINReader::serialize (const QString & filename, Model * model) [virtual]

Actually read the file.

Implements [storage::VVEStorage](#).

Definition at line 535 of file storage_bin.cpp.

References [QString::arg\(\)](#), [storage::BAD_CONTENT](#), [QFile::close\(\)](#), [storage::find_type\(\)](#), [storage::IO_ERROR](#), [storage::NO_ERROR](#), [QFile::open\(\)](#), [Model::serialize\(\)](#), [setLastError\(\)](#), and [Model::versionNumber\(\)](#).

Referenced by [storage::VVEStorage_BINReader::serialize\(\)](#).

```

536     {
537         last_error = NO__ERROR;
538         last_error_string = QString();
539
540         QFile _file(filename);
541         file = &_file;
542         has_next_element_name = false;
543         references.clear();
544         if(!file->open(QIODevice::ReadOnly))
545         {
546             setLastError(IO_ERROR, QString("Could not open file '%1' for reading").ar
g(filename));
547             return false;
548         }
549         QByteArray first_line = file->readLine();

```

```

550         if(first_line != "VVE BIN\n")
551         {
552             setLastError(BAD_CONTENT, QString("File '%1' is not a VVE BIN file").arg(
filename));
553             return false;
554         }
555
556         read(file, file_version);
557         file_version = QString("%1 BIN").arg(file_version);
558         version_number = model->versionNumber(file_version);
559
560         if(!checkFileStatus()) return false;
561
562         // Read type conversion
563         TYPES type;
564         while((type = read_type(file)) != T_INVALID)
565         {
566             unsigned char t;
567             unsigned char s;
568             read(file, t);
569             read(file, s);
570             NUMBER_CLASS k = (NUMBER_CLASS)t;
571             size_t size = (size_t)s;
572             type_conversion[type] = find_type(k, size);
573             type_size[type] = size;
574             type_class[type] = k;
575         }
576
577         type_conversion[T_STRING] = T_STRING;
578         type_conversion[T_BOOL] = T_BOOL;
579
580         bool result = model->serialize(*this);
581         file->close();
582         file = 0;
583         return result;
584     }

```

17.76.3.7 void storage::old::VVEStorage_BINReader::setLastError (int *error*, const QString & *error_str*) [virtual]

Change the text of the last error.

Any specific implementation might alter the text.

Reimplemented from [storage::VVEStorage](#).

Definition at line 777 of file storage_bin.cpp.

References [QString::arg\(\)](#), and [QFile::pos\(\)](#).

Referenced by [serialize\(\)](#), [startCompound\(\)](#), and [startReference\(\)](#).

```

778     {
779         last_error = error;
780         last_error_string = QString("Error at position %1 of the file:\n%2").arg(fi
le->pos()).arg(err);
781     }

```

17.76.3.8 `bool storage::old::VVEStorage_BINReader::setOption (int, int)` [**virtual**]

Change an option.

The options are implementation-dependent. If the option is not supported, the function returns 0.

Reimplemented from [storage::VVEStorage](#).

Definition at line 511 of file `storage_bin.cpp`.

References `storage::TCO_Exact`, `storage::TCO_NoCheck`, `storage::TCO_Permissive`, `storage::TCO_PermissiveNoWarning`, `storage::TCO_Strict`, and `storage::TypeChecking`.

```

512     {
513         switch(option)
514         {
515             case TypeChecking:
516                 switch(value)
517                 {
518                     case TCO_Exact:
519                     case TCO_Strict:
520                     case TCO_Permissive:
521                     case TCO_PermissiveNoWarning:
522                     case TCO_NoCheck:
523                         type_check = value;
524                         return true;
525                     default:
526                         return false;
527                 }
528                 break;
529             default:
530                 return false;
531         }
532     }
```

17.76.3.9 `bool storage::old::VVEStorage_BINReader::startCompound (const QString & name)` [**virtual**]

Start a new compound.

Useful to organize complicated data structure

When writing, always returns true. When reading, returns true only if the compound exists.

Implements [storage::VVEStorage](#).

Definition at line 598 of file `storage_bin.cpp`.

References `storage::BAD_CONTENT`, `QString::isEmpty()`, and `setLastError()`.

```

599     {
600         if(name.isEmpty())
601             return true;
602         const QString& next_name = nextElementName();
```

```

603     if(next_name != name)
604     {
605         setLastError(BAD_CONTENT, QString("Next compound has name '%1' instead of
the expected '%2'").arg(next_name, name));
606         return false;
607     }
608     elementUsed();
609     return true;
610 }

```

17.76.3.10 int storage::old::VVEStorage_BINReader::startReference (const QString & name, const QString & ref_type, reference_t ref) [virtual]

Start a reference.

All operations preceding the call to stopReference should be stored in a section reserved for references.

References should be loaded only on need.

When writing, always returns true. When reading, returns true only if the reference exists. If the reference does not exist, it will create it, so the reader has to set the fields.

Returns

the reference identifier or -1 if no reference is available.

Implements [storage::VVEStorage](#).

Definition at line 627 of file storage_bin.cpp.

References [storage::BAD_CONTENT](#), [QString::isEmpty\(\)](#), [setLastError\(\)](#), and [storage::TYPE_CHECK_ERROR](#).

```

628     {
629         if(!name.isEmpty())
630         {
631             const QString& next_name = nextElementName();
632             if(next_name != name)
633             {
634                 setLastError(BAD_CONTENT, QString("Error, expected reference named '%1'
but found element named '%2'").arg(name, next_name));
635                 return -1;
636             }
637         }
638         unsigned char type = nextElementType();
639         if(type != BT_START_REFERENCE)
640         {
641             setLastError(TYPE_CHECK_ERROR, QString("Element '%1' is not a reference")
.arg(name));
642             return -1;
643         }
644         qint32 id;
645         read(file, id);
646         if(!checkFileStatus()) return false;
647         elementUsed();

```

```
648         return id;
649     }
```

17.76.3.11 virtual bool storage::old::VVEStorage_BINReader::writer () [inline, virtual]

Returns

True is the object is a storage writer

Implements [storage::VVEStorage](#).

Definition at line 462 of file storage_bin.cpp.

```
462 { return false; }
```

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

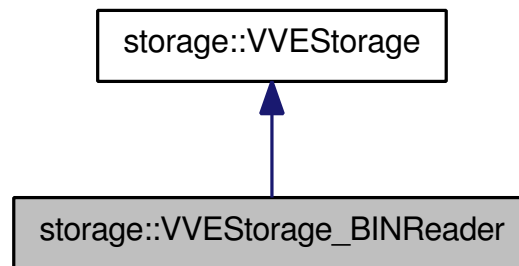
- vvelib/storage/storage_bin.cpp

17.77 storage::VVEStorage_BINReader Class Reference

Implementation reading an BIN file.

```
#include <storage/storage.h>
```

Inheritance diagram for storage::VVEStorage_BINReader:



Public Types

- typedef ptrdiff_t [reference_t](#)
Type of a reference identifier.

Public Member Functions

- bool [checkNextField](#) (const **QString** &name)
Test if there is a field with matching name and type.
- virtual bool [endCompound](#) ()
End the current compound.
- virtual bool [endReference](#) ()
Stop the current reference.
- virtual **QString** [filename](#) () const
Name of the file being processed.
- virtual bool [ignore](#) (const **QString** &name)
Instruct a reader to ignore the next field if called name.
- virtual bool [reader](#) ()
- virtual bool [serialize](#) (const **QString** &filename, [Model](#) *model)
Actually read the file.

- virtual void [setLastError](#) (int error, const **QString** &err)
Change the text of the last error.
- virtual bool [setOption](#) (int option, int value)
Change an option.
- virtual bool [startCompound](#) (const **QString** &name)
Start a new compound.
- virtual int [startReference](#) (const **QString** &name, const **QString** &ref_type, [reference_t](#) ref)
Start a reference.
- virtual bool [writer](#) ()

Protected Member Functions

- bool **checkFileStatus** ()
- void **elementUsed** ()
- const **QString** & **nextElementName** ()
- unsigned char **nextElementType** ()
- template<typename Read , typename Store >
bool **readData** (**QFile** *file, Store &value)
- template<typename T >
bool **readValue** (const **QString** &name, T &value)
- bool **resolveFrame** (bool named_field=true)
- bool **skipNext** ()
- bool **skipToEnd** ()

Protected Attributes

- **QFile** * **file**
- std::stack< [Frame](#) > [frame_stack](#)
Stack of frames. A frame groups fields.
- bool **has_next_element_name**
- **QString** **next_element_name**
- int [top_level_compounds](#)
Number of unnamed compounds on top of the stack.
- int **type_check**
- NUMBER_CLASS **type_class** [NB_STORAGE_TYPES]
- TYPES **type_conversion** [NB_STORAGE_TYPES]
- size_t **type_size** [NB_STORAGE_TYPES]

17.77.1 Detailed Description

Implementation reading an BIN file.

Definition at line 111 of file storage_bin.h.

17.77.2 Member Typedef Documentation

17.77.2.1 typedef ptrdiff_t storage::VVEStorage_BINReader::reference_t

Type of a reference identifier.

Reimplemented from [storage::VVEStorage](#).

Definition at line 115 of file storage_bin.h.

17.77.3 Member Function Documentation

17.77.3.1 bool storage::VVEStorage_BINReader::checkNextField (const QString & name) [virtual]

Test if there is a field with matching name and type.

For writer, this should always return False.

Implements [storage::VVEStorage](#).

Definition at line 1194 of file storage_bin.cpp.

```
1195     {
1196         resolveFrame();
1197         const QString& next_name = nextElementName();
1198         DEBUG_PRINT("Checking if field " << name << " is the next one: " << (next_name == name) << endl);
1199         return next_name == name;
1200     }
```

17.77.3.2 bool storage::VVEStorage_BINReader::endCompound () [virtual]

End the current compound.

Implements [storage::VVEStorage](#).

Definition at line 1002 of file storage_bin.cpp.

References `frame_stack`, `setLastError()`, and `storage::USER_ERROR`.

```
1003     {
1004         if(frame_stack.empty())
1005         {
1006             setLastError(USER_ERROR, "Trying to close a compound that wasn't opened");
1007             return false;
1008         }
```

```

1008     }
1009     if (frame_stack.top().unnamed_subframes > 0)
1010     {
1011         DEBUG_PRINT("Ending unnamed compound" << endl);
1012         frame_stack.top().unnamed_subframes--;
1013         return true;
1014     }
1015     DEBUG_PRINT("Ending named compound" << endl);
1016     frame_stack.pop();
1017     return skipToEnd();
1018 }

```

17.77.3.3 bool storage::VVEStorage_BINReader::endReference () [virtual]

Stop the current reference.

Implements [storage::VVEStorage](#).

Definition at line 1061 of file storage_bin.cpp.

```

1062 {
1063     DEBUG_PRINT("End of reference" << endl);
1064     return skipToEnd();
1065 }

```

17.77.3.4 virtual QString storage::VVEStorage_BINReader::filename () const [inline, virtual]

Name of the file being processed.

Implements [storage::VVEStorage](#).

Definition at line 147 of file storage_bin.h.

References [QFile::fileName\(\)](#).

```

148 {
149     if (file)
150         return file->fileName();
151     else
152         return "";
153 }

```

17.77.3.5 bool storage::VVEStorage_BINReader::ignore (const QString &) [virtual]

Instruct a reader to ignore the next field if called `name`.

This method is useful only for truly serial reader (like the BIN one)

Reimplemented from [storage::VVEStorage](#).

Definition at line 938 of file storage_bin.cpp.

References storage::BAD_CONTENT, and setLastError().

```

939 {
940     const QString& next_name = nextElementName();
941     if(next_name != name)
942     {
943         setLastError(BAD_CONTENT, QString("Trying skip element %1 when the next ele
ment is %2").arg(name).arg(next_name));
944         return false;
945     }
946     return skipNext();
947 }
```

17.77.3.6 virtual bool storage::VVEStorage_BINReader::reader () [inline, virtual]

Returns

True is the object is a storage reader

Implements [storage::VVEStorage](#).

Definition at line 134 of file storage_bin.h.

Referenced by serialize().

```

134 { return true; }
```

17.77.3.7 bool storage::VVEStorage_BINReader::serialize (const QString & filename, Model * model) [virtual]

Actually read the file.

Implements [storage::VVEStorage](#).

Definition at line 785 of file storage_bin.cpp.

References QString::arg(), storage::BAD_CONTENT, QFile::close(), storage::find_type(), frame_stack, storage::IO_ERROR, storage::NO__ERROR, QFile::open(), reader(), Model::serialize(), storage::old::VVEStorage_BINReader::serialize(), setLastError(), storage::USER_ERROR, and Model::versionNumber().

```

786 {
787     last_error = NO__ERROR;
788     last_error_string = QString();
789
790     QFile _file(filename);
791     file = &_file;
792     has_next_element_name = false;
793     references.clear();
794     if(!file->open(QIODevice::ReadOnly))
795     {
```

```
796     setLastError(IO_ERROR, QString("Could not open file '%1' for reading").arg(
filename));
797     return false;
798 }
799 QByteArray first_line = file->readLine();
800 if(first_line == "VVE BIN\n")
801 {
802     _file.close();
803     old::VVEStorage_BINReader reader;
804     return reader.serialize(filename, model);
805 }
806 if(first_line != "VVE BIN2\n")
807 {
808     setLastError(BAD_CONTENT, QString("File '%1' is not a VVE BIN file").arg(fi
lename));
809     return false;
810 }
811 DEBUG_PRINT("Starting reader for BIN2" << endl);
812
813 read(file, file_version);
814 file_version = QString("%1 BIN").arg(file_version);
815 version_number = model->versionNumber(file_version);
816
817 if(!checkFileStatus()) return false;
818
819 // Read type conversion
820 TYPES type;
821 while((type = read_type(file)) != T_INVALID)
822 {
823     unsigned char t;
824     unsigned char s;
825     read(file, t);
826     read(file, s);
827     NUMBER_CLASS k = (NUMBER_CLASS)t;
828     size_t size = (size_t)s;
829     type_conversion[type] = find_type(k, size);
830     type_size[type] = size;
831     type_class[type] = k;
832 }
833
834 type_conversion[T_STRING] = T_STRING;
835 type_conversion[T_BOOL] = T_BOOL;
836
837 frame_stack.push(Frame(0,true)); // Prevent from using the top-level frame
838 bool result = model->serialize(*this);
839 file->close();
840 if(result)
841 {
842     if(frame_stack.empty())
843     {
844         setLastError(USER_ERROR, "One too many compound have been closed.");
845         return false;
846     }
847     frame_stack.pop();
848     if(!frame_stack.empty())
849     {
850         setLastError(USER_ERROR, QString("%1 compounds have been opened and not c
losed.").arg(frame_stack.size()));
851         return false;
852     }
853 }
854 file = 0;
```

```
855     return result;
856 }
```

17.77.3.8 void storage::VVEStorage_BINReader::setLastError (int *error*, const QString & *error_str*) [virtual]

Change the text of the last error.

Any specific implementation might alter the text.

Reimplemented from [storage::VVEStorage](#).

Definition at line 1202 of file storage_bin.cpp.

References [QString::arg\(\)](#), and [QFile::pos\(\)](#).

Referenced by [endCompound\(\)](#), [ignore\(\)](#), [serialize\(\)](#), [startCompound\(\)](#), and [startReference\(\)](#).

```
1203 {
1204     last_error = error;
1205     last_error_string = QString("Error at position %1 of the file:\n%2").arg(file
->pos()).arg(err);
1206 }
```

17.77.3.9 bool storage::VVEStorage_BINReader::setOption (int, int) [virtual]

Change an option.

The options are implementation-dependent. If the option is not supported, the function returns 0.

Reimplemented from [storage::VVEStorage](#).

Definition at line 401 of file storage_bin.cpp.

References [storage::TCO_Exact](#), [storage::TCO_NoCheck](#), [storage::TCO_Permissive](#), [storage::TCO_PermissiveNoWarning](#), [storage::TCO_Strict](#), and [storage::TypeChecking](#).

```
402 {
403     switch(option)
404     {
405         case TypeChecking:
406             switch(value)
407             {
408                 case TCO_Exact:
409                 case TCO_Strict:
410                 case TCO_Permissive:
411                 case TCO_PermissiveNoWarning:
412                 case TCO_NoCheck:
413                     type_check = value;
414                     return true;
415                 default:
```

```

416         return false;
417     }
418     break;
419     default:
420         return false;
421 }
422 }
```

17.77.3.10 bool storage::VVEStorage_BINReader::startCompound (const QString & name) [virtual]

Start a new compound.

Useful to organize complicated data structure

When writing, always returns true. When reading, returns true only if the compound exists.

Implements [storage::VVEStorage](#).

Definition at line 916 of file storage_bin.cpp.

References [storage::BAD_CONTENT](#), [frame_stack](#), [QString::isEmpty\(\)](#), [QFile::pos\(\)](#), and [setLastError\(\)](#).

```

917 {
918     if(name.isEmpty())
919     {
920         DEBUG_PRINT("Start unnamed compound " << endl);
921         frame_stack.top().unnamed_subframes++; // Add an unnamed stack
922         return true;
923     }
924     if(!resolveFrame())
925         return false;
926     DEBUG_PRINT("Start compound " << name << " on pos " << file->pos () << endl);

927     const QString& next_name = nextElementName();
928     if(next_name != name)
929     {
930         setLastError(BAD_CONTENT, QString("Next compound has name '%1' instead of t
he expected '%2'").arg(next_name, name));
931         return false;
932     }
933     elementUsed();
934     frame_stack.push(Frame());
935     return true;
936 }
```

17.77.3.11 int storage::VVEStorage_BINReader::startReference (const QString & name, const QString & ref_type, reference_t ref) [virtual]

Start a reference.

All operations preceding the call to stopReference should be stored in a section reserved for references.

References should be loaded only on need.

When writing, always returns true. When reading, returns true only if the reference exists. If the reference does not exist, it will create it, so the reader has to set the fields.

Returns

the reference identifier or -1 if no reference is available.

Implements [storage::VVEStorage](#).

Definition at line 1030 of file storage_bin.cpp.

References [storage::BAD_CONTENT](#), [QString::isEmpty\(\)](#), [setLastError\(\)](#), and [storage::TYPE_CHECK_ERROR](#).

```

1031  {
1032      if(!resolveFrame(!name.isEmpty()))
1033          return false;
1034      if(!name.isEmpty())
1035      {
1036          DEBUG_PRINT("Starting reference named " << name << endl);
1037          const QString& next_name = nextElementName();
1038          if(next_name != name)
1039          {
1040              setLastError(BAD_CONTENT, QString("Error, expected reference named '%1' but found element named '%2'").arg(name, next_name));
1041              return -1;
1042          }
1043      }
1044      else
1045      {
1046          DEBUG_PRINT("Starting unnamed reference" << endl);
1047      }
1048      unsigned char type = nextElementType();
1049      if(type != BT_START_REFERENCE)
1050      {
1051          setLastError(TYPE_CHECK_ERROR, QString("Element '%1' is not a reference").arg(name));
1052          return -1;
1053      }
1054      qint32 id;
1055      read(file, id);
1056      if(!checkFileStatus()) return false;
1057      elementUsed();
1058      return id;
1059  }

```

17.77.3.12 virtual bool storage::VVEStorage_BINReader::writer () [inline, virtual]

Returns

True if the object is a storage writer

Implements [storage::VVEStorage](#).

Definition at line 135 of file storage_bin.h.

```
135 { return false; }
```

17.77.4 Member Data Documentation

17.77.4.1 `std::stack<Frame> storage::VVEStorage_BINReader::frame_stack` [protected]

Stack of frames. A frame groups fields.

Definition at line 173 of file `storage_bin.h`.

Referenced by `endCompound()`, `serialize()`, and `startCompound()`.

17.77.4.2 `int storage::VVEStorage_BINReader::top_level_compounds` [protected]

Number of unnamed compounds on top of the stack.

That can only happen if the named compound is replaced by an unnamed field.

Definition at line 180 of file `storage_bin.h`.

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

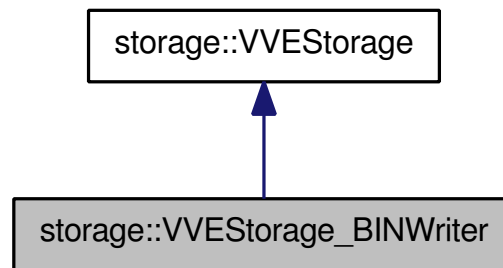
- `vvelib/storage/storage_bin.h`
- `vvelib/storage/storage_bin.cpp`

17.78 storage::VVEStorage_BINWriter Class Reference

Implementation writing an BIN file.

```
#include <storage/storage.h>
```

Inheritance diagram for storage::VVEStorage_BINWriter:



Public Types

- typedef ptrdiff_t [reference_t](#)
Type of a reference identifier.

Public Member Functions

- bool [checkNextField](#) (const **QString** &)
Test if there is a field with matching name and type.
- virtual bool [endCompound](#) ()
End the current compound.
- virtual bool [endReference](#) ()
Stop the current reference.
- virtual **QString** [filename](#) () const
Name of the file being processed.
- virtual bool [reader](#) ()
- virtual bool [serialize](#) (const **QString** &filename, [Model](#) *model)
Actually save the file.
- virtual bool [startCompound](#) (const **QString** &name)
Start a new compound.

- virtual int [startReference](#) (const **QString** &name, const **QString** &ref_type, [reference_t](#) ref)

Start a reference.

- virtual bool [writer](#) ()

Protected Types

- typedef std::map< [reference_t](#), int > **ref_t**

Protected Member Functions

- bool **checkFileStatus** ()
- bool **createElement** (const **QString** &name, unsigned char type)
- bool **resolveFrame** (bool named_field=true)
- template<typename T >
bool **writeField** (const **QString** &name, T &value)

Protected Attributes

- **QFile** * **file**
- std::stack< [Frame](#) > [frame_stack](#)
Stack of frames. A frame group fields.
- std::map< **QString**, [ref_t](#) > **references**

17.78.1 Detailed Description

Implementation writing an BIN file.

Definition at line 47 of file storage_bin.h.

17.78.2 Member Typedef Documentation

17.78.2.1 typedef ptrdiff_t storage::VVEStorage_BINWriter::reference_t

Type of a reference identifier.

Reimplemented from [storage::VVEStorage](#).

Definition at line 51 of file storage_bin.h.

17.78.3 Member Function Documentation

17.78.3.1 bool storage::VVEStorage_BINWriter::checkNextField (const QString & name) [inline, virtual]

Test if there is a field with matching name and type.

For writer, this should always return False.

Implements [storage::VVEStorage](#).

Definition at line 74 of file storage_bin.h.

```
74 { return false; }
```

17.78.3.2 bool storage::VVEStorage_BINWriter::endCompound () [virtual]

End the current compound.

Implements [storage::VVEStorage](#).

Definition at line 317 of file storage_bin.cpp.

References [frame_stack](#), [storage::VVEStorage::setLastError\(\)](#), and [storage::USER_ERROR](#).

```
318 {
319     if(frame_stack.empty())
320     {
321         setLastError(USER_ERROR, "Error, trying to finish a compound that didn't st
art");
322         return false;
323     }
324     if(frame_stack.top().unnamed_subframes > 0)
325     {
326         DEBUG_PRINT("Ending unnamed compound" << endl);
327         frame_stack.top().unnamed_subframes--;
328         DEBUG_PRINT("    unnamed compounds left: " << frame_stack.top().unnamed_sub
frames << endl);
329         return true;
330     }
331     if(!frame_stack.top().has_fields)
332     {
333         unsigned char type = BT_START_COMPOUND;
334         write(file, type);
335         if(!checkFileStatus()) return false;
336     }
337     DEBUG_PRINT("Ending named compound" << endl);
338     frame_stack.pop();
339     write_id(file, ""); // Mark of an end of compound
340     return true;
341 }
```

17.78.3.3 `bool storage::VVEStorage_BINWriter::endReference ()` [virtual]

Stop the current reference.

Implements [storage::VVEStorage](#).

Definition at line 366 of file `storage_bin.cpp`.

```
367 {  
368     DEBUG_PRINT("Ending reference" << endl);  
369     write_id(file, ""); // Mark end of reference too  
370     return true;  
371 }
```

17.78.3.4 `virtual QString storage::VVEStorage_BINWriter::filename () const` [inline, virtual]

Name of the file being processed.

Implements [storage::VVEStorage](#).

Definition at line 79 of file `storage_bin.h`.

References `QFile::fileName()`.

```
80 {  
81     if(file)  
82         return file->fileName();  
83     else  
84         return "";  
85 }
```

17.78.3.5 `virtual bool storage::VVEStorage_BINWriter::reader ()` [inline, virtual]

Returns

True is the object is a storage reader

Implements [storage::VVEStorage](#).

Definition at line 67 of file `storage_bin.h`.

```
67 { return false; }
```

17.78.3.6 `bool storage::VVEStorage_BINWriter::serialize (const QString & filename, Model * model)` [virtual]

Actually save the file.

Implements [storage::VVEStorage](#).

Definition at line 189 of file storage_bin.cpp.

References [QString::arg\(\)](#), [QFile::close\(\)](#), [QFile::error\(\)](#), [frame_stack](#), [storage::IO_ERROR](#), [storage::NO__ERROR](#), [QFile::open\(\)](#), [Model::serialize\(\)](#), [storage::VVEStorage::setLastError\(\)](#), [storage::USER_ERROR](#), [Model::version\(\)](#), and [Model::versionNumber\(\)](#).

```

190  {
191      last_error = NO__ERROR;
192      last_error_string = QString();
193
194      QFile _file(filename);
195      file = &_file;
196      references.clear();
197      if(!file->open(QIODevice::WriteOnly))
198      {
199          DEBUG_PRINT("error: " << file->error() << ": " << file->errorString() << endl);
200          setLastError(IO_ERROR, QString("Could not open file '%1' for writing").arg(
201              filename));
202          return false;
203          file->write("VVE BIN2\n");
204
205          file_version = model->version();
206          write(file, file_version);
207
208          file_version = QString("%1 BIN").arg(file_version);
209          version_number = model->versionNumber(file_version);
210
211          // Now write the size of the number types
212          #define WRITE_TYPE_SIZE(id, T) \
213              write(file, (unsigned char)id); \
214              write(file, (unsigned char)TypeTraits<T>::type); \
215              write(file, (unsigned char)sizeof(T));
216          FOR_ALL_TYPEIDS(WRITE_TYPE_SIZE)
217
218          #undef WRITE_TYPE_SIZE
219
220          write(file, (unsigned char)T_INVALID);
221
222          if(!checkFileStatus()) return false;
223
224          frame_stack.push(Frame(0,true)); // Prevent replacing the top-level
225          bool result = model->serialize(*this);
226          file->close();
227          if(result)
228          {
229              if(frame_stack.empty())
230              {
231                  setLastError(USER_ERROR, "One too many compound have been closed.");
232                  return false;
233              }
234              frame_stack.pop();
235              if(!frame_stack.empty())
236              {
237                  setLastError(USER_ERROR, QString("%1 compounds have been opened and not c
238                      losed.").arg(frame_stack.size()));

```

```
240         return false;
241     }
242 }
243 file = 0;
244 return result;
245 }
```

17.78.3.7 bool storage::VVEStorage_BINWriter::startCompound (const QString & name) [virtual]

Start a new compound.

Useful to organize complicated data structure

When writing, always returns true. When reading, returns true only if the compound exists.

Implements [storage::VVEStorage](#).

Definition at line 298 of file storage_bin.cpp.

References [frame_stack](#), and [QString::isEmpty\(\)](#).

```
299 {
300     if(!name.isEmpty())
301     {
302         DEBUG_PRINT("Starting compounds " << name << endl);
303         if(!resolveFrame())
304             return false;
305         write_id(file, name);
306         frame_stack.push(Frame());
307         return checkFileStatus();
308     }
309     else
310     {
311         DEBUG_PRINT("Starting unnamed compounds " << endl);
312         frame_stack.top().unnamed_subframes++; // Add one invisible element
313     }
314     return true;
315 }
```

17.78.3.8 int storage::VVEStorage_BINWriter::startReference (const QString & name, const QString & ref_type, reference_t ref) [virtual]

Start a reference.

All operations preceding the call to stopReference should be stored in a section reserved for references.

References should be loaded only on need.

When writing, always returns true. When reading, returns true only if the reference exists. If the reference does not exist, it will create it, so the reader has to set the fields.

Returns

the reference identifier or -1 if no reference is available.

Implements [storage::VVEStorage](#).

Definition at line 343 of file storage_bin.cpp.

```

344 {
345     DEBUG_PRINT("Creating reference named " << name << endl);
346     ref_t& ref_map = references[ref_type];
347
348     if(!createElement(name, BT_START_REFERENCE))
349         return false;
350
351     ref_t::iterator found = ref_map.find(ref);
352     int id = -1;
353     if(found == ref_map.end())
354     {
355         id = (int)ref_map.size();
356         ref_map[ref] = id;
357     }
358     else
359     {
360         id = found->second;
361     }
362     write(file, (qint32)id);
363     return id;
364 }
```

17.78.3.9 virtual bool storage::VVEStorage_BINWriter::writer() [**inline**, **virtual**]

Returns

True is the object is a storage writer

Implements [storage::VVEStorage](#).

Definition at line 68 of file storage_bin.h.

```
68 { return true; }
```

17.78.4 Member Data Documentation

17.78.4.1 std::stack<Frame> storage::VVEStorage_BINWriter::frame_stack [**protected**]

Stack of frames. A frame group fields.

Definition at line 97 of file storage_bin.h.

Referenced by endCompound(), serialize(), and startCompound().

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

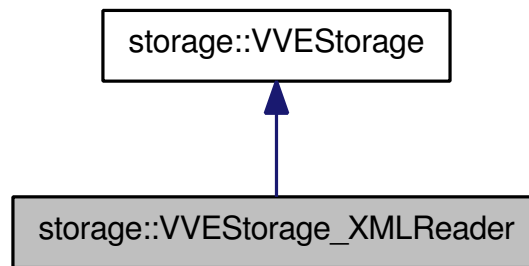
- vvelib/storage/storage_bin.h
- vvelib/storage/storage_bin.cpp

17.79 storage::VVEStorage_XMLReader Class Reference

Implementation reading an XML file.

```
#include <storage/storage.h>
```

Inheritance diagram for storage::VVEStorage_XMLReader:



Public Types

- typedef ptrdiff_t [reference_t](#)
Type of a reference identifier.

Public Member Functions

- virtual bool [checkNextField](#) (const **QString** &name)
Test if there is a field with matching name and type.
- virtual bool [endCompound](#) ()
End the current compound.
- virtual bool [endReference](#) ()
Stop the current reference.
- virtual **QString** [filename](#) () const
Name of the file being processed.
- virtual bool [reader](#) ()
- virtual bool [serialize](#) (const **QString** &filename, **Model** *model)
Actually read the file.
- virtual void [setLastError](#) (int error, const **QString** &err)
Change the text of the last error.

- virtual bool [setOption](#) (int option, int value)
Change an option.
- virtual bool [startCompound](#) (const **QString** &name)
Start a new compound.
- virtual int [startReference](#) (const **QString** &name, const **QString** &ref_type, [reference_t](#) ref)
Start a reference.
- virtual bool [writer](#) ()

Protected Member Functions

- template<typename From , typename To >
bool **extract_value_with_error** (**QDomElement** element, To &result)
- **QDomElement** **extractElement** (const **QString** &name)
- template<typename T >
bool **get_field** (const **QString** &name, T &result)
- void **setError_invalid_conversion** (const **QString** &name, const **QString** &value, const **QString** &type_from, const **QString** &type_to)
- void **setError_invalid_value** (const **QString** &name, const **QString** &type, const **QString** &value)
- void **setError_novalue** (const **QString** &name)
- bool **validType** (**QDomElement** element, TYPES type)

Protected Attributes

- **QString** **_filename**
- std::list< **QDomElement** > **compound_stack**
- **QDomDocument** **doc**
- **QDomElement** **root**
- int **type_checking**

17.79.1 Detailed Description

Implementation reading an XML file.

Definition at line 87 of file storage_xml.h.

17.79.2 Member Typedef Documentation

17.79.2.1 typedef ptrdiff_t storage::VVEStorage_XMLReader::reference_t

Type of a reference identifier.

Reimplemented from [storage::VVEStorage](#).

Definition at line 91 of file storage_xml.h.

17.79.3 Member Function Documentation

17.79.3.1 **bool storage::VVEStorage_XMLReader::checkNextField (const QString & name) [virtual]**

Test if there is a field with matching name and type.

For writer, this should always return False.

Implements [storage::VVEStorage](#).

Definition at line 395 of file storage_xml.cpp.

```
396 {
397     QDomElement last = compound_stack.back();
398     QDomElement child = last.firstChildElement(name);
399     if(child.isNull())
400         return false;
401     return true;
402 }
```

17.79.3.2 **bool storage::VVEStorage_XMLReader::endCompound () [virtual]**

End the current compound.

Implements [storage::VVEStorage](#).

Definition at line 416 of file storage_xml.cpp.

References [storage::NO_FIELD_ERROR](#), and [setLastError\(\)](#).

```
417 {
418     if(compound_stack.empty())
419     {
420         setLastError(NO_FIELD_ERROR, "Trying to end a compound not started");
421         return false;
422     }
423     compound_stack.pop_back();
424     return true;
425 }
```

17.79.3.3 **bool storage::VVEStorage_XMLReader::endReference () [virtual]**

Stop the current reference.

Implements [storage::VVEStorage](#).

Definition at line 460 of file storage_xml.cpp.

References `setLastError()`, and `storage::USER_ERROR`.

```
461 {  
462     if (compound_stack.empty())  
463     {  
464         setLastError(USER_ERROR, "Trying to close a reference that wasn't opened.")  
465     }  
466     return false;  
467     compound_stack.pop_back();  
468     return true;  
469 }
```

17.79.3.4 virtual QString storage::VVEStorage_XMLReader::filename () const [inline, virtual]

Name of the file being processed.

Implements [storage::VVEStorage](#).

Definition at line 120 of file `storage_xml.h`.

```
120 { return _filename; }
```

17.79.3.5 virtual bool storage::VVEStorage_XMLReader::reader () [inline, virtual]

Returns

True is the object is a storage reader

Implements [storage::VVEStorage](#).

Definition at line 109 of file `storage_xml.h`.

```
109 { return true; }
```

17.79.3.6 bool storage::VVEStorage_XMLReader::serialize (const QString & *filename*, Model * *model*) [virtual]

Actually read the file.

Implements [storage::VVEStorage](#).

Definition at line 198 of file `storage_xml.cpp`.

References `QString::arg()`, `QDomElement::attribute()`, `storage::BAD_CONTENT`, `QFile::close()`, `storage::IO_ERROR`, `storage::NO_ERROR`, `QFile::open()`, `Model::serialize()`, `QDomDocument::setContent()`, `setLastError()`, and `Model::versionNumber()`.

```

199  {
200      _filename = filename;
201      doc = QDomDocument("VVEStorage");
202      compound_stack.clear();
203
204      last_error = NO__ERROR;
205      last_error_string = QString();
206
207      QFile file(filename);
208      if(!file.open(QIODevice::ReadOnly))
209      {
210          setLastError(IO_ERROR, QString("Could not open file '%1' for reading").arg(
211          filename));
212          return false;
213      }
214      if(!doc.setContent(&file))
215      {
216          file.close();
217          setLastError(BAD_CONTENT, QString("File '%1' : Content is not valid XML.").
218          arg(filename));
219          return false;
220      }
221
222      root = doc.firstChildElement("VVEStorage");
223      if(root.isNull())
224      {
225          setLastError(BAD_CONTENT, QString("File '%1' has no root element").arg(file
226          name));
227          return false;
228      }
229
230      file_version = QString("%1 XML").arg(root.attribute("version", ""));
231      version_number = model->versionNumber(file_version);
232      compound_stack.push_back(root);
233      bool result = model->serialize(*this);
234      _filename = QString();
235      return result;
236  }

```

17.79.3.7 void storage::VVEStorage_XMLReader::setLastError (int error, const QString & error_str) [virtual]

Change the text of the last error.

Any specific implementation might alter the text.

Reimplemented from [storage::VVEStorage](#).

Definition at line 234 of file storage_xml.cpp.

References [QString::arg\(\)](#), [forall](#), and [QDomElement::tagName\(\)](#).

Referenced by [endCompound\(\)](#), [endReference\(\)](#), [serialize\(\)](#), [startCompound\(\)](#), and [startReference\(\)](#).

```

235  {
236      last_error = error;
237      last_error_string = err;
238      last_error_string += "\nStack of elements:";
239      QDomElement last;

```

```
240     forall(const QDomElement& el, compound_stack)
241     {
242         if(el != last)
243             last_error_string += QString("\n<%1>").arg(el.tagName());
244         last = el;
245     }
246 }
```

17.79.3.8 bool storage::VVEStorage_XMLReader::setOption (int, int) [virtual]

Change an option.

The options are implementation-dependent. If the option is not supported, the function returns 0.

Reimplemented from [storage::VVEStorage](#).

Definition at line 270 of file storage_xml.cpp.

References [storage::TCO_Exact](#), [storage::TCO_NoCheck](#), [storage::TCO_Permissive](#), [storage::TCO_PermissiveNoWarning](#), [storage::TCO_Strict](#), and [storage::TypeChecking](#).

```
271 {
272     switch(option)
273     {
274         case TypeChecking:
275             switch(value)
276             {
277                 case TCO_Exact:
278                 case TCO_Strict:
279                 case TCO_Permissive:
280                 case TCO_PermissiveNoWarning:
281                 case TCO_NoCheck:
282                     type_checking = value;
283                     return true;
284                 default:
285                     return false;
286             }
287             break;
288         default:
289             return false;
290     }
291 }
```

17.79.3.9 bool storage::VVEStorage_XMLReader::startCompound (const QString & name) [virtual]

Start a new compound.

Useful to organize complicated data structure

When writing, always returns true. When reading, returns true only if the compound exists.

Implements [storage::VVEStorage](#).

Definition at line 404 of file `storage_xml.cpp`.

References `storage::NO_FIELD_ERROR`, and `setLastError()`.

```

405  {
406      QDomElement new_compound = extractElement(name);
407      if(new_compound.isNull())
408      {
409          setLastError(NO_FIELD_ERROR, QString("Compound '%1' not found").arg(name));
410
411          return false;
412      }
413      compound_stack.push_back(new_compound);
414      return true;
415  }
```

17.79.3.10 `int storage::VVEStorage_XMLReader::startReference(const QString & name, const QString & ref_type, reference_t ref) [virtual]`

Start a reference.

All operations preceding the call to `stopReference` should be stored in a section reserved for references.

References should be loaded only on need.

When writing, always returns true. When reading, returns true only if the reference exists. If the reference does not exist, it will create it, so the reader has to set the fields.

Returns

the reference identifier or -1 if no reference is available.

Implements [storage::VVEStorage](#).

Definition at line 427 of file `storage_xml.cpp`.

References `QDomElement::attribute()`, `QString::isEmpty()`, `storage::NO_FIELD_ERROR`, `storage::REFERENCE_ERROR`, and `setLastError()`.

```

428  {
429      QDomElement place_holder = extractElement(name);
430      if(place_holder.isNull())
431      {
432          setLastError(NO_FIELD_ERROR, QString("Reference '%1' not found").arg(name));
433      }
434      return -1;
435  }
436
437      QString type = place_holder.attribute("ref_type");
438      if(type.isEmpty())
439      {
440          setLastError(REFERENCE_ERROR, QString("Element '%1' is not a reference but is used as one.").arg(name));
441      }
```



```
440     return -1;
441 }
442
443 if(type != ref_type)
444 {
445     setLastError(REFERENCE_ERROR, QString("Element '%1' is a reference of type
'%2', reference of type '%3' was expected.").arg(name, type, ref_type));
446     return -1;
447 }
448
449 bool ok;
450 uint ref = place_holder.attribute("ref_id", "").toUInt(&ok);
451 if(!ok)
452 {
453     setLastError(REFERENCE_ERROR, QString("Reference '%1' has no id").arg(name)
);
454     return -1;
455 }
456 compound_stack.push_back(place_holder);
457 return (int)ref;
458 }
```

17.79.3.11 virtual bool storage::VVEStorage_XMLReader::writer () [inline, virtual]

Returns

True is the object is a storage writer

Implements [storage::VVEStorage](#).

Definition at line 110 of file storage_xml.h.

```
110 { return false; }
```

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

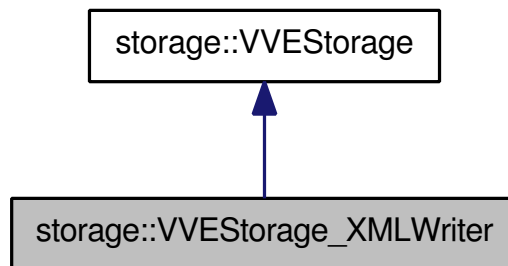
- [vvelib/storage/storage_xml.h](#)
- [vvelib/storage/storage_xml.cpp](#)

17.80 storage::VVEStorage_XMLWriter Class Reference

Implementation writing an XML file.

```
#include <storage/storage.h>
```

Inheritance diagram for storage::VVEStorage_XMLWriter:



Public Types

- typedef ptrdiff_t [reference_t](#)
Type of a reference identifier.

Public Member Functions

- virtual bool [checkNextField](#) (const **QString** &)
Test if there is a field with matching name and type.
- virtual bool [endCompound](#) ()
End the current compound.
- virtual bool [endReference](#) ()
Stop the current reference.
- virtual **QString** [filename](#) () const
Name of the file being processed.
- virtual bool [reader](#) ()
- virtual bool [serialize](#) (const **QString** &filename, **Model** *model)
Actually save the file.
- virtual bool [setOption](#) (int option, int value)
Change an option.

- virtual bool [startCompound](#) (const **QString** &name)
Start a new compound.
- virtual int [startReference](#) (const **QString** &name, const **QString** &ref_type, [reference_t](#) ref)
Start a reference.
- virtual bool [writer](#) ()

Protected Types

- typedef std::map< [reference_t](#), uint > **ref_map_t**
- typedef std::map< **QString**, ref_map_t > **ref_map_typed_t**

Protected Member Functions

- **QDomElement** [createElement](#) (const **QString** &name, const **QString** &type=**QString**())

Protected Attributes

- **QString** [_filename](#)
- std::list< **QDomElement** > [compound_stack](#)
- **QDomDocument** [doc](#)
- **QDomElement** [root](#)
- [ref_map_typed_t](#) [typed_references_map](#)
- bool [write_type](#)

17.80.1 Detailed Description

Implementation writing an XML file.

Definition at line 28 of file storage_xml.h.

17.80.2 Member Typedef Documentation

17.80.2.1 typedef ptrdiff_t storage::VVEStorage_XMLWriter::reference_t

Type of a reference identifier.

Reimplemented from [storage::VVEStorage](#).

Definition at line 32 of file storage_xml.h.

17.80.3 Member Function Documentation

17.80.3.1 virtual bool storage::VVEStorage_XMLWriter::checkNextField (const QString & name) [inline, virtual]

Test if there is a field with matching name and type.

For writer, this should always return False.

Implements [storage::VVEStorage](#).

Definition at line 57 of file storage_xml.h.

```
57 { return false; }
```

17.80.3.2 bool storage::VVEStorage_XMLWriter::endCompound () [virtual]

End the current compound.

Implements [storage::VVEStorage](#).

Definition at line 125 of file storage_xml.cpp.

References [storage::NO_FIELD_ERROR](#), and [storage::VVEStorage::setLastError\(\)](#).

```
126 {  
127     if (compound_stack.empty())  
128     {  
129         setLastError(NO_FIELD_ERROR, "Trying to end a compound not started");  
130         return false;  
131     }  
132     compound_stack.pop_back();  
133     return true;  
134 }
```

17.80.3.3 bool storage::VVEStorage_XMLWriter::endReference () [virtual]

Stop the current reference.

Implements [storage::VVEStorage](#).

Definition at line 161 of file storage_xml.cpp.

References [storage::VVEStorage::setLastError\(\)](#), and [storage::USER_ERROR](#).

```
162 {  
163     if (compound_stack.empty())  
164     {  
165         setLastError(USER_ERROR, "Trying to close a reference that wasn't opened.");  
166         return false;  
167     }
```

```
168     compound_stack.pop_back();
169     return true;
170 }
```

17.80.3.4 virtual QString storage::VVEStorage_XMLWriter::filename () const [inline, virtual]

Name of the file being processed.

Implements [storage::VVEStorage](#).

Definition at line 59 of file storage_xml.h.

```
59 { return _filename; }
```

17.80.3.5 virtual bool storage::VVEStorage_XMLWriter::reader () [inline, virtual]

Returns

True is the object is a storage reader

Implements [storage::VVEStorage](#).

Definition at line 50 of file storage_xml.h.

```
50 { return false; }
```

17.80.3.6 bool storage::VVEStorage_XMLWriter::serialize (const QString & *filename*, Model * *model*) [virtual]

Actually save the file.

Implements [storage::VVEStorage](#).

Definition at line 55 of file storage_xml.cpp.

References [QString::arg\(\)](#), [QFile::close\(\)](#), [QDomDocument::createElement\(\)](#), [storage::IO_ERROR](#), [QString::isEmpty\(\)](#), [storage::NO__ERROR](#), [QFile::open\(\)](#), [Model::serialize\(\)](#), [QDomElement::setAttribute\(\)](#), [storage::VVEStorage::setLastError\(\)](#), [QDomDocument::toByteArray\(\)](#), [storage::UNKNOWN_ERROR](#), [Model::version\(\)](#), and [Model::versionNumber\(\)](#).

```
56 {
57     _filename = filename;
58     doc = QDomDocument("VVEStorage");
59
60     last_error = NO__ERROR;
61     last_error_string = QString();
62
63     file_version = model->version();
```

```

64
65     typed_references_map.clear();
66     compound_stack.clear();
67
68     root = doc.createElement("VVEStorage");
69     if(!file_version.isEmpty())
70         root.setAttribute("version", file_version);
71     doc.appendChild(root);
72     compound_stack.push_back(root);
73
74     file_version = QString("%1 XML").arg(file_version);
75     version_number = model->versionNumber(file_version);
76
77     if(model->serialize(*this))
78     {
79         QFile file(filename);
80         if(!file.open(QIODevice::WriteOnly))
81         {
82             setLastError(IO_ERROR, QString("Could not open file '%1' for writing").ar
83 g(filename));
84             _filename = QString();
85             return false;
86         }
87         file.write(doc.toByteArray());
88         file.close();
89         _filename = QString();
90         return true;
91     }
92     _filename = QString();
93     if(!last_error)
94     {
95         last_error = UNKNOWN_ERROR;
96         last_error_string += "\nUnkown error.";
97     }
98     return false;
99 }

```

17.80.3.7 bool storage::VVEStorage_XMLWriter::setOption (int, int) [virtual]

Change an option.

The options are implementation-dependent. If the option is not supported, the function returns 0.

Reimplemented from [storage::VVEStorage](#).

Definition at line 30 of file storage_xml.cpp.

References [storage::TCO_Exact](#), [storage::TCO_NoCheck](#), [storage::TCO_-Permissive](#), [storage::TCO_PermissiveNoWarning](#), [storage::TCO_Strict](#), and [storage::TypeChecking](#).

```

31 {
32     switch(option)
33     {
34         case TypeChecking:
35             switch(value)
36             {

```

```
37         case TCO_Exact:
38         case TCO_Strict:
39         case TCO_Permissive:
40         case TCO_PermissiveNoWarning:
41             write_type = true;
42             return true;
43         case TCO_NoCheck:
44             write_type = false;
45             return true;
46         default:
47             return false;
48     }
49     break;
50     default:
51         return false;
52     }
53 }
```

17.80.3.8 bool storage::VVEStorage_XMLWriter::startCompound (const QString & name) [virtual]

Start a new compound.

Useful to organize complicated data structure

When writing, always returns true. When reading, returns true only if the compound exists.

Implements [storage::VVEStorage](#).

Definition at line 118 of file storage_xml.cpp.

```
119 {
120     QDomElement new_compound = createElement(name);
121     compound_stack.push_back(new_compound);
122     return true;
123 }
```

17.80.3.9 int storage::VVEStorage_XMLWriter::startReference (const QString & name, const QString & ref_type, reference_t ref) [virtual]

Start a reference.

All operations preceding the call to stopReference should be stored in a section reserved for references.

References should be loaded only on need.

When writing, always returns true. When reading, returns true only if the reference exists. If the reference does not exist, it will create it, so the reader has to set the fields.

Returns

the reference identifier or -1 if no reference is available.

Implements [storage::VVEStorage](#).

Definition at line 136 of file storage_xml.cpp.

References [QDomElement::setAttribute\(\)](#).

```
137 {
138     QDomElement place_holder = createElement(name);
139
140     ref_map_t& references_map = typed_references_map[ref_type];
141
142     ref_map_t::const_iterator found = references_map.find(ref);
143     if(found == references_map.end())
144     {
145         int id = (int)references_map.size();
146         place_holder.setAttribute("ref_id", id);
147         place_holder.setAttribute("ref_type", ref_type);
148         references_map[ref] = id;
149         compound_stack.push_back(place_holder);
150         return id;
151     }
152     else
153     {
154         place_holder.setAttribute("ref_id", found->second);
155         place_holder.setAttribute("ref_type", ref_type);
156         compound_stack.push_back(place_holder);
157         return found->second;
158     }
159 }
```

17.80.3.10 virtual bool storage::VVEStorage_XMLWriter::writer () [inline, virtual]

Returns

True if the object is a storage writer

Implements [storage::VVEStorage](#).

Definition at line 51 of file storage_xml.h.

```
51 { return true; }
```

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

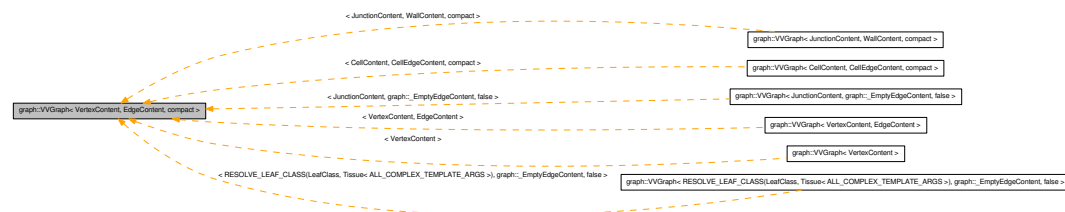
- vvelib/storage/[storage_xml.h](#)
- vvelib/storage/storage_xml.cpp

17.81 graph::VVGraph< VertexContent, EdgeContent, compact > Class Template Reference

Class representing a VV graph.

```
#include <graph/vvgraph.h>
```

Inheritance diagram for graph::VVGraph< VertexContent, EdgeContent, compact >:



Classes

- struct [neighbor_t](#)
Structure maintaining the data for a single neighbor.
- struct [search_result_t](#)
Type of the result of the search for a vertex in a neighborhood.
- struct [single_neighborhood_t](#)
Type of the neighborhood of a vertex.

Public Types

- typedef std::unordered_map< [vertex_t](#), typename neighborhood_t::iterator > [lookup_t](#)
Type of the fast-search map.
- typedef std::list< std::pair< [vertex_t](#), [single_neighborhood_t](#) > > [neighborhood_t](#)
Type of the list of vertexes, together with the neighborhood.
- typedef neighborhood_t::value_type [neighborhood_value_type](#)
Shortcut for the value_type of the neighborhood.
- typedef neighborhood_t::size_type [size_type](#)
Type of a size.
- typedef [vertex_t](#) [value_type](#)

Type of the value of the graph as standard container.

Smart pointer types

- typedef [Arc](#)< EdgeContent > [arc_t](#)
Weak pointer on an arc.
- typedef [Edge](#)< const EdgeContent > [const_edge_t](#)
Weak pointer on a constant edge.
- typedef [Edge](#)< EdgeContent > [edge_t](#)
Weak pointer on an edge.
- typedef [Vertex](#)< VertexContent > [vertex_t](#)
Smart pointer on a vertex.

Iterators

- typedef [util::CircIterator](#)< [neighbor_iterator](#) > [circ_neighbor_iterator](#)
Iterator used to iterate over the neighbors, but specifying the starting point.
- typedef [util::range](#)< [circ_neighbor_iterator](#) > [circ_neighbor_iterator_range](#)
Range of circular iterators.
- typedef [util::CircIterator](#)< [const_neighbor_iterator](#) > [const_circ_neighbor_iterator](#)
Iterator used to iterate over the neighbors, but specifying the starting point.
- typedef [util::range](#)< [const_circ_neighbor_iterator](#) > [const_circ_neighbor_iterator_range](#)
Range of circular iterators.
- typedef [util::SelectMemberIterator](#)< typename [neighborhood_t::const_iterator](#), const [vertex_t](#), &[neighborhood_value_type::first](#) > [const_iterator](#)
Constant iterator on the vertexes.
- typedef [util::SelectMemberIterator](#)< typename [edge_list_t::const_iterator](#), const [vertex_t](#), &[neighbor_t::target](#) > [const_neighbor_iterator](#)
Constant iterator on the neighbors of a vertex.
- typedef [util::range](#)< [const_neighbor_iterator](#) > [const_neighbor_iterator_range](#)
Constant range of the neighbors of a vertex.
- typedef [in_edges_t::const_iterator](#) [ineighbor_iterator](#)
Iterator on the incoming neighbors of a vertex.
- typedef [util::range](#)< [ineighbor_iterator](#) > [ineighbor_iterator_range](#)
Range of the incoming neighbors of a vertex.

- typedef [util::SelectMemberIterator](#)< typename neighborhood_t::iterator, const [vertex_t](#),&neighborhood_value_type::first > [iterator](#)
Iterator on the vertexes.
- typedef [util::SelectMemberIterator](#)< typename edge_list_t::iterator, [vertex_t](#),&[neighbor_t](#)::target > [neighbor_iterator](#)
Iterator on the neighbors of a vertex.
- typedef [util::range](#)< [neighbor_iterator](#) > [neighbor_iterator_range](#)
Range of the neighbors of a vertex.

Public Member Functions

- [VVGraph](#) (const [VVGraph](#) ©)
Copy constructor.
- [VVGraph](#) ()
Default constructor.

Vertex set lookup methods

- const [vertex_t](#) & [any](#) () const
Return a vertex from the graph.
- bool [contains](#) (const [vertex_t](#) &v) const
Test if v is in the graph.
- bool [empty](#) () const
Test if there is any vertex in the graph.
- const [vertex_t](#) & [operator\[\]](#) (size_type idx) const
Return the element of index i d x.
- const [vertex_t](#) & [reference](#) ([vertex_t](#) v) const
Get a reference on the vertex in the graph.
- size_type [size](#) () const
Return the number of vertexes on the graph.

Neighborhood lookup methods

- const [vertex_t](#) & [anyIn](#) (const [vertex_t](#) &v) const
Return a vertex in the neighborhood of v.
- [arc_t](#) [arc](#) (const [vertex_t](#) &src, const [vertex_t](#) &tgt)

Returns the arc (i.e.

- `const_edge_t edge` (const `vertex_t` &src, const `vertex_t` &tgt) const
Returns the edge from src to tgt.
- `edge_t edge` (const `vertex_t` &src, const `vertex_t` &tgt)
Returns the edge from src to tgt.
- `bool empty` (const `vertex_t` &v) const
Test if a vertex has a neighbor.
- `bool flag` (const `vertex_t` &src, const `vertex_t` &neighbor)
Flag a neighbor for quick retrieval.
- `const vertex_t &flagged` (const `vertex_t` &src) const
Return the flagged neighbor or a null vertex is no neighbor have been flagged.
- `const vertex_t &iAnyIn` (const `vertex_t` &v) const
Returns any incoming neighbor of v.
- `bool iEmpty` (const `vertex_t` &v) const
Test if a vertex has an incoming neighbor.
- `ineighbor_iterator_range iNeighbors` (const `vertex_t` &v) const
Return the range of incoming neighbors of v.
- `size_type iValence` (const `vertex_t` &v) const
Returns the number of incoming neighbors of v.
- `const_circ_neighbor_iterator_range neighbors` (const `vertex_t` &v, const `vertex_t` &n) const
Return the range of neighbors of v, starting at n.
- `circ_neighbor_iterator_range neighbors` (const `vertex_t` &v, const `vertex_t` &n)
Return the range of neighbors of v, starting at n.
- `neighbor_iterator_range neighbors` (const `vertex_t` &v)
Return the range of neighbors of v.
- `const_neighbor_iterator_range neighbors` (const `vertex_t` &v) const
Return the constant range of neighbors of v.
- `const vertex_t &nextTo` (const `vertex_t` &v, const `vertex_t` &neighbor, unsigned int n=1) const
Returns the nth vertex after neighbor in the neighborhood of v.
- `const vertex_t &prevTo` (const `vertex_t` &v, const `vertex_t` &ref, unsigned int n=1) const
Returns the nth vertex before ref in the neighborhood of v.

- const [vertex_t](#) & **source** (const [arc_t](#) &arc) const
- const [vertex_t](#) & **source** (const [edge_t](#) &edge) const
Return the source vertex of the edge.
- const [vertex_t](#) & **target** (const [arc_t](#) &arc) const
- const [vertex_t](#) & **target** (const [edge_t](#) &edge) const
Return the target vertex of the edge.
- [size_type](#) **valence** (const [vertex_t](#) &v) const
Returns the number of neighbors of v.

STL compatibility methods

- const_iterator **begin** () const
Returns an iterator on the beginning of the set of vertexes of the graph.
- iterator **begin** ()
Returns an iterator on the beginning of the set of vertexes of the graph.
- void **clear** (iterator it)
Clear the outgoing edges of a vertex.
- [size_type](#) **count** (const [vertex_t](#) &v) const
Count the number of vertexes v in the graph (can be 0 or 1 only).
- const_iterator **end** () const
Returns an iterator on the end of the set of vertexes of the graph.
- iterator **end** ()
Returns an iterator on the end of the set of vertexes of the graph.
- iterator **erase** (iterator pos)
Erase element at position pos.
- void **eraseAllEdges** (iterator it)
Clear the neighborhood of a vertex.
- [circ_neighbor_iterator](#) **eraseEdge** (const [vertex_t](#) &v, [circ_neighbor_iterator](#) pos)
Erase the edge pointed to by the iterator.
- [neighbor_iterator](#) **eraseEdge** (const [vertex_t](#) &v, [neighbor_iterator](#) pos)
Erase the edge pointed to by the iterator.
- const_iterator **find** (const [vertex_t](#) &v) const
Find the vertex v in the graph.
- iterator **find** (const [vertex_t](#) &v)
Find the vertex v in the graph.

- `const_neighbor_iterator findIn` (const `vertex_t` &*v*, const `vertex_t` &*n*) const
*Find the vertex *n* in the neighborhood of *v*.*
- `neighbor_iterator findIn` (const `vertex_t` &*v*, const `vertex_t` &*n*)
*Find the vertex *n* in the neighborhood of *v*.*
- `template<typename Iterator >`
`void insert` (Iterator first, Iterator last)
Insert all the vertexes from first to last-1 in the graph.
- `iterator insert` (iterator pos, const `vertex_t` &*v*)
Insert a new vertex in the graph.
- `void store_vertices` (std::vector< `vertex_t` > &vertices, bool cached=true)
const
Store the vertices in a STL container.

Whole structure methods

- `void clear` ()
Clear the graph.
- `void eraseAllEdges` ()
Clear all the edges in the graph.
- `VVGraph & operator=` (const `VVGraph` &other)
Copy the graph into another graph.
- `bool operator==` (const `VVGraph` &other) const
Test equality for the graphs.
- `template<typename VertexContainer >`
`VVGraph subgraph` (const VertexContainer &vertexes) const
Extract the subgraph containing the vertexes.
- `void swap` (`VVGraph` &other)
Swap the content of two graphs.

Neighborhood edition methods

- `bool clear` (const `vertex_t` &*v*)
Clear the outgoing edges of a vertex.
- `bool eraseAllEdges` (const `vertex_t` &*v*)
Clear the neighborhood of a vertex.
- `size_type eraseEdge` (const `vertex_t` &*src*, const `vertex_t` &*tgt*)
*Erase the edge from *src* to *tgt*.*

- `edge_t insertEdge` (const `vertex_t` &src, const `vertex_t` &tgt)
Insert a new edge in the graph, without ordering.
- `template<typename VertexContainer >`
`std::vector< edge_t > insertEdges` (const `vertex_t` &src, const VertexContainer &targets)
Insert a set of edges, keeping the relative order of the list of edges.
- `edge_t replace` (const `vertex_t` &v, const `vertex_t` &neighbor, const `vertex_t` &new_neighbor)
Replace a vertex by another in a neighborhood.
- `template<typename VertexContainer >`
`std::vector< edge_t > spliceAfter` (const `vertex_t` &v, const `vertex_t` &neighbor, const VertexContainer &new_neighbor)
Splice a list of vertices after a neighbor.
- `edge_t spliceAfter` (const `vertex_t` &v, const `vertex_t` &neighbor, const `vertex_t` &new_neighbor)
Insert neighbor in the neighborhood of v after reference.
- `template<typename VertexContainer >`
`std::vector< edge_t > spliceBefore` (const `vertex_t` &v, const `vertex_t` &neighbor, const VertexContainer &new_neighbor)
Splice a list of vertices before a neighbor.
- `edge_t spliceBefore` (const `vertex_t` &v, const `vertex_t` &neighbor, const `vertex_t` &new_neighbor)
Insert neighbor in the neighborhood of v before reference.

Vertex set edition methods

- `size_type erase` (const `vertex_t` &v)
Remove a vertex from the graph.
- `iterator insert` (const `vertex_t` &v)
Insert a new vertex in the graph.

Protected Types

- `typedef search_result_t< const single_neighborhood_t, int_const_neighbor_iterator > const_neighbor_found_t`
Constant result of a search in the neighborhood.
- `typedef neighbor_t::edge_list_t edge_list_t`
Type of the list of outgoing neighbors.

- typedef neighbor_t::base **EdgeCache**
- typedef __vvgraph_set< compact, vertex_t >::type in_edges_t
Type of the list of sources of the in-edges.
- typedef edge_list_t::const_iterator int_const_neighbor_iterator
Constant iterator on the outgoing edges.
- typedef edge_list_t::iterator int_neighbor_iterator
Iterator on the outgoing edges.
- typedef search_result_t< single_neighborhood_t, int_neighbor_iterator >
neighbor_found_t
Result of a search in the neighborhood.
- typedef single_neighborhood_t::base **VertexCache**

Protected Member Functions

- const_neighbor_found_t findInVertex (const vertex_t &v, const vertex_t &neighbor) const
Constant version of findInVertex(const vertex_t&, const vertex_t&).
- neighbor_found_t findInVertex (const vertex_t &v, const vertex_t &neighbor)
Find a vertex neighbor in the neighborhood of v.
- neighborhood_t::const_iterator findVertex (const vertex_t &v) const
Constant version of findVertex(const vertex_t&).
- neighborhood_t::iterator findVertex (const vertex_t &v)
Find a vertex in the graph and returns the iterator on it.
- std::pair< in_neighbor_iterator, bool > insertInEdge (const vertex_t &v, const vertex_t &new_neighbor)
Insert v in the in_edges of new_neighbor and return true if the insertion was actually done, or false if v was already there.
- void undoInEdge (const vertex_t &new_neighbor, in_neighbor_iterator &it)
Undo the insertion of an in edge as a roll-back mechanism if the whole edge insertion fails.
- void updateEdgeCache (const vertex_t &v, typename edge_list_t::iterator new_edge_it, const vertex_t &in_edge)
Update the edge cache information of vertex v in its neighborhood and the in_edge too.
- void updateVertexCache (const vertex_t &v, typename neighborhood_t::iterator it)

Update the vertex cache for vertex v.

Protected Attributes

- `intptr_t graph_id`

Unique graph identifier.

- `lookup_t lookup`

Hash map to optimise look-up.

- `neighborhood_t neighborhood`

Main data structure containing everything.

17.81.1 Detailed Description

```
template<typename VertexContent, typename EdgeContent = _-  
EmptyEdgeContent, bool compact = false> class graph::VVGraph< Ver-  
texContent, EdgeContent, compact >
```

Class representing a VV graph. A VV graph is an oriented rotational graph. That is, the edges of a given vertex have a circular order. That is, if an edge has three edges `e1`, `e2`, `e3`, there is no first edge, but we can say `e1 < e2 < e3 < e1`. In that case, '`<`' denote the succession.

`VertexContent` is the type used to hold the vertex data.

`EdgeContent` is the type used to hold the edge data. If no type is specified for `EdgeContent`, then an empty structure is used.

17.81.2 Performance notes

The graph includes two caching mechanisms to improve performances when iterating over the graph and retrieve edge information:

1. while iterating over the graph, the vertex cache the current iterator. So that accessing the neighborhood of the vertex is constant time (i.e. it's dereferencing the cached iterator).
2. while iterating over the neighborhood of a vertex, the target vertex (i.e. the one returned by the iteration) cache the iterator on the neighborhood. So that any operation using that specific edge (i.e. getting edge data, splicing, replacing or erasing) is also constant time.

To make this caching mechanism correct, the cached data are never copied. This means, to make use of the caching mechanism, you have to iterate using constant references (references will end up with compilation errors if used on the graph):

```
forall(const vertex &v, S)
{
    // Here the vertex v has the vertex cache
    forall(const vertex &n, S.neighbors(S)) // Makes use of the cache
    {
        S.edge(v,n)->pos = Point3d(1,2,3); // Uses the edge cache
        S.edge(n,v)->pos = Point3d(3,2,1); // Do not use any cache
    }
    v->pos = Point3d(0,0,0); // Valid
}
```

Warning

Do not iterate using constant references if you ever remove the vertex you iterate on ! (i.e. remove from the graph if you iterate on the graph or remove from the neighborhood if you iterate on the neighborhood)

Even though this code manipulate constant references, modifying the data pointed by the vertex is allowed. The vertexes are coded to allow that. That means if you are using vertexes in a dictionary or a set, using their content to compare them, you must not modify them ! The compiler won't guard you against it.

Definition at line 362 of file vvgraph.h.

17.81.3 Member Typedef Documentation

17.81.3.1 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> typedef Arc<EdgeContent> graph::VVGraph< VertexContent, EdgeContent, compact >::arc_t`

Weak pointer on an arc.

When an object of this type is destroyed, the content of the primary edge (i.e. the edge source -> target) is copied into the other.

Definition at line 394 of file vvgraph.h.

17.81.3.2 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> typedef util::CircIterator<neighbor_iterator> graph::VVGraph< VertexContent, EdgeContent, compact >::circ_neighbor_iterator`

Iterator used to iterate over the neighbors, but specifying the starting point.

Definition at line 615 of file vvgraph.h.

17.81.3.3 `template<typename VertexContent, typename
EdgeContent = _EmptyEdgeContent, bool compact =
false> typedef util::range<circ_neighbor_iterator>
graph::VVGraph< VertexContent, EdgeContent, compact
>::circ_neighbor_iterator_range`

Range of circular iterators.

Definition at line 620 of file vvgraph.h.

17.81.3.4 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false>
typedef util::CircIterator<const_neighbor_iterator>
graph::VVGraph< VertexContent, EdgeContent, compact
>::const_circ_neighbor_iterator`

Iterator used to iterate over the neighbors, but specifying the starting point.

Definition at line 653 of file vvgraph.h.

17.81.3.5 `template<typename VertexContent, typename
EdgeContent = _EmptyEdgeContent, bool compact =
false> typedef util::range<const_circ_neighbor_iterator>
graph::VVGraph< VertexContent, EdgeContent, compact
>::const_circ_neighbor_iterator_range`

Range of circular iterators.

Definition at line 658 of file vvgraph.h.

17.81.3.6 `template<typename VertexContent, typename EdgeContent =
_EmptyEdgeContent, bool compact = false> typedef Edge<const
EdgeContent> graph::VVGraph< VertexContent, EdgeContent,
compact >::const_edge_t`

Weak pointer on a constant edge.

Definition at line 387 of file vvgraph.h.

17.81.3.7 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
util::SelectMemberIterator<typename neighborhood_t::const_
iterator, const vertex_t, &neighborhood_value_type::first>
graph::VVGraph< VertexContent, EdgeContent, compact
>::const_iterator`

Constant iterator on the vertexes.

Definition at line 599 of file vvgraph.h.

17.81.3.8 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false>
typedef search_result_t<const single_neighborhood_t,
int_const_neighbor_iterator> graph::VVGraph< VertexContent,
EdgeContent, compact >::const_neighbor_found_t [protected]`

Constant result of a search in the neighborhood.

Definition at line 1731 of file vvgraph.h.

17.81.3.9 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
util::SelectMemberIterator<typename edge_list_t::const_iterator,
const vertex_t, &neighbor_t::target> graph::VVGraph<
VertexContent, EdgeContent, compact >::const_neighbor_iterator`

Constant iterator on the neighbors of a vertex.

Definition at line 641 of file vvgraph.h.

17.81.3.10 `template<typename VertexContent, typename
EdgeContent = _EmptyEdgeContent, bool compact =
false> typedef util::range<const_neighbor_iterator>
graph::VVGraph< VertexContent, EdgeContent, compact
>::const_neighbor_iterator_range`

Constant range of the neighbors of a vertex.

The `first` element of the pair is a constant iterator on the beginning of the range, the `second` element is the past-the-end constant iterator.

Definition at line 647 of file vvgraph.h.

17.81.3.11 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
neighbor_t::edge_list_t graph::VVGraph< VertexContent,
EdgeContent, compact >::edge_list_t [protected]`

Type of the list of outgoing neighbors.

Definition at line 498 of file vvgraph.h.

17.81.3.12 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
Edge<EdgeContent> graph::VVGraph< VertexContent,
EdgeContent, compact >::edge_t`

Weak pointer on an edge.

Definition at line 383 of file vvgraph.h.

17.81.3.13 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
__vvgraph_set<compact,vertex_t>::type graph::VVGraph<
VertexContent, EdgeContent, compact >::in_edges_t
[protected]`

Type of the list of sources of the in-edges.

Definition at line 503 of file vvgraph.h.

17.81.3.14 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
in_edges_t::const_iterator graph::VVGraph< VertexContent,
EdgeContent, compact >::ineighbor_iterator`

Iterator on the incoming neighbors of a vertex.

An incoming neighbor is a source vertex of an edge whose target is the current vertex.

Definition at line 628 of file vvgraph.h.

17.81.3.15 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
util::range<ineighbor_iterator> graph::VVGraph< VertexContent,
EdgeContent, compact >::ineighbor_iterator_range`

Range of the incoming neighbors of a vertex.

See also

[ineighbor_iterator](#)

Definition at line 635 of file vvgraph.h.

17.81.3.16 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
edge_list_t::const_iterator graph::VVGraph< VertexContent,
EdgeContent, compact >::int_const_neighbor_iterator
[protected]`

Constant iterator on the outgoing edges.

Definition at line 1722 of file vvgraph.h.

17.81.3.17 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
edge_list_t::iterator graph::VVGraph< VertexContent,
EdgeContent, compact >::int_neighbor_iterator [protected]`

Iterator on the outgoing edges.

Definition at line 1718 of file vvgraph.h.

17.81.3.18 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
util::SelectMemberIterator<typename neighborhood_t::iterator,
const vertex_t, &neighborhood_value_type::first>
graph::VVGraph< VertexContent, EdgeContent, compact
>::iterator`

Iterator on the vertexes.

Definition at line 594 of file vvgraph.h.

17.81.3.19 `template<typename VertexContent, typename
EdgeContent = _EmptyEdgeContent, bool compact =
false> typedef std::unordered_map<vertex_t, typename
neighborhood_t::iterator> graph::VVGraph< VertexContent,
EdgeContent, compact >::lookup_t`

Type of the fast-search map.

Definition at line 575 of file vvgraph.h.

17.81.3.20 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
search_result_t<single_neighborhood_t, int_neighbor_iterator>
graph::VVGraph< VertexContent, EdgeContent, compact
>::neighbor_found_t [protected]`

Result of a search in the neighborhood.

Definition at line 1727 of file vvgraph.h.

17.81.3.21 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
util::SelectMemberIterator<typename edge_list_t::iterator,
vertex_t, &neighbor_t::target> graph::VVGraph< VertexContent,
EdgeContent, compact >::neighbor_iterator`

Iterator on the neighbors of a vertex.

Definition at line 605 of file vvgraph.h.

17.81.3.22 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
util::range<neighbor_iterator> graph::VVGraph< VertexContent,
EdgeContent, compact >::neighbor_iterator_range`

Range of the neighbors of a vertex.

Definition at line 609 of file vvgraph.h.

17.81.3.23 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
std::list<std::pair<vertex_t, single_neighborhood_t> >
graph::VVGraph< VertexContent, EdgeContent, compact
>::neighborhood_t`

Type of the list of vertexes, together with the neighborhood.

Definition at line 570 of file vvgraph.h.

17.81.3.24 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
neighborhood_t::value_type graph::VVGraph< VertexContent,
EdgeContent, compact >::neighborhood_value_type`

Shortcut for the value_type of the neighborhood.

Definition at line 581 of file vvgraph.h.

17.81.3.25 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef
neighborhood_t::size_type graph::VVGraph< VertexContent,
EdgeContent, compact >::size_type`

Type of a size.

Definition at line 586 of file vvgraph.h.

17.81.3.26 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> typedef vertex_t
graph::VVGraph< VertexContent, EdgeContent, compact
>::value_type`

Type of the value of the graph as standard container.

Definition at line 400 of file vvgraph.h.

17.81.3.27 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> typedef Vertex<VertexContent> graph::VVGraph< VertexContent, EdgeContent, compact >::vertex_t`

Smart pointer on a vertex.

Definition at line 378 of file vvgraph.h.

17.81.4 Constructor & Destructor Documentation

17.81.4.1 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> graph::VVGraph< VertexContent, EdgeContent, compact >::VVGraph () [inline]`

Default constructor.

Creates an empty VV graph

Definition at line 407 of file vvgraph.h.

```
408         : graph_id(graph::getNextGraphId())
409         {}
```

17.81.4.2 `template<GRAPH_TEMPLATE > graph::VVGraph< GRAPH_TEMPLATE >::VVGraph (const VVGraph< VertexContent, EdgeContent, compact > ©) [inline]`

Copy constructor.

Maintains the caches in a valid state

Definition at line 2699 of file vvgraph.h.

```
2700     : graph_id(graph::getNextGraphId())
2701     {
2702         *this = copy;
2703     }
```

17.81.5 Member Function Documentation

17.81.5.1 `template<GRAPH_TEMPLATE > const VVGraph< GRAPH_ARGS >::vertex_t & graph::VVGraph< GRAPH_TEMPLATE >::any () const [inline]`

Return a vertex from the graph.

Returns

A vertex of the graph, or a null vertex if the graph is empty.

Example:

```
vvgraph S;  
// Add vertices in S  
const vertex& v = S.any(); // Get a vertex in S
```

Definition at line 1975 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::begin(), graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood, and graph::Vertex< VertexContent >::null.

```
1976     {  
1977         if(neighborhood.empty())  
1978             return vertex_t::null;  
1979         return *begin();  
1980     }
```

17.81.5.2 template<GRAPH_TEMPLATE > const VVGraph<
GRAPH_ARGS >::vertex_t & graph::VVGraph<
GRAPH_TEMPLATE >::anyIn (const vertex_t & v) const
[inline]

Return a vertex in the neighborhood of v.

Returns

A vertex in the neighborhood of v, or a null vertex if v is not in the graph or v has no neighborhood.

Example:

```
vvgraph S;  
forall(const vertex& v, S)  
{  
    const vertex& n = S.anyIn(v);  
    if(n)  
    {  
        edge e = S.edge(v,n); // e exist !  
    }  
}
```

Definition at line 1993 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex(), graph::Vertex< VertexContent >::isNull(), graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood, and graph::Vertex< VertexContent >::null.

```
1994     {  
1995         if(v.isNull())  
1996             return vertex_t::null;//vertex_t(0);  
1997         typename neighborhood_t::const_iterator it_found = this->findVertex(v);  
1998         if(it_found == neighborhood.end() || it_found->second.edges.empty())
```

```

1999         return vertex_t::null;//vertex_t(0);
2000         const edge_list_t& lst = it_found->second.edges;
2001         return lst.front().target;
2002     }

```

17.81.5.3 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS >::arc_t graph::VVGraph< GRAPH_TEMPLATE >::arc (const vertex_t &src, const vertex_t &tgt) [inline]`

Returns the arc (i.e.

undirected edge) between `src` and `tgt`.

When the arc is destroyed, the values of both half-edge are synchronized.

Example:

```

struct VertexContent();
struct EdgeContent {int a; }
typedef graph::VVGraph<VertexContent,EdgeContent> vvgraph;
typedef vvgraph::vertex_t vertex;
typedef vvgraph::arc_t arc;
typedef vvgraph::edge_t edge;
// ...

vvgraph S;
// initialize S
forall(const vertex& v, S)
{
    forall(const vertex& n, S.neighbors(v))
        if(v<n)
            S.arc(v,n)->a = 5
}

```

At the end, all edges have a value of 5 for `a`, even though we applied the function to only half the edges.

Definition at line 2383 of file `vvgraph.h`.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::Vertex< VertexContent >::id()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`.

```

2385     {
2386         neighbor_found_t found = this->findInVertex(src, tgt);
2387         if(!found)
2388             return arc_t();
2389         neighbor_found_t other = this->findInVertex(tgt, src);
2390         if(!other)
2391             return arc_t();
2392         return arc_t(src.id(), tgt.id(), &*found.it, &*other.it);
2393     }

```

17.81.5.4 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> const_iterator
graph::VVGraph< VertexContent, EdgeContent, compact >::begin
() const [inline]`

Returns an iterator on the beginning of the set of vertexes of the graph.

Definition at line 1558 of file vvgraph.h.

```
1558 { return neighborhood.begin(); }
```

17.81.5.5 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> iterator
graph::VVGraph< VertexContent, EdgeContent, compact >::begin
() [inline]`

Returns an iterator on the beginning of the set of vertexes of the graph.

Definition at line 1549 of file vvgraph.h.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact >::any().

```
1549 { return neighborhood.begin(); }
```

17.81.5.6 `template<GRAPH_TEMPLATE > void graph::VVGraph<
GRAPH_TEMPLATE >::clear (iterator it) [inline]`

Clear the outgoing edges of a vertex.

Definition at line 1910 of file vvgraph.h.

References util::SelectMemberIterator< Iterator, T, member, Reference, Pointer
>::base(), and graph::VVGraph< VertexContent, EdgeContent, compact
>::findVertex().

```
1911 {  
1912     const vertex_t& v = *it;  
1913     for(typename edge_list_t::iterator it_el = it.base()->second.edges.begin()  
1914     ;  
1915         it_el != it.base()->second.edges.end() ; ++it_el)  
1916     {  
1917         typename neighborhood_t::iterator it_found = this->findVertex(it_el->targ  
1918         et);  
1919         it_found->second.in_edges.erase(v);  
1920     }  
1921     it.base()->second.edges.clear();  
1922 }
```

17.81.5.7 `template<typename VertexContent, typename EdgeContent =
_EmptyEdgeContent, bool compact = false> void graph::VVGraph<
VertexContent, EdgeContent, compact >::clear () [inline]`

Clear the graph.

Definition at line 1474 of file vvgraph.h.

Referenced by `graph::VVGraph< VertexContent, EdgeContent, compact >::clear()`.

```
1474 { neighborhood.clear(); lookup.clear(); }
```

17.81.5.8 `template<GRAPH_TEMPLATE > bool graph::VVGraph<
GRAPH_TEMPLATE >::clear (const vertex_t & v) [inline]`

Clear the outgoing edges of a vertex.

Example:

```
vvgraph S;  
// Initialize S  
forall(const vertex& v, S)  
{  
    if(condition)  
    {  
        S.clear(v);  
        assert(S.empty(v));  
    }  
}
```

Definition at line 1898 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::clear()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::clear()`, and `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::clearOldGraphs()`.

```
1899 {  
1900     typename neighborhood_t::iterator it_found = this->findVertex(v);  
1901     if(it_found != neighborhood.end() )  
1902     {  
1903         clear(it_found);  
1904         return true;  
1905     }  
1906     return false;  
1907 }
```

17.81.5.9 template<GRAPH_TEMPLATE > bool graph::VVGraph<
GRAPH_TEMPLATE >::contains (const vertex_t & v) const
[inline]

Test if *v* is in the graph.

Returns

`true` is *v* is in the graph.

Example:

```
vvgraph S;  
vertex v;  
S.insert(v);  
assert(S.contains(v));
```

Definition at line 2253 of file `vvgraph.h`.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`,
and `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`.

Referenced by `graph::VVGraph< VertexContent, EdgeContent, compact >::subgraph()`.

```
2254        {  
2255            return this->findVertex(v) != neighborhood.end();  
2256        }
```

17.81.5.10 template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::size_type graph::VVGraph< GRAPH_TEMPLATE >::count
(const vertex_t & v) const [inline]

Count the number of vertexes *v* in the graph (can be 0 or 1 only).

Definition at line 1924 of file `vvgraph.h`.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::lookup`.

```
1925        {  
1926            return lookup.count(v);  
1927        }
```

17.81.5.11 template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::const_edge_t graph::VVGraph< GRAPH_TEMPLATE >::edge
(const vertex_t & src, const vertex_t & tgt) const [inline]

Returns the edge from *src* to *tgt*.

If the edge does not exists, returns a null edge.

Example:

```

struct VertexContent {};
struct EdgeContent { int a; }
typedef graph::VVGraph<VertexContent,EdgeContent> vvgraph;
typedef vvgraph::vertex_t vertex;
typedef vvgraph::edge_t edge;
typedef vvgraph::const_edge_t const_edge;
// ...

void fct(const vvgraph& S)
{
    forall(const vertex& v, S)
    {
        forall(const vertex& n, S.neighbors(v))
        {
            const_edge e = S.edge(v,n);
            cout << e->a << endl; // Here e->a cannot be modified
        }
    }
}

```

Definition at line 2397 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::Vertex< VertexContent >::id()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`.

```

2399     {
2400         const_neighbor_found_t found = this->findInVertex(src, tgt);
2401         if(!found)
2402             return const_edge_t();
2403         return const_edge_t(src.id(), tgt.id(), &*found.it);
2404     }

```

17.81.5.12 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS >::edge_t graph::VVGraph< GRAPH_TEMPLATE >::edge (const vertex_t & src, const vertex_t & tgt) [inline]`

Returns the edge from `src` to `tgt`.

If the edge does not exists, returns a null edge.

Example:

```

struct VertexContent {};
struct EdgeContent { int a; }
typedef graph::VVGraph<VertexContent,EdgeContent> vvgraph;
typedef vvgraph::vertex_t vertex;
typedef vvgraph::edge_t edge;
// ...

vvgraph S;
// initialize S
forall(const vertex& v, S)
{
    forall(const vertex& n, S.neighbors(v))
    {
        edge e = S.edge(v,n);
    }
}

```

```
    e->a = 10;
  }
}
```

Definition at line 2372 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex(), graph::Vertex< VertexContent >::id(), and graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it.

Referenced by algorithms::split().

```
2374     {
2375         neighbor_found_t found = this->findInVertex(src, tgt);
2376         if(!found)
2377             return edge_t();
2378         return edge_t(src.id(), tgt.id(), &*found.it);
2379     }
```

17.81.5.13 template<GRAPH_TEMPLATE > bool graph::VVGraph< GRAPH_TEMPLATE >::empty (const vertex_t & v) const [inline]

Test if a vertex has a neighbor.

Returns false if it does. If a vertex is not in the graph, it is considered as having no neighbor. In the same way, the null vertex has no neighbors.

Example:

```
vvgraph S;
// initialize S
forall(cnst vertex& v, S)
{
    if(!S.empty(v))
    {
        const vertex& n = S.anyIn(v);
        // Working with n a neighbor of v
    }
}
```

Definition at line 2017 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex(), graph::Vertex< VertexContent >::isNull(), and graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood.

```
2018     {
2019         if(v.isNull())
2020             return true;
2021         typename neighborhood_t::const_iterator it_found = this->findVertex(v);
2022         if(it_found == neighborhood.end())
2023             return true;
2024         return it_found->second.edges.empty();
2025     }
```

17.81.5.14 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> bool
graph::VVGraph< VertexContent, EdgeContent, compact >::empty
() const [inline]`

Test if there is any vertex in the graph.

Definition at line 796 of file vvgraph.h.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_
TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_
COMPLEX_TEMPLATE_ARGS >)>::border()`.

```
796 { return neighborhood.empty(); }
```

17.81.5.15 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> const_iterator
graph::VVGraph< VertexContent, EdgeContent, compact >::end ()
const [inline]`

Returns an iterator on the end of the set of vertexes of the graph.

Definition at line 1562 of file vvgraph.h.

```
1562 { return neighborhood.end(); }
```

17.81.5.16 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> iterator
graph::VVGraph< VertexContent, EdgeContent, compact >::end ()
[inline]`

Returns an iterator on the end of the set of vertexes of the graph.

Definition at line 1553 of file vvgraph.h.

```
1553 { return neighborhood.end(); }
```

17.81.5.17 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::iterator graph::VVGraph< GRAPH_TEMPLATE >::erase
(iterator pos) [inline]`

Erase element at position `pos`.

And return the iterator on the next element (as required for sequence containers in C++)

Definition at line 1817 of file vvgraph.h.

References `util::SelectMemberIterator< Iterator, T, member, Reference,
Pointer >::base()`, `graph::VVGraph< VertexContent, EdgeContent, compact`

>::eraseAllEdges(), graph::VVGraph< VertexContent, EdgeContent, compact >::lookup, and graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood.

```
1818     {
1819         eraseAllEdges(it);
1820         lookup.erase(*it);
1821         return neighborhood.erase(it.base());
1822     }
```

**17.81.5.18 template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::size_type graph::VVGraph< GRAPH_TEMPLATE >::erase
(const vertex_t & v) [inline]**

Remove a vertex from the graph.

Parameters

v vertex to erase

Returns

1 if the vertex was erased, 0 else.

Example:

```
vvgraph S;
// Add vertices to S
vertex v = S.any();
S.erase(v); // Remove a vertex of S
```

Definition at line 1826 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::eraseAllEdges(), graph::VVGraph< VertexContent, EdgeContent, compact >::lookup, and graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood.

Referenced by vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteCellInGraphs(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteJunction(), and vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::mergeCells().

```
1827     {
1828         if (eraseAllEdges(v))
1829         {
1830             typename lookup_t::iterator it_found = lookup.find(v);
1831             if (it_found == lookup.end()) return 0;
1832             typename neighborhood_t::iterator it = it_found->second;
```

```

1833         neighborhood.erase(it);
1834         lookup.erase(v);
1835         return 1;
1836     }
1837     return 0;
1838 }

```

17.81.5.19 `template<GRAPH_TEMPLATE> void graph::VVGraph<GRAPH_TEMPLATE>::eraseAllEdges(iterator it) [inline]`

Clear the neighborhood of a vertex.

All edges, to and from **it* will be erased.

Definition at line 1864 of file `vvgraph.h`.

References `util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::base()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`, `graph::VVGraph< VertexContent, EdgeContent, compact >::lookup`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`.

```

1865     {
1866         const vertex_t& v = *it_found;
1867         for(typename edge_list_t::iterator it_el = it_found.base()->second.edges.begin() ;
1868             it_el != it_found.base()->second.edges.end() ; ++it_el)
1869         {
1870             typename lookup_t::iterator it_found = lookup.find(it_el->target);
1871             if(it_found != lookup.end())
1872                 it_found->second->second.in_edges.erase(v);
1873         }
1874         it_found.base()->second.edges.clear();
1875         it_found.base()->second.flagged = 0;
1876         // Remove the incoming edges
1877         in_edges_t &in_edges = it_found.base()->second.in_edges;
1878         for( typename in_edges_t::iterator it = in_edges.begin() ;
1879             it != in_edges.end() ; ++it )
1880         {
1881             #ifndef VVGRAPH_NO_EDGE_CACHE
1882                 EdgeCache *cache = reinterpret_cast<EdgeCache*>(it->cache);
1883                 if(!cache or !cache->eraseEdge())
1884                 {
1885                     #endif
1886                         neighbor_found_t found = findInVertex(*it, v);
1887                         found.neighborhood->edges.erase(found.it);
1888                         if(found.neighborhood->flagged && *found.neighborhood->flagged == v) found.neighborhood->flagged = 0;
1889                     #ifndef VVGRAPH_NO_EDGE_CACHE
1890                 }
1891             #endif
1892         }
1893         in_edges.clear();
1894     }

```

17.81.5.20 template<GRAPH_TEMPLATE > void graph::VVGraph<
GRAPH_TEMPLATE >::eraseAllEdges () [inline]

Clear all the edges in the graph.

Example:

```
vvgraph S;  
// Initialize S  
vvgraph::size_type s1 = S.size(); // Save the number of vertices  
S.eraseAllEdges();  
assert(s1 == S.size()); // The number of vertices didn't change  
forall(const vertex& v, S)  
{  
    assert(S.empty(v)); // No neighbor  
}
```

Definition at line 1840 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact
>::neighborhood.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact >::erase(),
and graph::VVGraph< VertexContent, EdgeContent, compact >::eraseAllEdges().

```
1841        {  
1842            for(typename neighborhood_t::iterator it = neighborhood.begin();  
1843                it != neighborhood.end() ; ++it)  
1844            {  
1845                it->second.edges.clear();  
1846                it->second.in_edges.clear();  
1847                it->second.flagged = 0;  
1848            }  
1849        }
```

17.81.5.21 template<GRAPH_TEMPLATE > bool graph::VVGraph<
GRAPH_TEMPLATE >::eraseAllEdges (const vertex_t & v)
[inline]

Clear the neighborhood of a vertex.

All edges, to and from v will be erased.

Example:

```
vvgraph S;  
// Initialize S  
forall(const vertex& v, S)  
{  
    if(condition)  
    {  
        S.eraseAllEdges(v);  
        assert(S.empty(v)); // No outgoing edges  
        assert(S.iEmpty(v)); // No incoming edges  
    }  
}
```

Definition at line 1852 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::eraseAllEdges()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`.

```

1853     {
1854         typename neighborhood_t::iterator it_found = this->findVertex(v);
1855         if(it_found != neighborhood.end() )
1856         {
1857             eraseAllEdges(it_found);
1858             return true;
1859         }
1860         return false;
1861     }

```

17.81.5.22 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS >::circ_neighbor_iterator graph::VVGraph< GRAPH_TEMPLATE >::eraseEdge (const vertex_t & v, circ_neighbor_iterator pos) [inline]`

Erase the edge pointed to by the iterator.

Definition at line 2229 of file vvgraph.h.

References `util::CircIterator< ForwardIterator >::end()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::eraseEdge()`, and `util::CircIterator< ForwardIterator >::isInitIterator()`.

```

2230     {
2231         neighbor_iterator nit = eraseEdge(v, pos.base());
2232         if(pos.isInitIterator(nit))
2233             return pos.end();
2234         return circ_neighbor_iterator(pos, nit);
2235     }

```

17.81.5.23 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS >::neighbor_iterator graph::VVGraph< GRAPH_TEMPLATE >::eraseEdge (const vertex_t & v, neighbor_iterator pos) [inline]`

Erase the edge pointed to by the iterator.

Definition at line 2209 of file vvgraph.h.

References `util::SelectMemberIterator< Iterator, T, member, Reference, Pointer >::base()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`.

```

2210     {
2211         typename neighborhood_t::iterator it_found = this->findVertex(v);

```

```

2212         if(it_found == neighborhood.end())
2213             return neighbor_iterator();
2214         const vertex_t& n = *pos;
2215         typename neighborhood_t::iterator it_found2 = this->findVertex(n);
2216         if(it_found2 == neighborhood.end())
2217             return neighbor_iterator();
2218         size_type result = it_found2->second.in_edges.erase(v);
2219         if(result)
2220         {
2221             typename edge_list_t::iterator it = pos.base();
2222             return it_found->second.edges.erase(it);
2223         }
2224         return neighbor_iterator();
2225     }

```

17.81.5.24 template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::size_type graph::VVGraph< GRAPH_TEMPLATE
>::eraseEdge (const vertex_t & src, const vertex_t & tgt)
[inline]

Erase the edge from src to tgt.

Returns

True if successful.

Example:

```

vvgraph S;
// Initialize S
forall(const vertex& v, S)
{
    forall(const vertex& n, S.neighbors(v))
    {
        if(condition)
        {
            S.eraseEdge(v,n);
            break; // The iteration must not continue after the neighborhood has been
                altered
        }
    }
}

```

Definition at line 2239 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact
>::findInVertex(), graph::VVGraph< VertexContent, EdgeContent, compact
>::search_result_t< Neighborhood, Iterator >::it, graph::VVGraph< Vertex-
Content, EdgeContent, compact >::lookup, and graph::VVGraph< VertexContent,
EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact
>::eraseEdge(), and algorithms::split().

```

2240     {

```

```

2241     neighbor_found_t found = this->findInVertex(v, n);
2242     if(!found)
2243         return 0;
2244     if(found.neighborhood->flagged && *found.neighborhood->flagged == n) found.
neighborhood->flagged = 0;
2245     found.neighborhood->edges.erase(found.it);
2246     typename lookup_t::iterator it_found = lookup.find(n);
2247     if(it_found != lookup.end())
2248         it_found->second->second.in_edges.erase(v);
2249     return 1;
2250 }

```

17.81.5.25 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> const_iterator graph::VVGraph< VertexContent, EdgeContent, compact >::find (const vertex_t & v) const [inline]`

Find the vertex *v* in the graph.

Returns

An iterator on *v* if found, or [end\(\)](#)

Definition at line 1615 of file `vvgraph.h`.

```
1615 { return this->findVertex(v); }
```

17.81.5.26 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> iterator graph::VVGraph< VertexContent, EdgeContent, compact >::find (const vertex_t & v) [inline]`

Find the vertex *v* in the graph.

Returns

An iterator on *v* if found, or [end\(\)](#)

Definition at line 1609 of file `vvgraph.h`.

```
1609 { return this->findVertex(v); }
```

17.81.5.27 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS >::const_neighbor_iterator graph::VVGraph< GRAPH_TEMPLATE >::findIn (const vertex_t & v, const vertex_t & n) const [inline]`

Find the vertex *n* in the neighborhood of *v*.

Returns

- An iterator on *n* in the neighborhood of *v*, if found
- An iterator on the end of the neighborhood if *n* is not found in *v*
- Otherwise, the result is undefined

Definition at line 2199 of file `vvgraph.h`.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`.

```
2200     {
2201         const_neighbor_found_t found = this->findInVertex(v, n);
2202         if(found.neighborhood)
2203             return const_neighbor_iterator(found.it);
2204         return const_neighbor_iterator(found.it);
2205     }
```

17.81.5.28 template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::neighbor_iterator graph::VVGraph< GRAPH_TEMPLATE
>::findIn (const vertex_t & v, const vertex_t & n) [inline]

Find the vertex *n* in the neighborhood of *v*.

Returns

- An iterator on *n* in the neighborhood of *v*, if found
- An iterator on the end of the neighborhood if *n* is not found in *v*
- Otherwise, the result is undefined

Definition at line 2189 of file `vvgraph.h`.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`.

```
2190     {
2191         neighbor_found_t found = this->findInVertex(v, n);
2192         if(found.neighborhood)
2193             return neighbor_iterator(found.it);
2194         return neighbor_iterator();
2195     }
```

17.81.5.29 `template<GRAPH_TEMPLATE > VVGraph<
GRAPH_ARGS >::const_neighbor_found_t graph::VVGraph<
GRAPH_TEMPLATE >::findInVertex (const vertex_t & v, const
vertex_t & neighbor) const [inline, protected]`

Constant version of [findInVertex\(const vertex_t&, const vertex_t&\)](#).

Definition at line 2149 of file `vvgraph.h`.

References `graph::Vertex< VertexContent >::cache`, `graph::Vertex< VertexContent >::cache_source`, `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id`, `graph::Vertex< VertexContent >::id()`, `graph::Vertex< VertexContent >::isNull()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`.

```

2150     {
2151         if(v.isNull() || n.isNull() || v == n)
2152             return const_neighbor_found_t();
2153 #ifndef VVGRAPH_NO_EDGE_CACHE
2154         if(n.cache_source == v.id())
2155         {
2156             EdgeCache *cache = reinterpret_cast<EdgeCache*>(n.cache);
2157             if(cache->graph_id() == graph_id)
2158             {
2159                 return const_neighbor_found_t(cache->it(), cache->neighborhood());
2160             }
2161         }
2162         else if(v.cache_source == v.id())
2163         {
2164             EdgeCache *cache = reinterpret_cast<EdgeCache*>(v.cache);
2165             if(cache->graph_id() == graph_id and cache->it()->target == n)
2166             {
2167                 return const_neighbor_found_t(cache->it(), cache->neighborhood());
2168             }
2169         }
2170 #endif // VVGRAPH_NO_EDGE_CACHE
2171         typename neighborhood_t::const_iterator it_found = this->findVertex(v);
2172         if(it_found == neighborhood.end())
2173             return const_neighbor_found_t();
2174         const edge_list_t &lst = it_found->second.edges;
2175         const single_neighborhood_t &neighborhood = it_found->second;
2176         for(typename edge_list_t::const_iterator it = lst.begin() ;
2177             it != lst.end() ; ++it)
2178         {
2179             if(it->target == n)
2180             {
2181                 return const_neighbor_found_t(it, &neighborhood);
2182             }
2183         }
2184         return const_neighbor_found_t(lst.end(), &neighborhood, false);
2185     }

```


17.81.5.30 `template<GRAPH_TEMPLATE > VVGrahp< GRAPH_ARGS
>::neighbor_found_t graph::VVGrahp< GRAPH_TEMPLATE
>::findInVertex (const vertex_t & v, const vertex_t & neighbor)
[inline, protected]`

Find a vertex neighbor in the neighborhood of v.

If 'v' is in the graph and 'neighbor' is in its neighborhood, the result is convertible to true, 'neighborhood' points toward the neighborhood structure and 'it' points toward the neighbor.

If 'v' is in the graph, but 'neighbor' is not in the neighborhood, the result is convertible to false, 'neighborhood' points toward the neighborhood structure and 'it' is neighborhood->edges.end().

If v is not in the graph, the result is convertible to false and the address of the stored neighborhood is 0.

Definition at line 2109 of file vvgraph.h.

References graph::Vertex< VertexContent >::cache, graph::Vertex< VertexContent >::cache_source, graph::VVGrahp< VertexContent, EdgeContent, compact >::findVertex(), graph::VVGrahp< VertexContent, EdgeContent, compact >::graph_id, graph::Vertex< VertexContent >::id(), graph::Vertex< VertexContent >::isNull(), and graph::VVGrahp< VertexContent, EdgeContent, compact >::neighborhood.

Referenced by graph::VVGrahp< VertexContent, EdgeContent, compact >::arc(), graph::VVGrahp< VertexContent, EdgeContent, compact >::edge(), graph::VVGrahp< VertexContent, EdgeContent, compact >::eraseAllEdges(), graph::VVGrahp< VertexContent, EdgeContent, compact >::eraseEdge(), graph::VVGrahp< VertexContent, EdgeContent, compact >::findIn(), graph::VVGrahp< VertexContent, EdgeContent, compact >::flag(), graph::VVGrahp< VertexContent, EdgeContent, compact >::neighbors(), graph::VVGrahp< VertexContent, EdgeContent, compact >::nextTo(), graph::VVGrahp< VertexContent, EdgeContent, compact >::prevTo(), graph::VVGrahp< VertexContent, EdgeContent, compact >::replace(), graph::VVGrahp< VertexContent, EdgeContent, compact >::spliceAfter(), graph::VVGrahp< VertexContent, EdgeContent, compact >::spliceBefore(), and graph::VVGrahp< VertexContent, EdgeContent, compact >::subgraph().

```

2110     {
2111         if(v.isNull() || n.isNull() || v == n)
2112             return neighbor_found_t();
2113 #ifndef VVGrahp_NO_EDGE_CACHE
2114         if(n.cache_source == v.id())
2115         {
2116             EdgeCache* cache = reinterpret_cast<EdgeCache*>(n.cache);
2117             if(cache->graph_id() == graph_id)
2118             {
2119                 return neighbor_found_t(cache->it(), cache->neighborhood());
2120             }
2121         }
2122         else if(v.cache_source == v.id())
2123         {
2124             EdgeCache* cache = reinterpret_cast<EdgeCache*>(v.cache);
2125             if(cache->graph_id() == graph_id and cache->it()->target == n)

```

```

2126         {
2127             return neighbor_found_t(cache->it(), cache->neighborhood());
2128         }
2129     }
2130 #endif // VVGRAPH_NO_EDGE_CACHE
2131     typename neighborhood_t::iterator it_found = this->findVertex(v);
2132     if(it_found == neighborhood.end())
2133         return neighbor_found_t();
2134     edge_list_t &lst = it_found->second.edges;
2135     single_neighborhood_t &neighborhood = it_found->second;
2136     for(typename edge_list_t::iterator it = lst.begin() ;
2137         it != lst.end() ; ++it)
2138     {
2139         if(it->target == n)
2140         {
2141             return neighbor_found_t(it, &neighborhood);
2142         }
2143     }
2144     return neighbor_found_t(lst.end(), &neighborhood, false);
2145 }

```

17.81.5.31 **template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::neighborhood_t::const_iterator graph::VVGraph<
GRAPH_TEMPLATE >::findVertex (const vertex_t & v) const
[inline, protected]**

Constant version of [findVertex\(const vertex_t&\)](#).

Definition at line 2088 of file vvgraph.h.

References [graph::Vertex< VertexContent >::cache](#), [graph::Vertex< VertexContent >::cache_source](#), [graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id](#), [graph::VVGraph< VertexContent, EdgeContent, compact >::lookup](#), and [graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood](#).

```

2089     {
2090 #ifndef VVGRAPH_NO_VERTEX_CACHE
2091         if(v.cache)
2092         {
2093             if(v.cache_source == 0)
2094             {
2095                 VertexCache* cache = reinterpret_cast<VertexCache*>(v.cache);
2096                 if(cache->graph_id() == graph_id)
2097                     return cache->neighborhood();
2098             }
2099         }
2100 #endif // VVGRAPH_NO_VERTEX_CACHE
2101     typename lookup_t::const_iterator it_found = lookup.find(v);
2102     if(it_found != lookup.end())
2103         return it_found->second;
2104     return neighborhood.end();
2105 }

```

```

17.81.5.32 template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::neighborhood_t::iterator graph::VVGraph<
GRAPH_TEMPLATE >::findVertex (const vertex_t & v)
[inline, protected]

```

Find a vertex in the graph and returns the iterator on it.

Goal is to introduce a new optimization. When iterating over the vertices of the graph, they will cache their own iterator. So accessing their neighborhood would be constant time.

Definition at line 2065 of file vvgraph.h.

References graph::Vertex< VertexContent >::cache, graph::Vertex< VertexContent >::cache_source, graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id, graph::VVGraph< VertexContent, EdgeContent, compact >::lookup, and graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact >::anyIn(), graph::VVGraph< VertexContent, EdgeContent, compact >::clear(), graph::VVGraph< VertexContent, EdgeContent, compact >::contains(), graph::VVGraph< VertexContent, EdgeContent, compact >::empty(), graph::VVGraph< VertexContent, EdgeContent, compact >::eraseAllEdges(), graph::VVGraph< VertexContent, EdgeContent, compact >::eraseEdge(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::find(), graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex(), graph::VVGraph< VertexContent, EdgeContent, compact >::flagged(), graph::VVGraph< VertexContent, EdgeContent, compact >::iAnyIn(), graph::VVGraph< VertexContent, EdgeContent, compact >::iEmpty(), graph::VVGraph< VertexContent, EdgeContent, compact >::iNeighbors(), graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdge(), graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdges(), graph::VVGraph< VertexContent, EdgeContent, compact >::iValence(), graph::VVGraph< VertexContent, EdgeContent, compact >::neighbors(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::reference(), graph::VVGraph< VertexContent, EdgeContent, compact >::replace(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::source(), graph::VVGraph< VertexContent, EdgeContent, compact >::subgraph(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::target(), and graph::VVGraph< VertexContent, EdgeContent, compact >::valence().

```

2066     {
2067 #ifndef VVGRAPH_NO_VERTEX_CACHE
2068     if (v.cache)
2069     {
2070         if (v.cache_source == 0)
2071         {
2072             VertexCache* cache = reinterpret_cast<VertexCache*>(v.cache);
2073             if (cache->graph_id() == graph_id)
2074             {

```

```

2075         return cache->neighborhood();
2076     }
2077 }
2078 }
2079 #endif // VVGRAPH_NO_VERTEX_CACHE
2080 typename lookup_t::const_iterator it_found = lookup.find(v);
2081 if(it_found != lookup.end())
2082     return it_found->second;
2083 return neighborhood.end();
2084 }

```

17.81.5.33 `template<GRAPH_TEMPLATE > bool graph::VVGraph<GRAPH_TEMPLATE >::flag (const vertex_t & src, const vertex_t & neighbor) [inline]`

Flag a neighbor for quick retrieval.

Returns

true if the vertex has been correctly flagged

Example:

```

vvgraph S;
// Initialize S
forall(const vertex& v, S)
{
    forall(const vertex& n, S.neighbors(v))
    {
        if(condition(v,n))
        {
            S.flag(v,n);
            break;
        }
    }
}
// ...
forall(const vertex& v, S)
{
    const vertex& n = S.flagged(v);
    if(n) // If a neighbor has been flagged
    {
        // Work with the flagged neighbor
    }
}

```

Definition at line 2407 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`.

```

2409     {

```

```

2410     neighbor_found_t found = this->findInVertex(src, neighbor);
2411     if(!found)
2412         return false;
2413     found.neighborhood->flagged = &(found.it->target);
2414     return true;
2415 }
```

17.81.5.34 `template<GRAPH_TEMPLATE > const VVGraph< GRAPH_ARGS >::vertex_t & graph::VVGraph< GRAPH_TEMPLATE >::flagged (const vertex_t & src) const`
[inline]

Return the flagged neighbor or a null vertex is no neighbor have been flagged.

See also

[VVGraph::flag\(const vertex_t&, const vertex_t&\)](#)

Definition at line 2419 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`, and `graph::Vertex< VertexContent >::null`.

```

2420     {
2421         typename neighborhood_t::const_iterator found = this->findVertex(src);
2422         if(found == neighborhood.end() or found->second.flagged == 0)
2423             return vertex_t::null;
2424         return *found->second.flagged;
2425     }
```

17.81.5.35 `template<GRAPH_TEMPLATE > const VVGraph< GRAPH_ARGS >::vertex_t & graph::VVGraph< GRAPH_TEMPLATE >::iAnyIn (const vertex_t & v) const`
[inline]

Returns any incoming neighbor of v.

Returns

A vertex in the incoming neighborhood of v, or a null vertex if v is not in the graph or v has no incoming neighbor.

Example:

```

vvgraph S;
// initialize S
forall(const vertex& n, S)
{
    const vertex& v = S.iAnyIn(v);
    if(v)
```

```

    {
        edge e = S.edge(v, n);
        // Work with e the edge from v to n
    }
}

```

Definition at line 2029 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, `graph::Vertex< VertexContent >::isNull()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`, and `graph::Vertex< VertexContent >::null`.

```

2030     {
2031         if(v.isNull())
2032             return vertex_t::null;//vertex_t(0);
2033         typename neighborhood_t::const_iterator it_found = this->findVertex(v);
2034         if(it_found == neighborhood.end() || it_found->second.in_edges.empty())
2035             return vertex_t::null;//vertex_t(0);
2036         const in_edges_t& lst = it_found->second.in_edges;
2037         return *lst.begin();
2038     }

```

17.81.5.36 `template<GRAPH_TEMPLATE > bool graph::VVGraph< GRAPH_TEMPLATE >::iEmpty (const vertex_t & v) const [inline]`

Test if a vertex has an incoming neighbor.

Returns false if it does. If a vertex is not in the graph, it is considered as having no incoming neighbor. In the same way, the null vertex has no incoming neighbors.

Example:

```

vvgraph S;
// initialize S
forall(const vertex& n, S)
{
    if(!S.iEmpty(n))
    {
        const vertex& v = S.iAnyIn(n);
        edge e = S.edge(v,n); // It exists !
    }
}

```

Definition at line 2053 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, `graph::Vertex< VertexContent >::isNull()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`.

```

2054     {
2055         if(v.isNull())
2056             return true;
2057         typename neighborhood_t::const_iterator it_found = this->findVertex(v);
2058         if(it_found == neighborhood.end())
2059             return true;
2060         return it_found->second.in_edges.empty();
2061     }

```

17.81.5.37 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::ineighbor_iterator_range graph::VVGraph<
GRAPH_TEMPLATE >::iNeighbors (const vertex_t & v) const
[inline]`

Return the range of incoming neighbors of v.

Iterating over iNeighbors will go through all the vertexes having an edge toward v.

Example:

```
vvgraph S;
// initialize S
forall(const vertex& n, S)
{
    forall(const vertex& v, S.iNeighbors(v))
    {
        // n is a neighbor of v
    }
}
```

Definition at line 2636 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex(), graph::Vertex< VertexContent >::isNull(), util::make_range(), and graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood.

```
2637     {
2638         ineighbor_iterator_range result;
2639         if(v.isNull())
2640             return result;
2641         typename neighborhood_t::const_iterator it_found = this->findVertex(v);
2642         if(it_found == neighborhood.end())
2643             return result;
2644         const in_edges_t &lst = it_found->second.in_edges;
2645         result = util::make_range(ineighbor_iterator(lst.begin()),
2646                                 ineighbor_iterator(lst.end()));
2647         return result;
2648     }
```

17.81.5.38 `template<GRAPH_TEMPLATE > template<typename Iterator >
void graph::VVGraph< GRAPH_TEMPLATE >::insert (Iterator
first, Iterator last) [inline]`

Insert all the vertexes from first to last-1 in the graph.

Definition at line 1965 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::insert().

```
1966     {
1967         for(Iterator it = first ; it != last ; ++it)
1968         {
1969             insert(*it);
1970         }
1971     }
```

17.81.5.39 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS >::iterator graph::VVGraph< GRAPH_TEMPLATE >::insert(iterator pos, const vertex_t & v) [inline]`

Insert a new vertex in the graph.

The iterator is ignored ...

Definition at line 1958 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::insert()`.

```
1959     {
1960         return this->insert(v);
1961     }
```

17.81.5.40 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS >::iterator graph::VVGraph< GRAPH_TEMPLATE >::insert(const vertex_t & v) [inline]`

Insert a new vertex in the graph.

Returns

An iterator on the inserted vertex.

Example:

```
vvgraph S;
vertex v; // Create a vertex
S.insert(v); // Insert it in the graph
```

Definition at line 1931 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::lookup`, `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`, `util::tie()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::updateVertexCache()`.

Referenced by `graph::VVGraph< VertexContent, EdgeContent, compact >::insert()`, `algorithms::split()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::subgraph()`.

```
1932     {
1933         typename lookup_t::iterator it_found;
1934         bool inserted;
1935         util::tie(it_found, inserted) = lookup.insert(std::make_pair(v, typename neighborhood_t::iterator()));
1936         if(inserted)
1937         {
1938             neighborhood.push_front(std::make_pair(v, single_neighborhood_t()));
1939             typename neighborhood_t::iterator it = neighborhood.begin();
1940             it_found->second = it;
1941             updateVertexCache(it->first, it);
1942         }
```



```

1942         return it;
1943     }
1944     return it_found->second;
1945 }
```

17.81.5.41 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::edge_t graph::VVGraph< GRAPH_TEMPLATE >::insertEdge
(const vertex_t & src, const vertex_t & tgt) [inline]`

Insert a new edge in the graph, without ordering.

If `new_neighbor` is already in the neighborhood of `v`, the insertion fails.

Returns

The just created edge if everything succeed, or a null edge.

Example:

```

vvgraph S;
vertex v1, v2;
S.insert(v1);
S.insert(v2);
S.insertEdge(v1, v2); // Insert the edge between v1 and v2
S.insertEdge(v2, v1); // Insert the edge between v2 and v1
```

Definition at line 2550 of file `vvgraph.h`.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`,
`graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id`,
`graph::Vertex< VertexContent >::id()`, `graph::VVGraph< VertexContent, Edge-`
`Content, compact >::insertInEdge()`, `graph::Vertex< VertexContent >::isNull()`,
`graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`, `util::tie()`,
and `graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache()`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_-`
`TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_-`
`COMPLEX_TEMPLATE_ARGS >)>::connectFromJunctions()`, and `algo-`
`rithms::split()`.

```

2552     {
2553         if(v.isNull() || new_neighbor.isNull() || v == new_neighbor )
2554             return edge_t();
2555         typename neighborhood_t::iterator it_found = this->findVertex(v);
2556         if(it_found == neighborhood.end())
2557             return edge_t();
2558         ineighbor_iterator it_in;
2559         bool inserted;
2560         util::tie(it_in, inserted) = insertInEdge(v, new_neighbor);
2561         if(!inserted)
2562             return edge_t();
2563         edge_list_t &lst = it_found->second.edges;
2564         lst.push_front(neighbor_t(new_neighbor, EdgeContent(), it_found->second,
graph_id));
2565         typename edge_list_t::iterator it = lst.begin();
```

```

2566         updateEdgeCache(v, it, *it_in);
2567         return edge_t(v.id(), new_neighbor.id(), &*it);
2568     }

```

17.81.5.42 `template<GRAPH_TEMPLATE > template<typename VertexContainer > std::vector< typename VVGraph< GRAPH_ARGS >::edge_t > graph::VVGraph< GRAPH_TEMPLATE >::insertEdges(const vertex_t & src, const VertexContainer & targets) [inline]`

Insert a set of edges, keeping the relative order of the list of edges.

Definition at line 2525 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id`, `graph::Vertex< VertexContent >::id()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insertInEdge()`, `graph::Vertex< VertexContent >::isNull()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`, `util::tie()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache()`.

```

2526     {
2527         std::vector<edge_t> result;
2528         typename neighborhood_t::iterator it_found = this->findVertex(v);
2529         if(it_found == neighborhood.end()) return result;
2530         ineighbor_iterator it_in;
2531         bool inserted;
2532         edge_list_t& lst = it_found->second.edges;
2533         typename edge_list_t::iterator lst_it = lst.begin();
2534         result.reserve(new_neighbors.size());
2535         for(typename VertexContainer::const_iterator it = new_neighbors.begin() ;
2536             it != new_neighbors.end() ; ++it) {
2537             const vertex_t& new_neighbor = *it;
2538             if(new_neighbor.isNull() or new_neighbor == v) return result;
2539             util::tie(it_in, inserted) = this->insertInEdge(v, new_neighbor);
2540             if(!inserted) return result;
2541             typename edge_list_t::iterator new_lst_it = lst.insert(lst_it, neighbor_t
(new_neighbor, EdgeContent(), it_found->second, graph_id));
2542             updateEdgeCache(v, new_lst_it, *it_in);
2543             result.push_back(edge_t(v.id(), new_neighbor.id(), &*new_lst_it));
2544         }
2545         return result;
2546     }

```

17.81.5.43 `template<GRAPH_TEMPLATE > std::pair< typename VVGraph< GRAPH_ARGS >::ineighbor_iterator, bool > graph::VVGraph< GRAPH_TEMPLATE >::insertInEdge(const vertex_t & v, const vertex_t & new_neighbor) [inline, protected]`

Insert v in the in_edges of new_neighbor and return true if the insertion was actually done, or false if v was already there.

Definition at line 2260 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::lookup.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdge(), graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdges(), graph::VVGraph< VertexContent, EdgeContent, compact >::replace(), graph::VVGraph< VertexContent, EdgeContent, compact >::spliceAfter(), graph::VVGraph< VertexContent, EdgeContent, compact >::spliceBefore(), and graph::VVGraph< VertexContent, EdgeContent, compact >::subgraph().

```

2261     {
2262         typename lookup_t::iterator found = lookup.find(new_neighbor);
2263         if(found == lookup.end())
2264             return std::make_pair(ineighbor_iterator(), false);
2265         //typename neighborhood_t::iterator found = neighborhood.find(new_neighbor)
2266     ;
2267     //if(found == neighborhood.end())
2268     //return std::make_pair(ineighbor_iterator(), false);
2269     return found->second->second.in_edges.insert(v);
2270 }
```

17.81.5.44 template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS >::size_type graph::VVGraph< GRAPH_TEMPLATE >::iValence (const vertex_t & v) const [inline]

Returns the number of incoming neighbors of v.

The incoming neighbors are the set of vertexes with on edge ending on v.

Example:

```

vvgraph S;
// initialize S
forall(const vertex& n, S)
{
    size_t nb_inneighbors = S.iValence(n);
    // nb_inneighbors is the number of vertices source of an edge toward n
}
```

Definition at line 2042 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex(), graph::Vertex< VertexContent >::isNull(), and graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood.

```

2043     {
2044         if(v.isNull())
2045             return 0;
2046         typename neighborhood_t::const_iterator it_found = this->findVertex(v);
2047         if(it_found == neighborhood.end())
2048             return 0;
2049         return it_found->second.in_edges.size();
2050     }
```

17.81.5.45 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::const_circ_neighbor_iterator_range graph::VVGraph<
GRAPH_TEMPLATE >::neighbors (const vertex_t & v, const
vertex_t & n) const [inline]`

Return the range of neighbors of v, starting at n.

Example:

```
vvgraph S;
// initialize S
forall(const vertex& m, S.neighbors(v,n))
{
    // m is a neighbor of v
    // on the first loop, m == n
}
```

Definition at line 2604 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::Vertex< VertexContent >::isNull()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`, `util::make_range()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`.

```
2605     {
2606         const_circ_neighbor_iterator_range result;
2607         if(v.isNull())
2608             return result;
2609         const_neighbor_found_t found = this->findInVertex(v, n);
2610         if(!found)
2611             return result;
2612         const edge_list_t &lst = found.neighborhood->edges;
2613         result = util::make_range(const_circ_neighbor_iterator(lst.begin(), lst.end
2614         (), found.it),
2615                                 const_circ_neighbor_iterator(lst.begin(), lst.end
2616         ())),
2615         return result;
2616     }
```

17.81.5.46 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::circ_neighbor_iterator_range graph::VVGraph<
GRAPH_TEMPLATE >::neighbors (const vertex_t & v, const
vertex_t & n) [inline]`

Return the range of neighbors of v, starting at n.

Example:

```
vvgraph S;
// initialize S
forall(const vertex& m, S.neighbors(v,n))
{
    // m is a neighbor of v
    // on the first loop, m == n
}
```

Definition at line 2620 of file vvgraph.h.

References graph::VVGrahp< VertexContent, EdgeContent, compact >::findInVertex(), graph::Vertex< VertexContent >::isNull(), graph::VVGrahp< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it, util::make_range(), and graph::VVGrahp< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood.

```

2621     {
2622         circ_neighbor_iterator_range result;
2623         if(v.isNull())
2624             return result;
2625         neighbor_found_t found = this->findInVertex(v, n);
2626         if(!found)
2627             return result;
2628         edge_list_t &lst = found.neighborhood->edges;
2629         result = util::make_range(circ_neighbor_iterator(lst.begin(), lst.end(), fo
und.it),
2630                                 circ_neighbor_iterator(lst.begin(), lst.end()));
2631         return result;
2632     }

```

17.81.5.47 template<GRAPH_TEMPLATE > VVGrahp< GRAPH_ARGS >::neighbor_iterator_range graph::VVGrahp< GRAPH_TEMPLATE >::neighbors (const vertex_t & v) [inline]

Return the range of neighbors of v.

Example:

```

vvgraph S;
// initialize S
forall(const vertex& v, S)
{
    forall(const vertex& n, S.neighbors(v))
    {
        // n is a neighbor of v
    }
}

```

Definition at line 2588 of file vvgraph.h.

References graph::VVGrahp< VertexContent, EdgeContent, compact >::findVertex(), graph::Vertex< VertexContent >::isNull(), util::make_range(), and graph::VVGrahp< VertexContent, EdgeContent, compact >::neighborhood.

```

2589     {
2590         neighbor_iterator_range result;
2591         if(v.isNull())
2592             return result;
2593         typename neighborhood_t::iterator it_found = this->findVertex(v);
2594         if(it_found == neighborhood.end())
2595             return result;
2596         edge_list_t &lst = it_found->second.edges;
2597         result = util::make_range(neighbor_iterator(lst.begin()),

```

```

2598                                     neighbor_iterator(lst.end()));
2599     return result;
2600 }

```

17.81.5.48 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::const_neighbor_iterator_range graph::VVGraph<
GRAPH_TEMPLATE >::neighbors (const vertex_t & v) const
[inline]`

Return the constant range of neighbors of *v*.

Example:

```

vvgraph S;
// initialize S
forall(const vertex& v, S)
{
    forall(const vertex& n, S.neighbors(v))
    {
        // n is a neighbor of v
    }
}

```

Definition at line 2572 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, `graph::Vertex< VertexContent >::isNull()`, `util::make_range()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`.

Referenced by `solver::Solver< nb_vars, identifier >::FindPartials()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::mergeCells()`, and `solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson()`.

```

2573     {
2574         const_neighbor_iterator_range result;
2575         if(v.isNull())
2576             return result;
2577         typename neighborhood_t::const_iterator it_found = this->findVertex(v);
2578         if(it_found == neighborhood.end())
2579             return result;
2580         const edge_list_t &lst = it_found->second.edges;
2581         result = util::make_range(const_neighbor_iterator(lst.begin()),
2582                                 const_neighbor_iterator(lst.end()));
2583         return result;
2584     }

```

17.81.5.49 `template<GRAPH_TEMPLATE > const VVGraph<
GRAPH_ARGS >::vertex_t & graph::VVGraph<
GRAPH_TEMPLATE >::nextTo (const vertex_t & v, const vertex_t
& neighbor, unsigned int n = 1) const [inline]`

Returns the *n*th vertex after *neighbor* in the neighborhood of *v*.

Returns

The vertex found or a null vertex if there is any problem.

Example:

```

vvgraph S;
// initialize S
forall(const vertex& v, S)
{
    forall(const vertex& n, S.neighbors(v))
    {
        const vertex& m, S.nextTo(v);
        // m is the vertex after n in v
    }
}

```

Definition at line 2332 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex(), graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it, graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood, and graph::Vertex< VertexContent >::null.

Referenced by vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >) >::mergeCells(), and algorithms::split().

```

2335     {
2336         const_neighbor_found_t found = this->findInVertex(v, ref);
2337         if(!found)
2338             return vertex_t::null;//vertex_t(0);
2339         typename edge_list_t::const_iterator it = found.it;
2340         const edge_list_t& lst = found.neighborhood->edges;
2341         for(unsigned int i = 0 ; i < n ; ++i)
2342         {
2343             ++it;
2344             if(it == lst.end())
2345                 it = lst.begin();
2346         }
2347         return it->target;
2348     }

```

17.81.5.50 **template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS >**
& graph::VVGraph< GRAPH_TEMPLATE >::operator= (const
VVGraph< VertexContent, EdgeContent, compact > & other)
[inline]

Copy the graph into another graph.

Definition at line 2766 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id, graph::VVGraph< VertexContent, EdgeContent, compact >::lookup, graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood,

graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache(), and
graph::VVGraph< VertexContent, EdgeContent, compact >::updateVertexCache().

```

2767     {
2768         //     for(const_iterator it = other.begin() ; it != other.end() ; ++it)
2769         //     {
2770         //         this->insert(*it);
2771         //     }
2772
2773         //     for(const_iterator it = other.begin() ; it != other.end() ; ++it)
2774         //     {
2775         //         vertex_t prev(0);
2776         //         const vertex_t& v = *it;
2777         //         const_neighbor_iterator_range ns = other.neighbors(v);
2778         //         for(const_neighbor_iterator itn = ns.first ; itn != ns.second ; ++i
2779         tn)
2780         //         {
2781         //             const vertex_t& n = *itn;
2782         //             if(prev)
2783         //                 this->splceAfter(v, prev, n);
2784         //             else
2785         //                 this->insertEdge(v,n);
2786         //             prev = n;
2787         //         }
2788
2789         // Copy the structure
2790         neighborhood = other.neighborhood;
2791         lookup.clear();
2792         typename neighborhood_t::iterator it;
2793         for(it = neighborhood.begin() ; it != neighborhood.end() ; ++it)
2794         {
2795             lookup[it->first] = it;
2796         }
2797         // And reconstruct the cache
2798         typename edge_list_t::iterator it_e;
2799         for(it = neighborhood.begin() ; it != neighborhood.end() ; ++it)
2800         {
2801             const vertex_t& src = it->first;
2802             updateVertexCache(src, it);
2803 #ifndef VVGRAPH_NO_EDGE_CACHE
2804             if(not compact)
2805             {
2806                 edge_list_t& lst = it->second.edges;
2807                 for(it_e = lst.begin() ; it_e != lst.end() ; ++it_e)
2808                 {
2809                     // Find the incoming vertex corresponding the this edge
2810                     typename lookup_t::iterator it_found = lookup.find(it_e->target);
2811                     const vertex_t& iv = *(it_found->second->second.in_edges.find(src));
2812                     //const vertex_t& iv = *(neighborhood[it_e->target].in_edges.find(src
2813                 ));
2814                 updateEdgeCache(src, it_e, iv);
2815                 it_e->set (&it->second, graph_id);
2816             }
2817 #endif
2818         }
2819         return *this;
2820     }

```


17.81.5.51 `template<GRAPH_TEMPLATE > bool graph::VVGraph<
GRAPH_TEMPLATE >::operator==(const VVGraph<
VertexContent, EdgeContent, compact > & other) const
[inline]`

Test equality for the graphs.

Two graphs are equal if they shared the same vertices and their neighborhood are equivalent.

Note

Current implementation do not consider invariance of neighborhood by rotation.

Definition at line 2719 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::lookup, and graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood.

```

2720     {
2721         if(neighborhood.size() != other.neighborhood.size())
2722             return false;
2723         for(typename neighborhood_t::const_iterator it = neighborhood.begin() ;
2724             it != neighborhood.end() ; ++it)
2725         {
2726             const vertex_t& v1 = it->first;
2727             typename lookup_t::const_iterator it_found = other.lookup.find(v1);
2728             if(it_found == other.lookup.end())
2729                 return false;
2730             if(it->second.edges.empty())
2731             {
2732                 if(!it_found->second->second.edges.empty())
2733                     return false;
2734             }
2735             else
2736             {
2737                 const edge_list_t& lst = it->second.edges;
2738                 const edge_list_t& olst = it_found->second->second.edges;
2739                 if(lst.size() != olst.size()) return false;
2740                 const vertex_t& v2 = lst.begin()->target;
2741                 bool found = false;
2742                 for(typename edge_list_t::const_iterator it_olst = olst.begin() ;
2743                     it_olst != olst.end() ; ++it_olst)
2744                 {
2745                     if(it_olst->target == v2)
2746                     {
2747                         found = true;
2748                         for(typename edge_list_t::const_iterator it_lst = lst.begin() ;
2749                             it_lst != lst.end() ; ++it_lst)
2750                         {
2751                             if(it_lst->target != it_olst->target) return false;
2752                             ++it_olst;
2753                             if(it_olst == olst.end()) it_olst = olst.begin();
2754                         }
2755                         break;
2756                     }
2757                 }
2758                 if(!found) return false;
2759             }

```

```

2760     }
2761     return true;
2762 }
```

17.81.5.52 `template<GRAPH_TEMPLATE > const VVGraph<GRAPH_ARGS >::vertex_t & graph::VVGraph<GRAPH_TEMPLATE >::operator[] (size_type idx) const [inline]`

Return the element of index *idx*.

Note

This is a convenience function whose complexity is $O(idx)$

Returns

The element *idx* position after the first one (using iterators)

Example:

```

vvgraph S;
// Fill in S
size_t i = 0, selected = 0;
forall(const vertex& v, S)
{
    if(condition(v))
    {
        selected = i;
        break;
    }
    ++i;
}
// ...
const vertex& v = S[selected]; // Work with the vertex previously selected
```

This construct is useful in case you need to refer to your vertices by 32bits numbers. It is true if you use selection with OpenGL for example.

Definition at line 1984 of file `vvgraph.h`.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`.

```

1985     {
1986         typename neighborhood_t::const_iterator it = neighborhood.begin();
1987         std::advance(it, idx);
1988         return it->first;
1989     }
```

17.81.5.53 `template<GRAPH_TEMPLATE > const VVGraph<
GRAPH_ARGS >::vertex_t & graph::VVGraph<
GRAPH_TEMPLATE >::prevTo(const vertex_t & v, const vertex_t
& ref, unsigned int n = 1) const [inline]`

Returns the nth vertex before ref in the neighborhood of v.

If ref is not in the neighborhood of v, return a null vertex.

Example:

```
vvgraph S;
// initialize S
forall(const vertex& v, S)
{
    forall(const vertex& n, S.neighbors(v))
    {
        const vertex& m, S.prevTo(v);
        // m is the vertex before n in v
    }
}
```

Definition at line 2352 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`, and `graph::Vertex< VertexContent >::null`.

```
2355     {
2356         const_neighbor_found_t found = this->findInVertex(v, ref);
2357         if(!found)
2358             return vertex_t::null;//vertex_t(0);
2359         typename edge_list_t::const_iterator it = found.it;
2360         const edge_list_t& lst = found.neighborhood->edges;
2361         for(unsigned int i = 0 ; i < n ; ++i)
2362         {
2363             if(it == lst.begin())
2364                 it = lst.end();
2365             --it;
2366         }
2367         return it->target;
2368     }
```

17.81.5.54 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> const vertex_t&
graph::VVGraph< VertexContent, EdgeContent, compact
>::reference(vertex_t v) const [inline]`

Get a reference on the vertex in the graph.

If the vertex do not exist in the graph, the function returns a null vertex.

Note

This function is useful if you obtained the vertex not from this graph but will use it intensively.

Example:

```

vvgraph S1, S2;
// Initialize S1 and S2 with the same vertices but different neighborhood
forall(const vertex& v1, S1)
{
    const vertex& v2 = S2.reference(v1);
    // Use v2 in S2 and v1 in v1 for speed optimization
}

```

Definition at line 780 of file vvgraph.h.

```

781     {
782         typename neighborhood_t::const_iterator found = this->findVertex(v);
783         if(found != neighborhood.end())
784             return found->first;
785         return vertex_t::null;//vertex_t(0);
786     }

```

17.81.5.55 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS >::edge_t graph::VVGraph< GRAPH_TEMPLATE >::replace(const vertex_t & v, const vertex_t & neighbor, const vertex_t & new_neighbor) [inline]`

Replace a vertex by another in a neighborhood.

Parameters

- ← *v* [Vertex](#) whose neighborhood is changed.
- ← *neighbor* [Vertex](#) to replace
- ← *new_neighbor* [Vertex](#) replacing neighbor

If *new_neighbor* is already in the neighborhood of *v*, then the operation fails and nothing is changed.

Returns

The edge between *v* and *new_neighbor* or a null edge if anything goes wrong.

Example:

```

vvgraph S;
// Initialize S
forall(const vertex& v, S)
{
    forall(const vertex& n, S.neighbors(v))
    {

```

```

        if(condition)
        {
            vertex n1;
            S.replace(v,n,n1); // Now n1 is where n was in v
            break; // The iteration must not continue after the neighborhood has been
                altered
        }
    }
}

```

Definition at line 2307 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex(), graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex(), graph::Vertex< VertexContent >::id(), graph::VVGraph< VertexContent, EdgeContent, compact >::insertInEdge(), graph::Vertex< VertexContent >::isNull(), graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it, util::tie(), and graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache().

Referenced by vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::connectFromJunctions(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteJunction(), vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::mergeCells(), and algorithms::split().

```

2310     {
2311         if(new_neighbor.isNull() or (v == new_neighbor) or (neighbor == new_neighbo
r))
2312             return edge_t();
2313         neighbor_found_t found = this->findInVertex(v, neighbor);
2314         if(!found)
2315             return edge_t();
2316         typename neighborhood_t::iterator n_found = this->findVertex(neighbor);
2317         inneighbor_iterator it_in;
2318         bool inserted;
2319         util::tie(it_in, inserted) = insertInEdge(v, new_neighbor);
2320         if(!inserted)
2321             return edge_t();
2322         n_found->second.in_edges.erase(v);
2323         found.it->target = new_neighbor;
2324         found.it->clear_edge();
2325         updateEdgeCache(v, found.it, *it_in);
2326         if(n_found->second.flagged and *n_found->second.flagged == neighbor) n_foun
d->second.flagged = 0;
2327         return edge_t(v.id(), new_neighbor.id(), &*found.it);
2328     }

```

17.81.5.56 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> size_type
graph::VVGraph< VertexContent, EdgeContent, compact >::size ()
const [inline]`

Return the number of vertexes on the graph.

Definition at line 791 of file vvgraph.h.

Referenced by `solver::Solver< nb_vars, identifier >::solveAdaptiveCrankNicholson()`, `solver::Solver< nb_vars, identifier >::solveAdaptiveEuler()`, `solver::Solver< nb_vars, identifier >::solveAdaptiveRungeKutta()`, `solver::Solver< nb_vars, identifier >::solveFixedpoint()`, and `graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::store_vertices()`.

```
791 { return neighborhood.size(); }
```

17.81.5.57 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> const vertex_t&
graph::VVGraph< VertexContent, EdgeContent, compact
>::source (const edge_t & edge) const [inline]`

Return the source vertex of the edge.

Example:

```
void fct(edge e, const vvgraph& S)
{
    const vertex& src = S.source(e);
    const vertex& tgt = S.target(e);
    // Work with the vertices
}
```

Definition at line 1201 of file vvgraph.h.

```
1202     {
1203         typename neighborhood_t::const_iterator found = this->findVertex(
vertex_t(edge.source()));
1204         if(found != neighborhood.end())
1205             return found->first;
1206         return vertex_t::null;//vertex_t(0);
1207     }
```

17.81.5.58 `template<GRAPH_TEMPLATE > template<typename
VertexContainer > std::vector< typename VVGraph<
GRAPH_ARGS >::edge_t > graph::VVGraph<
GRAPH_TEMPLATE >::spliceAfter (const vertex_t & v, const
vertex_t & neighbor, const VertexContainer & new_neighbor)
[inline]`

Splice a list of vertices after a neighbor.

The order of the vertices is kept.

Definition at line 2452 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex(), graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id, graph::Vertex< VertexContent >::id(), graph::VVGraph< VertexContent, EdgeContent, compact >::insertInEdge(), graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it, graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood, util::tie(), and graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache().

```

2455     {
2456         std::vector<edge_t> result;
2457         neighbor_found_t found = this->findInVertex(v, neighbor);
2458         if(!found) return result;
2459         ineighbor_iterator it_in;
2460         bool inserted;
2461         ++found.it;
2462         result.reserve(new_neighbors.size());
2463         for(typename VertexContainer::const_iterator it = new_neighbors.begin() ;
2464             it != new_neighbors.end() ; ++it) {
2465             const vertex_t& new_neighbor = *it;
2466             util::tie(it_in, inserted) = insertInEdge(v, new_neighbor);
2467             if(!inserted) return result;
2468             typename edge_list_t::iterator new_edge_it = found.neighborhood->edges.in
sert(found.it, neighbor_t(new_neighbor, EdgeContent(), *found.neighborhood,
graph_id));
2469             updateEdgeCache(v, new_edge_it, *it_in);
2470             result.push_back(edge_t(v.id(), new_neighbor.id(), &*new_edge_it));
2471         }
2472         return result;
2473     }

```

17.81.5.59 **template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS**
>::edge_t graph::VVGraph< GRAPH_TEMPLATE >::spliceAfter
(const vertex_t & v, const vertex_t & neighbor, const vertex_t &
new_neighbor) [inline]

Insert neighbor in the neighborhood of v after reference.

If new_neighbor is already in the neighborhood of v, the insertion fails.

Returns

The just created edge if everything succeed, or a null edge.

Example:

```

vvgraph S;
// Initialize S
forall(const vertex& v, S)
{
    forall(const vertex& n, S.neighbors(v))

```

```

    {
        if(condition)
        {
            vertex n1;
            S.spliceAfter(v,n,n1); // Now n1 is after n in v
            break; // The iteration must not continue after the neighborhood has been
                altered
        }
    }
}

```

Definition at line 2429 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id`, `graph::Vertex< VertexContent >::id()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insertInEdge()`, `graph::Vertex< VertexContent >::isNull()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`, `util::tie()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache()`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::connectFromJunctions()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::mergeCells()`, and `algorithms::split()`.

```

2432     {
2433         if(new_neighbor.isNull() || v == new_neighbor)
2434             return edge_t();
2435         neighbor_found_t found = this->findInVertex(v, neighbor);
2436         if(!found)
2437             return edge_t();
2438         inneighbor_iterator it_in;
2439         bool inserted;
2440         util::tie(it_in, inserted) = insertInEdge(v, new_neighbor);
2441         if(!inserted)
2442             return edge_t();
2443         ++found.it;
2444         typename edge_list_t::iterator new_edge_it = found.neighborhood->edges.insert(
            found.it, neighbor_t(new_neighbor, EdgeContent(), *found.neighborhood,
            graph_id));
2445         updateEdgeCache(v, new_edge_it, *it_in);
2446         return edge_t(v.id(), new_neighbor.id(), &*new_edge_it);
2447     }

```


17.81.5.60 `template<GRAPH_TEMPLATE > template<typename
VertexContainer > std::vector< typename VVGraph<
GRAPH_ARGS >::edge_t > graph::VVGraph<
GRAPH_TEMPLATE >::spliceBefore (const vertex_t & v, const
vertex_t & neighbor, const VertexContainer & new_neighbor)
[inline]`

Splice a list of vertices before a neighbor.

The order of the vertices is kept.

Definition at line 2499 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id`, `graph::Vertex< VertexContent >::id()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insertInEdge()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`, `util::tie()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache()`.

```

2502     {
2503         std::vector<edge_t> result;
2504         neighbor_found_t found = this->findInVertex(v, neighbor);
2505         if(!found) return result;
2506         inneighbor_iterator it_in;
2507         bool inserted;
2508         result.reserve(new_neighbors.size());
2509         for(typename VertexContainer::const_iterator it = new_neighbors.begin() ;
2510             it != new_neighbors.end() ; ++it) {
2511             const vertex_t& new_neighbor = *it;
2512             util::tie(it_in, inserted) = insertInEdge(v, new_neighbor);
2513             if(!inserted) return result;
2514             typename edge_list_t::iterator new_edge_it = found.neighborhood->edges.in
sert(found.it, neighbor_t(new_neighbor, EdgeContent(), *found.neighborhood,
graph_id));
2515             updateEdgeCache(v, new_edge_it, *it_in);
2516             result.push_back(edge_t(v.id(), new_neighbor.id(), &*new_edge_it));
2517         }
2518         return result;
2519     }

```

17.81.5.61 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::edge_t graph::VVGraph< GRAPH_TEMPLATE
>::spliceBefore (const vertex_t & v, const vertex_t & neighbor,
const vertex_t & new_neighbor) [inline]`

Insert neighbor in the neighborhood of v before reference.

If `new_neighbor` is already in the neighborhood of v, the insertion fails.

Returns

The just created edge if everything succeed, or a null edge.

Example:

```

vvgraph S;
// Initialize S
forall(const vertex& v, S)
{
    forall(const vertex& n, S.neighbors(v))
    {
        if(condition)
        {
            vertex n1;
            S.spliceBefore(v,n,n1); // Now n1 is before n in v
            break; // The iteration must not continue after the neighborhood has been
                altered
        }
    }
}

```

Definition at line 2477 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id`, `graph::Vertex< VertexContent >::id()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insertInEdge()`, `graph::Vertex< VertexContent >::isNull()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::neighborhood`, `util::tie()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache()`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::connectFromJunctions()`, `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::mergeCells()`, and `algorithms::split()`.

```

2480     {
2481         if(new_neighbor.isNull() || v == new_neighbor)
2482             return edge_t();
2483         neighbor_found_t found = this->findInVertex(v, neighbor);
2484         if(!found)
2485             return edge_t();
2486         inneighbor_iterator it_in;
2487         bool inserted;
2488         util::tie(it_in, inserted) = insertInEdge(v, new_neighbor);
2489         if(!inserted)
2490             return edge_t();
2491         typename edge_list_t::iterator new_edge_it = found.neighborhood->edges.inse
rt(found.it, neighbor_t(new_neighbor, EdgeContent(), *found.neighborhood,
graph_id));
2492         updateEdgeCache(v, new_edge_it, *it_in);
2493         return edge_t(v.id(), new_neighbor.id(), &*new_edge_it);
2494     }

```

17.81.5.62 `template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> void
graph::VVGraph< VertexContent, EdgeContent, compact
>::store_vertices (std::vector< vertex_t > & vertices, bool cached =
true) const [inline]`

Store the vertices in a STL container.

By default, the cache will be setup for quick vertex access.

Example:

```

vvgraph S;
// Initialize S
std::vector<vertex> vertices;
S.store_vertices(vertices);
forall(const vertex& v, vertices)
{
    // The cache information is used to speed up lookup
    if(condition)
    {
        S.erase(v); // Valid, but do not use v in S after that
    }
}

```

Definition at line 1584 of file vvgraph.h.

```

1585     {
1586         vertices.resize(this->size(), vertex_t::null);
1587         size_t k = 0;
1588         for(typename neighborhood_t::const_iterator it = neighborhood.begin() ;
1589
1590             it != neighborhood.end() ; ++it)
1591         {
1592             const vertex_t& v = it->first;
1593             vertices[k] = v;
1594             #ifndef VVGRAPH_NO_CACHE
1595             if(cached)
1596             {
1597                 vertices[k].cache = v.cache;
1598                 vertices[k].cache_source = v.cache_source;
1599             }
1600             #endif
1601             ++k;
1602         }
1603     }

```

17.81.5.63 `template<GRAPH_TEMPLATE > template<typename
VertexContainer > VVGraph< GRAPH_ARGS >
graph::VVGraph< GRAPH_TEMPLATE >::subgraph (const
VertexContainer & vertexes) const [inline]`

Extract the subgraph containing the vertexes.

The subgraph is the set of vertexes and the edges whose source and target are in the extracted vertexes.

Example:

```

vvgraph S;
// Initialize S
std::vector<vertex> to_keep;
// Fill in to_keep with a subset of the vertices of S
vvgraph S1 = S.subgraph(to_keep);

```

Definition at line 2652 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::contains()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insert()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insertInEdge()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >::it`, `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`, `graph::VVGraph< VertexContent, EdgeContent, compact >::updateEdgeCache()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::updateVertexCache()`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::saveSubgraphs()`.

```

2653     {
2654         typename VertexContainer::const_iterator it;
2655         VVGraph result;
2656         // First, insert the vertexes in the result graph
2657         for(it = verts.begin() ; it != verts.end() ; ++it)
2658         {
2659             result.insert(*it);
2660         }
2661         typename neighborhood_t::const_iterator it_orign;
2662         typename neighborhood_t::iterator it_n;
2663         typename edge_list_t::const_reverse_iterator it_sn;
2664         // Then, creates the in and out edges
2665         for(it_n = result.neighborhood.begin() ; it_n != result.neighborhood.end()
; ++it_n)
2666         {
2667             const vertex_t& v = it_n->first;
2668             result.updateVertexCache(v, it_n);
2669             // For each vertex, find out which edges to add
2670             it_orign = this->findVertex(v);
2671             const edge_list_t& orig_lst = it_orign->second.edges;
2672             edge_list_t& lst = it_n->second.edges;
2673             // Going backward because it is easier to retrieve the iterator ...
2674             for( it_sn = orig_lst.rbegin() ; it_sn != orig_lst.rend() ; ++it_sn )
2675             {
2676                 // For each existing edge from the vertex, keep only the ones whose
2677                 // target are on the subgraph
2678                 if(result.contains(it_sn->target))
2679                 {
2680                     // Insert the incoming edge
2681                     inneighbor_iterator it_in = result.insertInEdge(v, it_sn->target).firs
t;
2682                     // Insert the outgoing edge
2683                     lst.push_front(neighbor_t(it_sn->target, *it_sn, it_n->second, result
.graph_id));
2684                     // Update the cache

```

```

2685         result.updateEdgeCache(v, lst.begin(), *it_in);
2686     }
2687 }
2688 // At last, update the flag, if any
2689 if(it_orign->second.flagged and result.contains(*it_orign->second.flagged
))
2690 {
2691     const_neighbor_found_t found = this->findInVertex(v, *it_orign->second.
flagged);
2692     it_n->second.flagged = &(found.it->target);
2693 }
2694 }
2695 return result;
2696 }
```

17.81.5.64 template<GRAPH_TEMPLATE > void graph::VVGraph<
GRAPH_TEMPLATE >::swap (VVGraph< VertexContent,
EdgeContent, compact > & other) [inline]

Swap the content of two graphs.

Definition at line 2823 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::graph_
id, graph::VVGraph< VertexContent, EdgeContent, compact >::lookup, and
graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood.

Referenced by std::swap().

```

2824 {
2825     neighborhood.swap(other.neighborhood);
2826     lookup.swap(lookup);
2827     std::swap(graph_id, other.graph_id);
2828 }
```

17.81.5.65 template<typename VertexContent, typename EdgeContent
= _EmptyEdgeContent, bool compact = false> const vertex_t&
graph::VVGraph< VertexContent, EdgeContent, compact >::target
(const edge_t & edge) const [inline]

Return the target vertex of the edge.

Example:

```

void fct(edge e, const vvgraph& S)
{
    const vertex& src = S.source(e);
    const vertex& tgt = S.target(e);
    // Work with the vertices
}
```

Definition at line 1230 of file vvgraph.h.

```

1231 {
```

```

1232         typename neighborhood_t::const_iterator found = this->findVertex(
vertex_t(edge.target()));
1233         if(found != neighborhood.end())
1234             return found->first;
1235         return vertex_t::null;//vertex_t(0);
1236     }

```

17.81.5.66 `template<GRAPH_TEMPLATE > void graph::VVGraph<GRAPH_TEMPLATE >::undoInEdge (const vertex_t & new_neighbor, ineighbor_iterator & it) [inline, protected]`

Undo the insertion of an in edge as a roll-back mechanism if the whole edge insertion fails.

Definition at line 2272 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::lookup.

```

2274     {
2275         typename lookup_t::iterator found = lookup.find(new_neighbor);
2276         //typename neighborhood_t::iterator found = neighborhood.find(new_neighbor)
;
2277         //if(found == neighborhood.end())
2278         //return;
2279         if(found == lookup.end())
2280             return;
2281         found->second->second.in_edges.erase(it);
2282     }

```

17.81.5.67 `template<GRAPH_TEMPLATE > void graph::VVGraph<GRAPH_TEMPLATE >::updateEdgeCache (const vertex_t & v, typename edge_list_t::iterator new_edge_it, const vertex_t & in_edge) [inline, protected]`

Update the edge cache information of vertex v in its neighborhood and the in_edge too.

Definition at line 2285 of file vvgraph.h.

References graph::Vertex< VertexContent >::cache, graph::Vertex< VertexContent >::cache_source, graph::Vertex< VertexContent >::id(), and graph::VVGraph< VertexContent, EdgeContent, compact >::neighbor_t::target.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdge(), graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdges(), graph::VVGraph< VertexContent, EdgeContent, compact >::operator=(), graph::VVGraph< VertexContent, EdgeContent, compact >::replace(), graph::VVGraph< VertexContent, EdgeContent, compact >::spliceAfter(), graph::VVGraph< VertexContent, EdgeContent, compact >::spliceBefore(), and graph::VVGraph< VertexContent, EdgeContent, compact >::subgraph().

```

2288     {

```

```

2289 #ifndef VVGRAPH_NO_EDGE_CACHE
2290     if(not compact)
2291     {
2292         // Caching in the neighbor
2293         neighbor_t& neighbor = *new_edge_it;
2294         neighbor.setIt(new_edge_it);
2295         neighbor.target.cache_source = v.id();
2296         neighbor.target.cache = reinterpret_cast<void*>(&neighbor);
2297
2298         // Caching in the incoming edge source
2299         in_edge.cache_source = v.id();
2300         in_edge.cache = reinterpret_cast<void*>(&neighbor);
2301     }
2302 #endif // VVGRAPH_NO_EDGE_CACHE
2303     }

```

17.81.5.68 `template<GRAPH_TEMPLATE > void graph::VVGraph<
GRAPH_TEMPLATE >::updateVertexCache (const vertex_t & v,
typename neighborhood_t::iterator it) [inline, protected]`

Update the vertex cache for vertex v.

Definition at line 1948 of file vvgraph.h.

References graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact >::insert(),
graph::VVGraph< VertexContent, EdgeContent, compact >::operator=(), and
graph::VVGraph< VertexContent, EdgeContent, compact >::subgraph().

```

1950     {
1951 #ifndef VVGRAPH_NO_VERTEX_CACHE
1952         it->second.update(graph_id, v, it);
1953 #endif
1954     }

```

17.81.5.69 `template<GRAPH_TEMPLATE > VVGraph< GRAPH_ARGS
>::size_type graph::VVGraph< GRAPH_TEMPLATE >::valence
(const vertex_t & v) const [inline]`

Returns the number of neighbors of v.

If v is not in the graph, the behavior is undefined.

Example:

```

vvgraph S;
// initialize S
forall(const vertex& v, S)
{
    size_t nb_neighbors = S.valence(v); // Number of neighbors of v in S
    // ...
}

```

Definition at line 2006 of file vvgraph.h.

References `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, `graph::Vertex< VertexContent >::isNull()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::neighborhood`.

Referenced by `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::border()`, and `vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteJunction()`.

```

2007     {
2008         if(v.isNull())
2009             return 0;
2010         typename neighborhood_t::const_iterator it_found = this->findVertex(v);
2011         if(it_found == neighborhood.end())
2012             return 0;
2013         return it_found->second.edges.size();
2014     }

```

17.81.6 Member Data Documentation

17.81.6.1 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> intptr_t graph::VVGraph< VertexContent, EdgeContent, compact >::graph_id [protected]`

Unique graph identifier.

Definition at line 1807 of file `vvgraph.h`.

Referenced by `graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdge()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdges()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::operator=()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::spliceAfter()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::spliceBefore()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::subgraph()`, `graph::VVGraph< VertexContent, EdgeContent, compact >::swap()`, and `graph::VVGraph< VertexContent, EdgeContent, compact >::updateVertexCache()`.

17.81.6.2 `template<typename VertexContent, typename EdgeContent = _EmptyEdgeContent, bool compact = false> lookup_t graph::VVGraph< VertexContent, EdgeContent, compact >::lookup [protected]`

Hash map to optimise look-up.

Definition at line 1802 of file `vvgraph.h`.

Referenced by graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::clear(), graph::VVGraph< VertexContent, EdgeContent, compact >::count(), graph::VVGraph< VertexContent, EdgeContent, compact >::erase(), graph::VVGraph< VertexContent, EdgeContent, compact >::eraseAllEdges(), graph::VVGraph< VertexContent, EdgeContent, compact >::eraseEdge(), graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex(), graph::VVGraph< VertexContent, EdgeContent, compact >::insert(), graph::VVGraph< VertexContent, EdgeContent, compact >::insertInEdge(), graph::VVGraph< VertexContent, EdgeContent, compact >::operator=(), graph::VVGraph< VertexContent, EdgeContent, compact >::operator==(), graph::VVGraph< VertexContent, EdgeContent, compact >::swap(), and graph::VVGraph< VertexContent, EdgeContent, compact >::undoInEdge().

**17.81.6.3 template<typename VertexContent, typename EdgeContent =
 _EmptyEdgeContent, bool compact = false> neighborhood_t
 graph::VVGraph< VertexContent, EdgeContent, compact
 >::neighborhood [protected]**

Main data structure containing everything.

Definition at line 1797 of file vvgraph.h.

Referenced by graph::VVGraph< VertexContent, EdgeContent, compact >::any(), graph::VVGraph< VertexContent, EdgeContent, compact >::anyIn(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::begin(), graph::VVGraph< VertexContent, EdgeContent, compact >::clear(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::clear(), graph::VVGraph< VertexContent, EdgeContent, compact >::contains(), graph::VVGraph< VertexContent, EdgeContent, compact >::empty(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::empty(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::end(), graph::VVGraph< VertexContent, EdgeContent, compact >::erase(), graph::VVGraph< VertexContent, EdgeContent, compact >::eraseAllEdges(), graph::VVGraph< VertexContent, EdgeContent, compact >::eraseEdge(), graph::VVGraph< VertexContent, EdgeContent, compact >::findInVertex(), graph::VVGraph< VertexContent, EdgeContent, compact >::findVertex(), graph::VVGraph< VertexContent, EdgeContent, compact >::flagged(), graph::VVGraph< VertexContent, EdgeContent, compact >::iAnyIn(), graph::VVGraph< VertexContent, EdgeContent, compact >::iEmpty(), graph::VVGraph< VertexContent, EdgeContent, compact >::iNeighbors(), graph::VVGraph< VertexContent, EdgeContent, compact >::insert(), graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdge(), graph::VVGraph< VertexContent, EdgeContent, compact >::insertEdges(), graph::VVGraph< VertexContent, EdgeContent, compact >::iValence(), graph::VVGraph< VertexContent, EdgeContent, com-

```

pact >::neighbors(), graph::VVGraph< VertexContent, EdgeContent, compact >::operator=(), graph::VVGraph< VertexContent, EdgeContent, compact >::operator==( ), graph::VVGraph< VertexContent, EdgeContent, compact >::operator[ ](), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::reference(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::size(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::source(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::store_vertices(), graph::VVGraph< VertexContent, EdgeContent, compact >::subgraph(), graph::VVGraph< VertexContent, EdgeContent, compact >::swap(), graph::VVGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), graph::_EmptyEdgeContent, false >::target(), and graph::VVGraph< VertexContent, EdgeContent, compact >::valence().

```

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

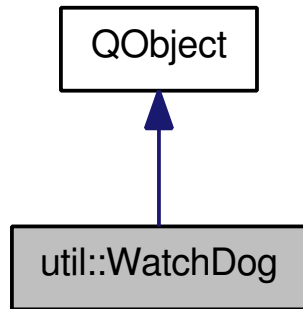
- [vvelib/graph/vvgraph.h](#)

17.82 util::WatchDog Class Reference

Watch the modifications of file objects for you.

```
#include <util/watchdog.h>
```

Inheritance diagram for util::WatchDog:



Signals

- void **registerFile** (std::string)
- void **unregisterFile** (std::string)

Public Member Functions

- void **addObject** ([FileObject](#) &obj)
Watch a new object.
- void **addObject** ([FileObject](#) *obj)
Watch a new object.
- void **changeFilename** ([FileObject](#) *obj, std::string fn)
Change the filename of an object.
- bool **guarded** ([FileObject](#) &obj) const
- bool **guarded** ([FileObject](#) *obj) const
Test if an object is being watched.
- void **removeObject** ([FileObject](#) &obj)
Do not watch this object anymore.
- void **removeObject** ([FileObject](#) *obj)
Do not watch this object anymore.
- bool **watch** (const std::set< std::string > &filenames)

Check if any of the file names correspond to one of the watched object.

- [WatchDog](#) ([QObject](#) *m)

Constructor.

Protected Attributes

- `std::set< FileObject * > fileObjects`

List of file objects to handle.

- `QObject * model`

[Model](#) to guard.

- `std::map< std::string, std::set< FileObject * > > names`

Map the file names to the file objects.

17.82.1 Detailed Description

Watch the modifications of file objects for you.

Definition at line 134 of file watchdog.h.

17.82.2 Constructor & Destructor Documentation

17.82.2.1 `util::WatchDog::WatchDog (QObject * m)`

Constructor.

Parameters

m Pointer toward the model to watch files for

Definition at line 25 of file watchdog.cpp.

References [QObject::connect\(\)](#).

```

26     : QObject()
27     , model(m)
28     {
29         connect(this, SIGNAL(registerFile(std::string)), m, SLOT(registerFile(std::string)));
30         connect(this, SIGNAL(unregisterFile(std::string)), m, SLOT(unregisterFile(std::string)));
31     }
```

17.82.3 Member Function Documentation

17.82.3.1 void util::WatchDog::addObject (FileObject & *obj*) [inline]

Watch a new object.

Definition at line 159 of file watchdog.h.

References addObject().

Referenced by addObject().

```
159 { addObject(&obj); }
```

17.82.3.2 void util::WatchDog::addObject (FileObject * *obj*)

Watch a new object.

Definition at line 33 of file watchdog.cpp.

References fileObjects, util::FileObject::getFilename(), and names.

Referenced by tissue_model::TissueModel< RealModel, TissueClass >::TissueModel(), and bspline_tissue_model::TissueModel< RealModel, TissueClass >::TissueModel().

```
34 {
35     fileObjects.insert(obj);
36     names[obj->getFilename()].insert(obj);
37     if(!obj->getFilename().empty())
38         emit registerFile(obj->getFilename());
39 }
```

17.82.3.3 void util::WatchDog::changeFilename (FileObject * *obj*, std::string *fn*)

Change the filename of an object.

Note

When an object is handled by the [WatchDog](#), you have to use this method to change its filename.

Definition at line 46 of file watchdog.cpp.

References util::FileObject::getFilename(), names, and util::FileObject::setFilename().

Referenced by bspline_tissue_model::TissueModel< RealModel, TissueClass >::readTissueParam().

```
47 {
48     // First, unregister the file
```

```

49     std::string oldName = obj->getFilename();
50     if(oldName == fn)
51         return;
52     std::set<FileObject*>& ns = names[oldName];
53     if(!oldName.empty())
54         emit_unregisterFile(oldName);
55     ns.erase(obj);
56     if(ns.empty())
57         names.erase(oldName);
58     // Then add the new file
59     names[fn].insert(obj);
60     if(!fn.empty())
61         emit_registerFile(fn);
62     obj->setFilename(fn);
63 }

```

17.82.3.4 bool util::WatchDog::guarded (FileObject * *obj*) const

Test if an object is being watched.

Definition at line 41 of file watchdog.cpp.

References fileObjects.

```

42     {
43         return (fileObjects.find(obj) != fileObjects.end());
44     }

```

17.82.3.5 void util::WatchDog::removeObject (FileObject & *obj*) [inline]

Do not watch this object anymore.

Definition at line 174 of file watchdog.h.

References removeObject().

Referenced by removeObject().

```

174 { removeObject(&obj); }

```

17.82.3.6 void util::WatchDog::removeObject (FileObject * *obj*)

Do not watch this object anymore.

Definition at line 65 of file watchdog.cpp.

References fileObjects, util::FileObject::getFilename(), and names.

```

66     {
67         std::string name = obj->getFilename();
68         std::set<FileObject*>& ns = names[name];
69         if(!name.empty())
70             emit_unregisterFile(name);

```

```

71     ns.erase(obj);
72     if(ns.empty())
73         names.erase(name);
74     fileObjects.erase(obj);
75 }
```

17.82.3.7 bool util::WatchDog::watch (const std::set< std::string > & *filenames*)

Check if any of the file names correspond to one of the watched object.

Parameters

filenames Set of file names that were recently modified

If the [WatchDog](#) finds a file name it handles in the set, it emits a [FileObject::modified\(\)](#) signal for this object.

Definition at line 78 of file watchdog.cpp.

References [forall](#), and [names](#).

Referenced by [tissue_model::TissueModel< RealModel, TissueClass >::modifiedFiles\(\)](#), and [bspline_tissue_model::TissueModel< RealModel, TissueClass >::modifiedFiles\(\)](#).

```

79 {
80     bool result = false;
81     forall(const std::string& str, filenames)
82     {
83         std::map<std::string, std::set<FileObject*> >::iterator found = names.find(
84             str);
85         if(found != names.end())
86         {
87             result = true;
88             forall(FileObject* obj, found->second)
89             {
90                 obj->update();
91             }
92         }
93     }
94     return result;
95 }
```

17.82.4 Member Data Documentation

17.82.4.1 std::set<FileObject*> util::WatchDog::fileObjects [protected]

List of file objects to handle.

Definition at line 198 of file watchdog.h.

Referenced by [addObject\(\)](#), [guarded\(\)](#), and [removeObject\(\)](#).

17.82.4.2 QObject* util::WatchDog::model [protected]

[Model](#) to guard.

Definition at line 194 of file watchdog.h.

**17.82.4.3 std::map<std::string, std::set<FileObject*> >
util::WatchDog::names [protected]**

Map the file names to the file objects.

Definition at line 202 of file watchdog.h.

Referenced by addObject(), changeFilename(), removeObject(), and watch().

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

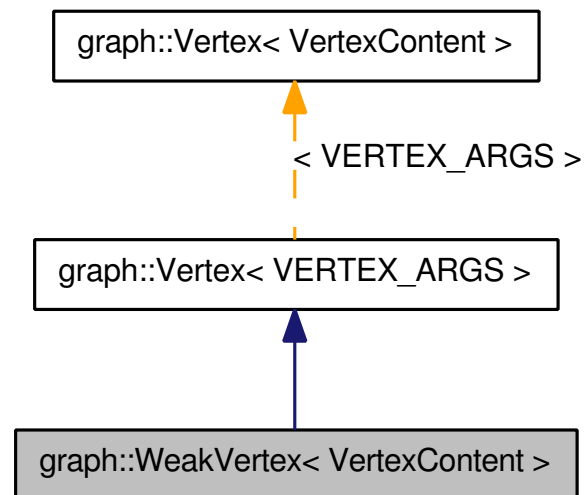
- vvelib/util/[watchdog.h](#)
- vvelib/util/watchdog.cpp

17.83 graph::WeakVertex< VertexContent > Class Template Reference

Weak pointer on a vertex.

```
#include <graph/vertex.h>
```

Inheritance diagram for graph::WeakVertex< VertexContent >:



Public Types

- typedef VertexContent `content_t`
Type of the content of the vertex.
- typedef `vertex_identity_t` `identity_t`
Type of the identifier of the vertex.
- typedef VertexContent * `pointer`
Type of the equivalent pointer.
- typedef `Vertex< VERTEX_ARGS >` `strong_ref`
Strong reference corresponding to the weak one.

Public Member Functions

- bool `isNull ()` const
Test if a vertex is a null vertex.

- [WeakVertex](#) & **operator=** (const [strong_ref](#) &other)
- [WeakVertex](#) & **operator=** (const [identity_t](#) &id)
- [WeakVertex](#) & **operator=** (const [WeakVertex](#) &other)

Set the content of the weak reference.

- [WeakVertex](#) (const [identity_t](#) &id)

Construct a weak reference from an id.

- [WeakVertex](#) (const [WeakVertex](#) ©)

Copy constructor.

- [WeakVertex](#) (const [strong_ref](#) &v)

Construct a weak reference from a strong one.

- [WeakVertex](#) ()

Construct an empty weak vertex.

17.83.1 Detailed Description

```
template<typename VertexContent> class graph::WeakVertex< VertexContent
>
```

Weak pointer on a vertex. Be careful using this class as there is no garanty the vertex still exists.

Definition at line 503 of file vertex.h.

17.83.2 Member Typedef Documentation

17.83.2.1 `template<typename VertexContent> typedef VertexContent
graph::WeakVertex< VertexContent >::content_t`

Type of the content of the vertex.

Reimplemented from [graph::Vertex< VERTEX_ARGS >](#).

Definition at line 516 of file vertex.h.

17.83.2.2 `template<typename VertexContent> typedef vertex_identity_t
graph::WeakVertex< VertexContent >::identity_t`

Type of the identifier of the vertex.

Reimplemented from [graph::Vertex< VERTEX_ARGS >](#).

Definition at line 511 of file vertex.h.

17.83.2.3 `template<typename VertexContent> typedef VertexContent* graph::WeakVertex< VertexContent >::pointer`

Type of the equivalent pointer.

Reimplemented from [graph::Vertex< VERTEX_ARGS >](#).

Definition at line 526 of file vertex.h.

17.83.2.4 `template<typename VertexContent> typedef Vertex<VERTEX_ARGS> graph::WeakVertex< VertexContent >::strong_ref`

Strong reference corresponding to the weak one.

Definition at line 521 of file vertex.h.

17.83.3 Constructor & Destructor Documentation

17.83.3.1 `template<typename VertexContent> graph::WeakVertex< VertexContent >::WeakVertex () [inline]`

Construct an empty weak vertex.

At the difference of normal vertices, constructing a weak vertex creates a null one.

Definition at line 534 of file vertex.h.

```
535         : Vertex<VERTEX_ARGS>(0)
536     {}
```

17.83.3.2 `template<typename VertexContent> graph::WeakVertex< VertexContent >::WeakVertex (const strong_ref & v) [inline]`

Construct a weak reference from a strong one.

Definition at line 541 of file vertex.h.

References `graph::Vertex< VertexContent >::_content`, and `graph::Vertex< VERTEX_ARGS >::_content`.

```
542         : Vertex<VERTEX_ARGS>(0)
543     {
544         this->_content = v._content;
545     }
```

17.83.3.3 `template<typename VertexContent> graph::WeakVertex< VertexContent >::WeakVertex (const WeakVertex< VertexContent > & copy) [inline]`

Copy constructor.

Definition at line 550 of file vertex.h.

References `graph::Vertex< VertexContent >::_content`, and `graph::Vertex< VERTEX_ARGS >::_content`.

```

551         : Vertex<VERTEX_ARGS>(0)
552     {
553         this->_content = copy._content;
554     }
```

17.83.3.4 `template<typename VertexContent> graph::WeakVertex< VertexContent >::WeakVertex (const identity_t & id) [inline, explicit]`

Construct a weak reference from an id.

Definition at line 566 of file vertex.h.

References `graph::Vertex< VERTEX_ARGS >::_content`.

```

567         : Vertex<VERTEX_ARGS>(0)
568     {
569         this->_content = reinterpret_cast<real_pointer>(id);
570         if (this->_content and this->_content->count == 0)
571             this->_content = 0;
572     }
```

17.83.4 Member Function Documentation

17.83.4.1 `template<typename VertexContent> bool graph::WeakVertex< VertexContent >::isNull () const [inline]`

Test if a vertex is a null vertex.

Reimplemented from [graph::Vertex< VERTEX_ARGS >](#).

Definition at line 608 of file vertex.h.

References `graph::Vertex< VERTEX_ARGS >::_content`.

```

609     {
610         return this->_content == 0 or this->_content->count == 0;
611     }
```

17.83.4.2 `template<typename VertexContent> WeakVertex& graph::WeakVertex< VertexContent >::operator= (const WeakVertex< VertexContent > & other) [inline]`

Set the content of the weak reference.

Definition at line 577 of file vertex.h.

References `graph::Vertex< VertexContent >::_content`, and `graph::Vertex< VERTEX_ARGS >::_content`.

```
578     {  
579         this->_content = other._content;  
580         return *this;  
581     }
```

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

- `vvelib/graph/vertex.h`

Chapter 18

File Documentation

18.1 doc/examples/context_menu.cpp File Reference

Example of a viewer that creates a context menu.

```
#include <vve.h>
#include <QContextMenuEvent>
#include <QAction>
#include <QMenu>
#include <iostream>
#include "model.moc"
```

Functions

- **DEFINE_MODEL** (MyModel)
- **DEFINE_VIEWER** (MyViewer)

18.1.1 Detailed Description

Example of a viewer that creates a context menu. You will find a complete description of the content in the section [Creating a context menu](#)

Definition in file [context_menu.cpp](#).

18.2 doc/examples/hello_world.cpp File Reference

Simple Hello World example.

```
#include <vve.h>
```

Functions

- **DEFINE_MODEL** (HelloWorldModel)

18.2.1 Detailed Description

Simple Hello World example.

Definition in file [hello_world.cpp](#).

18.3 doc/examples/select.cpp File Reference

Example with 3D object selection.

```
#include <vve.h>
#include <util/vector.h>
```

Typedefs

- typedef [util::Vector](#)< 3, double > **Point3d**
- typedef [vvgraph::vertex_t](#) **vertex**
- typedef [graph::VVGraph](#)< VertexContent > **vvgraph**

Functions

- **DEFINE_MODEL** (SelectModel)
- **DEFINE_VIEWER** (SelectViewer)

18.3.1 Detailed Description

Example with 3D object selection.

Definition in file [select.cpp](#).

18.4 doc/examples/simple_model.cpp File Reference

Minimal model file.

```
#include <vve.h>
```

Functions

- **DEFINE_MODEL** (MyModel)

18.4.1 Detailed Description

Minimal model file.

Definition in file [simple_model.cpp](#).

18.5 vvelib/algorithms/cellsystem.h File Reference

Include the definitions necessary for the cell system division algorithm and model.

```
#include <config.h>
#include <algorithms/complex.h>
#include <geometry/intersection.h>
#include <geometry/area.h>
#include <util/matrix.h>
#include <model.h>
#include <util/parms.h>
#include <sstream>
#include <algorithms/tissue.h>
#include <storage/storage.h>
#include <storage/complex.h>
#include <iostream>
```

Classes

- class [cell_system::CellSystem< Complex, MyModel >](#)
Full cell-system model.
- class [cell_system::CellSystemCell](#)
Content of a vertex for the 2D cell system.
- class [cell_system::CellSystemDivisionParams](#)
Parameters for cell-system division.
- class [cell_system::CellSystemJunction](#)
Content of a vertex for the 2D cell system.
- class [cell_system::DivisionParams](#)
Structure storing the right hand side of a division rule.

Namespaces

- namespace [cell_system](#)
Namespace defining the cell system division algorithm and model.

Defines

- `#define EPSILON 0.00001`

Typedefs

- `typedef int cell_system::label_t`
Type of the label of a cell (for full cell-system only).

Functions

- `template<class Complex >`
`DivisionData< typename Complex::junction_content > cell_system::findDivisionPoints (const typename Complex::cell &c, Complex &T, const CellSystemDivisionParams ¶ms)`
Division algorithm for cell-system.
- `template<class Complex >`
`DivisionData< typename Complex::junction_content > cell_system::findDivisionPoints (const typename Complex::cell &c, Point3d ¢er, Complex &T, const Point3d &direction)`
Find the division points for a wall going through a given point.
- `std::string cell_system::removeWhitespace (std::string s)`
Convenience function that removes white spaces before and after the string.
- `template<class Complex >`
`double cell_system::volumeLeftCell (const typename Complex::cell &c, Complex &T, const Point3d ¢er, const DivisionData< typename Complex::junction_content > &result)`
Compute the volume of the left cell of a division.

18.5.1 Detailed Description

Include the definitions necessary for the cell system division algorithm and model.

Definition in file [cellsystem.h](#).

18.6 vvelib/algorithms/complex.h File Reference

Definition of a 2D cell complex.

```
#include <config.h>
#include <graph/vvgraph.h>
#include <graph/vvbigraph.h>
#include <algorithms/insert.h>
#include <util/parms.h>
#include <util/palette.h>
#include <util/color.h>
#include <util/leaf_class.h>
#include <util/assert.h>
#include <geometry/intersection.h>
#include <geometry/area.h>
#include <storage/fwd.h>
#include <iostream>
```

Classes

- class [vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >::division_result_t](#)
Describe the result of a cell division.
- class [vvcomplex::DivisionData< JunctionContent >](#)
Class holding the data needed to actually divide a cell.
- class [vvcomplex::InModelDivisionParam](#)
Class used to define the division parameters from within the model class.
- class [vvcomplex::VVComplex< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass >](#)
Class handling the representation and development of 2D cell complex.
- class [vvcomplex::VVComplexGraph< CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, compact >](#)
Definition of the bipartite graph with specialized methods for the complex graph.

Namespaces

- namespace [vvcomplex](#)
Definition of classes and functions for 2d cell complexes.

Defines

- #define **ALL_COMPLEX_TEMPLATE_ARGS** [Model](#), CellContent, JunctionContent, WallContent, CellContent
- #define **COMPLEX_ARGS** CellContent, JunctionContent, CellJunctionContent, JunctionCellContent, com
- #define [EXPORT_COMPLEX_EDGES](#)(ComplexClass)
Export the types of the edges of the complex, i.e.
- #define [EXPORT_COMPLEX_EDGES_PREFIX](#)(ComplexClass, prefix)
Export the types of the edges of the complex, i.e.
- #define [EXPORT_COMPLEX_GRAPHS](#)(ComplexClass)
Export the types of the graphs of the complex, i.e.
- #define [EXPORT_COMPLEX_GRAPHS_PREFIX](#)(ComplexClass, prefix)
Export the types of the graphs of the complex, i.e.
- #define [EXPORT_COMPLEX_TYPES](#)(ComplexClass)
Export all the types of the graph.
- #define [EXPORT_COMPLEX_TYPES_PREFIX](#)(ComplexClass, prefix)
Export all the types of the graph.
- #define [EXPORT_COMPLEX_VERTICES](#)(ComplexClass)
Export the types of the vertices of the complex, i.e.
- #define [EXPORT_COMPLEX_VERTICES_PREFIX](#)(ComplexClass, prefix)
Export the types of the vertices of the complex, i.e.
- #define [IMPORT_COMPLEX_EDGES](#)(ComplexClass)
Import the edges types for another template where the complex class is not completely instantiated.
- #define [IMPORT_COMPLEX_GRAPHS](#)(ComplexClass)
Import the graphs types for another template where the complex class is not completely instantiated.
- #define [IMPORT_COMPLEX_MODEL](#)(ComplexClass, cplx) typename
ComplexClass::model_t &model = *(cplx).model;

Import the model from the instance `cplx` of the complex of type `ComplexClass`.

- #define **IMPORT_COMPLEX_TYPES**(ComplexClass)
Import all the types of the complex for another template where the complex class is not completely instantiated.
- #define **IMPORT_COMPLEX_VERTICES**(ComplexClass)
Import the vertices types for another template where the complex class is not completely instantiated.
- #define **MANDATORY_COMPLEX_TEMPLATE_ARGS** Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent,

Typedefs

Types defined by **EXPORT_COMPLEX_TYPES(VVComplex)**

- typedef VVComplex::cell **vvcomplex::cell**
Type of a cell.
- typedef VVComplex::cell_edge **vvcomplex::cell_edge**
Type of an edge in the cell graph.
- typedef VVComplex::cell_graph **vvcomplex::cell_graph**
Type of the cell graph.
- typedef VVComplex::cell_junction_edge **vvcomplex::cell_junction_edge**
Type of an edge from a cell to a junction.
- typedef VVComplex::complex_graph **vvcomplex::complex_graph**
Type of the complex graph (i.e. linking cells to junctions and vice-versa).
- typedef VVComplex::const_cell_edge **vvcomplex::const_cell_edge**
Type of an edge in the cell graph.
- typedef VVComplex::const_cell_junction_edge **vvcomplex::const_cell_junction_edge**
Type of an edge from a cell to a junction.
- typedef VVComplex::const_junction_cell_edge **vvcomplex::const_junction_cell_edge**
Type of an edge from a junction to a cell.
- typedef VVComplex::const_wall **vvcomplex::const_wall**
Type of a wall (i.e. an edge in the wall graph).
- typedef VVComplex::junction **vvcomplex::junction**
Type of a junction.

- typedef VVComplex::junction_cell_edge [vvcomplex::junction_cell_edge](#)
Type of an edge from a junction to a cell.
- typedef VVComplex::wall [vvcomplex::wall](#)
Type of a wall (i.e. an edge in the wall graph).
- typedef VVComplex::wall_graph [vvcomplex::wall_graph](#)
Type of the wall graph.

Functions

- template<typename Complex >
std::vector< typename Complex::junction > [vvcomplex::contour](#) (const Complex &T)
 - template<typename CplxGraph >
void [vvcomplex::create_cplxgraph_methods](#) (CplxGraph &G)
 - template<class VVComplex >
void [vvcomplex::FindCenter](#) (const typename VVComplex::cell &c, VVComplex &T)
Compute the position of the center of a cell.
 - template<class VVComplex >
[Point3d](#) [vvcomplex::findCenter](#) (const typename VVComplex::cell &c, VVComplex &T)
Return the position of the center of a cell.
 - template<class VVComplex >
DivisionData< typename VVComplex::junction_content > [vvcomplex::findDivisionPoints](#) (const typename VVComplex::cell &c, VVComplex &T, const InModelDivisionParam ¶m)
Implementation of the in model division scheme.
 - template<typename VVComplex >
void [vvcomplex::FindOppositeWall](#) (const typename VVComplex::cell &c, DivisionData< typename VVComplex::junction_content > &result, VVComplex &T, double cellWallMin, bool strictCellWallMin)
Find the wall opposite to the point already selected.
 - bool [vvcomplex::FindWallMin](#) ([Point3d](#) &v, const [Point3d](#) &v1, const [Point3d](#) &v2, double mw, double *displacement=0)
Move point v to avoid segment borders.
 - template<typename VVComplex >
void [vvcomplex::testDivisionOnVertices](#) (const typename VVComplex::cell &c, DivisionData< typename VVComplex::junction_content > &result, VVComplex &T, double epsilon)
Test if the division point in result is close enough to one of the extremum.

18.6.1 Detailed Description

Definition of a 2D cell complex.

Definition in file [complex.h](#).

18.6.2 Define Documentation

18.6.2.1 #define EXPORT_COMPLEX_EDGES(ComplexClass)

Value:

```
typedef ComplexClass::cell_edge cell_edge; \
    typedef ComplexClass::wall wall; \
    typedef ComplexClass::cell_junction_edge cell_junction_edge; \
    typedef ComplexClass::junction_cell_edge junction_cell_edge; \
    typedef ComplexClass::const_cell_edge const_cell_edge; \
    typedef ComplexClass::const_wall const_wall; \
    typedef ComplexClass::const_cell_junction_edge const_cell_junction_edge; \
    typedef ComplexClass::const_junction_cell_edge const_junction_cell_edge
```

Export the types of the edges of the complex, i.e.

cell_edge, wall, cell_junction_edge and junction_cell_edge. Equivalent to:

```
typedef ComplexClass::cell_edge cell_edge;
typedef ComplexClass::wall wall;
typedef ComplexClass::cell_junction_edge cell_junction_edge;
typedef ComplexClass::junction_cell_edge junction_cell_edge;
typedef ComplexClass::const_cell_edge const_cell_edge;
typedef ComplexClass::const_wall const_wall;
typedef ComplexClass::const_cell_junction_edge const_cell_junction_edge;
typedef ComplexClass::const_junction_cell_edge const_junction_cell_edge
```

Definition at line 464 of file complex.h.

18.6.2.2 #define EXPORT_COMPLEX_EDGES_PREFIX(ComplexClass, prefix)

Value:

```
typedef ComplexClass::cell_edge prefix##_cell_edge; \
    typedef ComplexClass::wall prefix##_wall; \
    typedef ComplexClass::cell_junction_edge prefix##_cell_junction_edge; \
    typedef ComplexClass::junction_cell_edge prefix##_junction_cell_edge; \
    typedef ComplexClass::const_cell_edge prefix##_const_cell_edge; \
    typedef ComplexClass::const_wall prefix##_const_wall; \
    typedef ComplexClass::const_cell_junction_edge prefix##_const_cell_junction_edi; \
    typedef ComplexClass::const_junction_cell_edge prefix##_const_junction_cell_edi
```

Export the types of the edges of the complex, i.e.

cell_edge, wall, cell_junction_edge and junction_cell_edge. Equivalent to:

```
typedef ComplexClass::cell_edge prefix_cell_edge;
typedef ComplexClass::wall prefix_wall;
typedef ComplexClass::cell_junction_edge prefix_cell_junction_edge;
typedef ComplexClass::junction_cell_edge prefix_junction_cell_edge;
typedef ComplexClass::const_cell_edge prefix_const_cell_edge;
typedef ComplexClass::const_wall prefix_const_wall;
typedef ComplexClass::const_cell_junction_edge prefix_const_cell_junction_edge;
typedef ComplexClass::const_junction_cell_edge prefix_const_junction_cell_edge
```

Definition at line 543 of file complex.h.

18.6.2.3 #define EXPORT_COMPLEX_GRAPHS(ComplexClass)

Value:

```
typedef ComplexClass::complex_graph complex_graph; \
    typedef ComplexClass::cell_graph cell_graph; \
    typedef ComplexClass::wall_graph wall_graph
```

Export the types of the graphs of the complex, i.e.
complex_graph, cell_graph, wall_graph. Equivalent to:

```
typedef ComplexClass::complex_graph complex_graph;
typedef ComplexClass::cell_graph cell_graph;
typedef ComplexClass::wall_graph wall_graph
```

Definition at line 487 of file complex.h.

18.6.2.4 #define EXPORT_COMPLEX_GRAPHS_PREFIX(ComplexClass, prefix)

Value:

```
typedef ComplexClass::complex_graph prefix##_complex_graph; \
    typedef ComplexClass::cell_graph prefix##_cell_graph; \
    typedef ComplexClass::wall_graph prefix##_wall_graph
```

Export the types of the graphs of the complex, i.e.
complex_graph, cell_graph, wall_graph.

```
typedef ComplexClass::complex_graph prefix_complex_graph;
typedef ComplexClass::cell_graph prefix_cell_graph;
typedef ComplexClass::wall_graph prefix_wall_graph
```

Definition at line 566 of file complex.h.

18.6.2.5 #define EXPORT_COMPLEX_TYPES(ComplexClass)

Value:

```
EXPORT_COMPLEX_VERTICES(ComplexClass); \
    EXPORT_COMPLEX_EDGES(ComplexClass); \
    EXPORT_COMPLEX_GRAPHS(ComplexClass)
```

Export all the types of the graph.

Equivalent to

```
EXPORT_COMPLEX_VERTICES (ComplexClass);  
EXPORT_COMPLEX_EDGES (ComplexClass);  
EXPORT_COMPLEX_GRAPHES (ComplexClass);
```

Definition at line 504 of file complex.h.

18.6.2.6 **#define EXPORT_COMPLEX_TYPES_PREFIX(ComplexClass, prefix)**

Value:

```
EXPORT_COMPLEX_VERTICES_PREFIX (ComplexClass, prefix); \  
EXPORT_COMPLEX_EDGES_PREFIX (ComplexClass, prefix); \  
EXPORT_COMPLEX_GRAPHES_PREFIX (ComplexClass, prefix)
```

Export all the types of the graph.

Equivalent to

```
EXPORT_COMPLEX_VERTICES_PREFIX (ComplexClass, prefix);  
EXPORT_COMPLEX_EDGES_PREFIX (ComplexClass, prefix);  
EXPORT_COMPLEX_GRAPHES_PREFIX (ComplexClass, prefix);
```

Definition at line 583 of file complex.h.

18.6.2.7 **#define EXPORT_COMPLEX_VERTICES(ComplexClass)**

Value:

```
typedef ComplexClass::cell cell; \  
typedef ComplexClass::junction junction
```

Export the types of the vertices of the complex, i.e.

the cell and junction. Equivalent to:

```
typedef ComplexClass::cell cell;  
typedef ComplexClass::junction junction
```

Definition at line 442 of file complex.h.

18.6.2.8 **#define EXPORT_COMPLEX_VERTICES_PREFIX(ComplexClass, prefix)**

Value:

```
typedef ComplexClass::cell prefix##_cell; \  
typedef ComplexClass::junction prefix##_junction
```

Export the types of the vertices of the complex, i.e.
the cell and junction, but prefix the defined type. Equivalent to:

```
typedef ComplexClass::cell prefix_cell;
typedef ComplexClass::junction prefix_junction
```

Definition at line 521 of file complex.h.

18.6.2.9 #define IMPORT_COMPLEX_EDGES(ComplexClass)

Value:

```
typedef typename ComplexClass::cell_edge cell_edge; \
typedef typename ComplexClass::wall wall; \
typedef typename ComplexClass::cell_junction_edge cell_junction_edge; \
typedef typename ComplexClass::junction_cell_edge junction_cell_edge; \
typedef typename ComplexClass::const_cell_edge const_cell_edge; \
typedef typename ComplexClass::const_wall const_wall; \
typedef typename ComplexClass::const_cell_junction_edge \
    const_cell_junction_edge; \
typedef typename ComplexClass::const_junction_cell_edge \
    const_junction_cell_edge
```

Import the edges types for another template where the complex class is not completely instantiated.

Definition at line 608 of file complex.h.

18.6.2.10 #define IMPORT_COMPLEX_GRAPHS(ComplexClass)

Value:

```
typedef typename ComplexClass::complex_graph complex_graph; \
typedef typename ComplexClass::cell_graph cell_graph; \
typedef typename ComplexClass::wall_graph wall_graph
```

Import the graphs types for another template where the complex class is not completely instantiated.

Definition at line 626 of file complex.h.

18.6.2.11 #define IMPORT_COMPLEX_MODEL(ComplexClass, cplx) typename ComplexClass::model_t &model = *(cplx).model;

Import the model from the instance `cplx` of the complex of type `ComplexClass`.

As a result, a variable named `model` is defined as a reference to the model.

Definition at line 640 of file complex.h.

Referenced by `vvcomplex::FindCenter()`, `vvcomplex::findCenter()`, `tissue::findDivisionPoints()`, `vvcomplex::findDivisionPoints()`, `cell_system::findDivisionPoints()`, `vvcomplex::FindOppositeWall()`, `vvcomplex::testDivisionOnVertices()`, and `cell_system::volumeLeftCell()`.

18.6.2.12 #define IMPORT_COMPLEX_TYPES(ComplexClass)

Value:

```
IMPORT_COMPLEX_VERTICES(ComplexClass); \
IMPORT_COMPLEX_EDGES(ComplexClass); \
IMPORT_COMPLEX_GRAPHS(ComplexClass)
```

Import all the types of the complex for another template where the complex class is not completely instantiated.

Definition at line 651 of file complex.h.

Referenced by `complex_factory::hex_grid()`, `complex_factory::objreader()`, and `complex_factory::square_grid()`.

18.6.2.13 #define IMPORT_COMPLEX_VERTICES(ComplexClass)

Value:

```
typedef typename ComplexClass::cell cell; \
typedef typename ComplexClass::junction junction
```

Import the vertices types for another template where the complex class is not completely instantiated.

Definition at line 596 of file complex.h.

Referenced by `vvcomplex::findCenter()`, `tissue::findDivisionPoints()`, `cell_system::findDivisionPoints()`, `vvcomplex::FindOppositeWall()`, and `cell_system::volumeLeftCell()`.

18.7 vvelib/storage/complex.h File Reference

```
#include <config.h>
#include <storage/storage.h>
#include <storage/graph.h>
#include <algorithms/complex.h>
#include <iostream>
#include <storage/vector.h>
```

Namespaces

- namespace [vvcomplex](#)

Definition of classes and functions for 2d cell complexes.

Functions

- `template<typename V1 , typename V2 , typename E1 , typename E2 , bool compact>`
`bool vvcomplex::serialization (storage::VVEStorage &store,`
`VVComplexGraph< V1, V2, E1, E2, compact > &G)`

18.7.1 Detailed Description

Definition in file [complex.h](#).

18.8 vvelib/algorithms/draw_graphs.h File Reference

This file contains algorithms to draw graphs.

```
#include <config.h>
#include <util/gl.h>
#include <geometry/geometry.h>
```

Namespaces

- namespace [algorithms](#)
Namespace containing various useful algorithms.

Functions

- `template<typename GraphType , typename Model >`
`void algorithms::drawSymetricVVGraph (Model *model, const GraphType &G)`
This function draws a graph using lines between the vertices.

18.8.1 Detailed Description

This file contains algorithms to draw graphs.

Definition in file [draw_graphs.h](#).

18.9 vvelib/algorithms/graph.h File Reference

Implement common algorithms on graphs.

```
#include <config.h>
#include <graph/vvgraph.h>
#include <set>
#include <util/unorderedset.h>
#include <utility>
```

Namespaces

- namespace [algorithms](#)
Namespace containing various useful algorithms.

Functions

- `template<typename VertexContent , typename EdgeContent , bool compact, typename Model , typename tag_t >`
`void algorithms::shortest_paths_Dijkstra (const std::unordered_set< Vertex< VertexContent > > &srcs, const graph::VVGraph< VertexContent, EdgeContent, compact > &m, Model &model, const tag_t &t)`
- `template<typename VertexContent , typename EdgeContent , bool compact, typename Model >`
`void algorithms::shortest_paths_Dijkstra (const Vertex< VertexContent > &src, const graph::VVGraph< VertexContent, EdgeContent, compact > &m, const Model &model)`
- `template<typename VertexContent , typename EdgeContent , bool compact, typename Model , type-`
`name tag_t >`
`void algorithms::shortest_paths_Dijkstra (const Vertex< VertexContent > &src, const graph::VVGraph< VertexContent, EdgeContent, compact > &m, Model &model, const tag_t &t)`
- `template<typename VVGraph , typename Model >`
`void algorithms::shortest_paths_FloydWarshall (const VVGraph &m, const Model &model)`
Compute the shortest path for all pair of vertices in the graph.
- `template<typename VertexContent , typename EdgeContent , bool compact, typename Model , type-`
`name tag_t >`
`void algorithms::shortest_paths_FloydWarshall (const graph::VVGraph< VertexContent, EdgeContent, compact > &m, Model &model, const tag_t &t)`

18.9.1 Detailed Description

Implement common algorithms on graphs.

Definition in file [graph.h](#).

18.10 vvelib/storage/graph.h File Reference

```
#include <config.h>
#include <storage/storage.h>
#include <graph/vvgraph.h>
#include <graph/vvbigraph.h>
#include <iostream>
```

Namespaces

- namespace [graph](#)
Contains all the classes related to the graphs.

Functions

- `template<typename T >`
`Edge< T > graph::create_object (Edge< T > const &e)`
- `template<typename T >`
`Vertex< T > graph::create_object (Vertex< T > const &v)`
- `template<typename V1 , typename V2 , typename E1 , typename E2 , bool compact>`
`bool graph::serialization (storage::VVEStorage &store, VVBigraph< V1, V2, E1, E2, compact > &G)`
- `template<typename V , typename E , bool compact>`
`bool graph::serialization (storage::VVEStorage &store, VVGraph< V, E, compact > &G)`
- `template<typename T >`
`bool graph::serialization (storage::VVEStorage &store, Edge< T > &e)`
- `template<typename T >`
`bool graph::serialization (storage::VVEStorage &store, Vertex< T > &v)`

18.10.1 Detailed Description

Definition in file [graph.h](#).

18.11 vvelib/algorithms/insert.h File Reference

Defines the [algorithms::Insert](#) class template.

```
#include <config.h>
#include <utility>
#include <util/static_assert.h>
#include <graph/vvgraph.h>
```

Classes

- class [algorithms::Insert< vvgraph, do_checks >](#)
Insert a new vertex on an edge.

Namespaces

- namespace [algorithms](#)
Namespace containing various useful algorithms.

Functions

- template<class VertexContent , class EdgeContent , bool compact>
const [graph::Vertex](#)< VertexContent > & **algorithms::insert** (const [graph::Vertex](#)< VertexContent > &a, const [graph::Vertex](#)< VertexContent > &b, [graph::VVGraph](#)< VertexContent, EdgeContent, compact > &S, const [graph::Vertex](#)< VertexContent > &x=[graph::Vertex](#)< VertexContent >(0))
- template<class Graph >
[Graph::edge_t](#) **algorithms::insertAfter** (const typename [Graph::vertex_t](#) &v, const typename [Graph::vertex_t](#) &ref, const typename [Graph::vertex_t](#) &nv, [Graph](#) &S)
Splice nv after ref in v if ref is not null.
- template<class Graph >
[Graph::edge_t](#) **algorithms::insertBefore** (const typename [Graph::vertex_t](#) &v, const typename [Graph::vertex_t](#) &ref, const typename [Graph::vertex_t](#) &nv, [Graph](#) &S)
Splice nv before ref in v if ref is not null.

18.11.1 Detailed Description

Defines the [algorithms::Insert](#) class template.

Definition in file [insert.h](#).

18.12 vvelib/algorithms/parallel.h File Reference

Defines the help class for parallel execution.

```
#include <config.h>
#include <cstdlib>
#include <algorithms/parallel_impl.h>
```

Classes

- class [parallel::ThreadEval](#)
Base class for function evaluated in a thread.

Namespaces

- namespace [parallel](#)
Define a thread pool for multi-threading computation.

Defines

- #define **FIRST_ARG**(F) typename F::first_argument_type
- #define **FOURTH_ARG**(F) typename F::fourth_argument_type
- #define **ONLY_ARG**(F) typename F::argument_type
- #define **SECOND_ARG**(F) typename F::second_argument_type
- #define **THIRD_ARG**(F) typename F::third_argument_type

Functions

- template<class Arg1 , class Arg2 , class Arg3 , class Arg4 >
FreeFunction4< Arg1, Arg2, Arg3, Arg4 > **parallel::freeFunction**
(void(*f)(Arg1, Arg2, Arg3, Arg4))
- template<class Arg1 , class Arg2 , class Arg3 >
FreeFunction3< Arg1, Arg2, Arg3 > **parallel::freeFunction** (void(*f)(Arg1,
Arg2, Arg3))
- template<class Arg1 , class Arg2 >
FreeFunction2< Arg1, Arg2 > **parallel::freeFunction** (void(*f)(Arg1, Arg2))
- template<class Arg1 >
FreeFunction1< Arg1 > **parallel::freeFunction** (void(*f)(Arg1))
- template<class Class , typename Arg1 , typename Arg2 , typename Arg3 >
MemberFunction3< Class, Arg1, Arg2, Arg3 > **parallel::memberFunction**
(void(Class::*m)(Arg1, Arg2, Arg3))

- `template<class Class , typename Arg1 , typename Arg2 >`
`MemberFunction2< Class, Arg1, Arg2 > parallel::memberFunction`
`(void(Class::*m)(Arg1, Arg2))`
- `template<class Class , typename Arg >`
`MemberFunction1< Class, Arg > parallel::memberFunction`
`(void(Class::*m)(Arg))`
- `template<class Class >`
`MemberFunction0< Class > parallel::memberFunction (void(Class::*m)())`
- `template<typename Function >`
`void parallel::threadEval (const Function &fct, FIRST_ARG(Function) a1,`
`SECOND_ARG(Function) a2, THIRD_ARG(Function) a3, FOURTH_`
`ARG(Function) a4)`
- `template<typename Function >`
`void parallel::threadEval (const Function &fct, FIRST_ARG(Function) a1,`
`SECOND_ARG(Function) a2, THIRD_ARG(Function) a3)`
- `template<typename Function >`
`void parallel::threadEval (const Function &fct, FIRST_ARG(Function) a1,`
`SECOND_ARG(Function) a2)`
- `template<typename Function >`
`void parallel::threadEval (const Function &fct, ONLY_ARG(Function) a)`
- `template<typename Function >`
`void parallel::threadEval (const Function &fct)`
Evaluate a function without arguments using the thread pool.
- `void parallel::threadJoin ()`

18.12.1 Detailed Description

Defines the help class for parallel execution.

Definition in file [parallel.h](#).

18.13 vvelib/algorithms/solver.h File Reference

Defines the solver namespace.

```
#include <config.h>
#include <cmath>
#include <iostream>
#include <string>
#include <util/vector.h>
#include <util/matrix.h>
#include <algorithms/parallel.h>
```

Classes

- class [solver::Solver< nb_vars, identifier >](#)
Implement a set of solvers for ODE on a graph.

Namespaces

- namespace [solver](#)
Implement a generic solver of ODE on a graph.

Defines

- #define **VERTEX** [graph::Vertex<VertexContent>](#)

Enumerations

- enum [solver::SolvingMethod](#) {
[solver::Euler](#), [solver::AdaptiveEuler](#), [solver::Fixedpoint](#), [solver::Midpoint](#),
[solver::RungeKutta](#), [solver::AdaptiveRungeKutta](#),
[solver::AdaptiveCrankNicholson](#) }
Available solving methods.
- enum [solver::ToleranceType](#) { [solver::MAX_COMPONENT](#), [solver::MEAN_COMPONENT](#) }
Type of the tolerances.

Functions

- `std::ostream & solver::operator<< (std::ostream &s, const ToleranceType &m)`
- `std::ostream & solver::operator<< (std::ostream &s, const SolvingMethod &m)`
- `QTextStream & solver::operator<< (QTextStream &s, const SolvingMethod &m)`
- `std::istream & solver::operator>> (std::istream &s, ToleranceType &m)`
- `std::istream & solver::operator>> (std::istream &s, SolvingMethod &m)`
- `QTextStream & solver::operator>> (QTextStream &s, SolvingMethod &m)`

18.13.1 Detailed Description

Defines the solver namespace.

Definition in file [solver.h](#).

18.14 vvelib/algorithms/split.h File Reference

Defines the [algorithms::split](#) function.

```
#include <config.h>
```

Namespaces

- namespace [algorithms](#)
Namespace containing various useful algorithms.

Functions

- `template<class vvgraph >`
`vvgraph::vertex_t algorithms::split (const typename vvgraph::vertex_t &v, const
typename vvgraph::vertex_t &n1, const typename vvgraph::vertex_t &n2, vv-
graph &S)`
*Split the vertex v in two, attaching all the neighbors of v from $n1$ to $n2$ (included) to
the new vertex.*

18.14.1 Detailed Description

Defines the [algorithms::split](#) function.

Definition in file [split.h](#).

18.15 vvelib/algorithms/tissue.h File Reference

Definition of the tissue algorithm.

```
#include <config.h>
#include <algorithms/complex.h>
#include <geometry/geometry.h>
```

Classes

- struct [tissue::CellPinchingParams](#)
Parameters of the cell pinching algorithm.
- struct [tissue::ClosestMidAlgoParams](#)
Parameters for the closest mid.
- struct [tissue::ClosestWallAlgoParams](#)
Parameters for the closest wall algorithm.
- struct [tissue::ShortWallAlgoParams](#)
Parameters for the shortest wall algorithm.
- class [tissue::Tissue< Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, Leaf-Class >](#)
Class handling the development and representation of a cell tissue.

Namespaces

- namespace [tissue](#)
The tissue module handle cell division in a cell tissue.

Typedefs

- typedef [Tissue::cell_graph](#) [tissue::cell_graph](#)
Documentation alias for the type of the cell graph.
- typedef [Tissue::tissue_graph](#) [tissue::tissue_graph](#)
Documentation alias for the type of the tissue graph.
- typedef [Tissue::vertex](#) [tissue::vertex](#)
Documentation alias for the type of a graph vertex.

Enumerations

- enum [tissue::CELL_DIVISION_ALGORITHM](#) { CLOSEST_MID, SHORT_WALL, CLOSEST_WALL }

Enumeration of the provided division algorithms.

Functions

- template<class Complex >
void [tissue::cellPinching](#) (const typename Complex::cell &c, Complex &T, DivisionData< typename Complex::junction_content > &data, const CellPinchingParams ¶ms)

Pinching algorithm.

- template<class Complex >
DivisionData< typename Complex::junction_content > [tissue::findDivisionPoints](#) (const typename Complex::cell &c, Complex &T, const ClosestWallAlgoParams ¶ms)

Implementation of the closest wall algorithm.

- template<class Complex >
DivisionData< typename Complex::junction_content > [tissue::findDivisionPoints](#) (const typename Complex::cell &c, Complex &T, const ClosestMidAlgoParams ¶ms)

Implementation of the closest mid algorithm.

- template<class Complex >
DivisionData< typename Complex::junction_content > [tissue::findDivisionPoints](#) (const typename Complex::cell &c, Complex &T, const ShortWallAlgoParams ¶ms)

Implementation of the shortest wall algorithm.

18.15.1 Detailed Description

Definition of the tissue algorithm.

Definition in file [tissue.h](#).

18.16 vvelib/algorithms/triangle_growth.h File Reference

This file contains the classes needed to describe the growth of a tissue from a triangular mesh growing through time.

```
#include <config.h>
#include <geometry/geometry.h>
#include <vector>
#include <algorithms/complex.h>
#include <QString>
#include <storage/storage.h>
#include <util/unorderedset.h>
#include <util/unorderedmap.h>
#include <util/assert.h>
#include <QTextStream>
```

Classes

- struct [algorithms::TriangleSurface::Cell](#)
Type of a cell, i.e.
- struct [algorithms::TriangleSurface::Position](#)
Structure describing the position of a point on a triangulated surface, relatively to this surface.
- class [algorithms::TriangleGrowth](#)
Growth description using a set of triangles moving.
- class [algorithms::TriangleSurface](#)
Class representing a triangulated surface.
- struct [algorithms::TriangleSurface::Vertex](#)
Type of a vertex, i.e.

Namespaces

- namespace [algorithms](#)
Namespace containing various useful algorithms.

18.16.1 Detailed Description

This file contains the classes needed to describe the growth of a tissue from a triangular mesh growing through time.

Definition in file [triangle_growth.h](#).

18.17 vvelib/bspline_tissue_model.h File Reference

This files include the bspline [tissue](#) model helper.

```
#include <vve.h>
#include <config.h>
#include <model.h>
#include <viewer.h>
#include <util/forall.h>
#include <algorithms/tissue.h>
#include <geometry/geometry.h>
#include <util/parms.h>
#include <util/palette.h>
#include <util/vector.h>
#include <util/watchdog.h>
#include <vector>
#include <string>
#include <util/matrix.h>
#include <qgl.h>
#include <qstring.h>
#include "bsurface.h"
#include <iostream>
```

Classes

- struct [bspline_tissue_model::TissueModel](#)< [RealModel](#), [TissueClass](#) >::[CompareSize](#)
Operator class to compare the size of cells.
- class [bspline_tissue_model::TissueModel](#)< [RealModel](#), [TissueClass](#) >
Base class for the bspline tissue model helper.

Namespaces

- namespace [bspline_tissue_model](#)
Namespace containing the base class for bspline tissue model helper.

Defines

- `#define CellAttributes`
- `#define CellEdgeAttributes`
- `#define CellJunctionEdgeAttributes`
- `#define EndModel }; DEFINE_MODEL(ModelClass);`
Macro defining the end of the model.

- `#define JunctionAttributes`
- `#define JunctionCellEdgeAttributes`
- `#define ModelInit`
- `#define StartModel`
Macro defining the start of the model.

- `#define WallAttributes`

Typedefs

- `typedef util::Palette::Color Color`
Typedef to easy the use of colors.

Variables

- `const double bspline_tissue_model::epsilon = 0.0001`
Relative error for geometry.

18.17.1 Detailed Description

This files include the bspline `tissue` model helper. This helper define a tissue whose growth is driven by a set of bspline surfaces defining the geometry at key-points.

The namespace `bspline_tissue_model` contains the base class for this helper.

To use the helper, simply include `<bspline_tissue_model.h>`. Any extra attributes for vertices or edges should be defined before including this file.

Once included, the model starts with

```
StartModel
```

and ends with

```
EndModel
```

The tissue defined in the model uses cells which contains these attributes:

```
double area;           // Area of the cell
Point3d pos;           // Position of the cell
Point3d normal;        // Normal to the surface
Point2d uv;            // Local coordinate of the cell on the bspline
double value;          // Value used to draw the cell
```

and junctions which contains these attributes:

```
Point3d pos;           // Position of the junction
Point3d normal;        // Normal to the surface
Point2d uv;            // Local coordinate of the junction on the bspline
```

In addition, the user can add its own attributes by defining the macro `VertexAttributes` **before** including this file:

```
#define CellAttribute \
    double attr1; \
    int attr2; \
    bool attr3;
#define JunctionAttribute \
    double attr1; \
    bool attr2; \
    int attr3;

#include <bspline_tissue_model.h>
```

Note

As this is a macro, to span over many lines you need to end the inner lines with a backslash.

Similarly, the user can add attributes to the cell and tissue edges by defining the macros `CellEdgeAttributes`, `WallAttributes`, `CellJunctionEdgeAttributes` and `JunctionCellEdgeAttributes`.

Within the model class, named `ModelClass`, all the methods and variable of the class `bspline_tissue_model::TissueModel` are available. Reference to the parent class can be done using the `ParentClass` typedef.

For example, if yo want to augment the drawing method, you might write:

```
void draw(Viewer* viewer)
{
    ParentClass::draw(viewer);
    // my other commands
}
```

As a small example, here is a code that creates a single cell and change its color when the user click on it.

```
#define CellAttributes \
    bool clicked; // Is the current cell clicked?

#include <bspline_tissue_model.h>

StartModel

// Define
```

```
int clickedColor, unclickedColor;
double dt;          // How much mecanic advance at each step
double maxArea;     // Maximum area of a cell

// Read the extra parameter
void readParam(util::Parms& parms)
{
    parms("Main", "dt", dt);
    parms("Main", "MaxArea", maxArea);
    parms("View", "ClickedColor", clickedColor);
    parms("View", "UnclickedColor", unclickedColor);
}

// Initialize the tissue
void initialize()
{
    // First, create the cell
    initTissue();
    // Then, initialise the content of the cells
    forall(const cell& c, T.C)
        c->clicked = false;
}

// Update the status of the cell after user interaction
void postSelection(const QPoint&, Viewer* viewer)
{
    const cell& c = cellFromId(viewer->selectedName());
    // If a cell was selected
    if(c)
    {
        c->clicked ^= true; // Invert its state
    }
}

// Main loop: tissue growth and cell division
void step()
{
    // Change the current time, then update the vertices positions and cell areas

    time += dt;
    updatePositions();
    updateCellsArea();
    ordered_cells_t ordered_divide;

    // Find the cells to be divided, order them and divide them
    forall(const cell& c, T.C)
    {
        vvcomplex::FindCenter(c, T);
        if(c->area > maxArea)
        {
            ordered_divide.insert(c);
        }
    }

    // How to divide
    forall(const cell& c, ordered_divide)
    {
        T.divideCell(c);
    }
}

// Define the color depending on the clicked state
```



```

Color getCellColor(const cell& c)
{
    if(c->clicked)
        return palette.getColor(clickedColor);
    else
        return palette.getColor(unclickedColor);
}

// Define the color depending on the clicked state
Color getCellCenterColor(const cell& c)
{
    if(c->clicked)
        return palette.getColor(clickedColor);
    else
        return palette.getColor(unclickedColor);
}

// When a cell is clicked, the property is inherited by the children
void updateFromOld(const cell& cl, const cell& cr, const cell& c,
                  const Tissue::division_data&, Tissue&)
{
    cl->clicked = cr->clicked = c->clicked;
}

EndModel

```

Definition in file [bspline_tissue_model.h](#).

18.17.2 Define Documentation

18.17.2.1 `#define EndModel` ; `DEFINE_MODEL(ModelClass)`;

Macro defining the end of the model.

Definition at line 651 of file [bspline_tissue_model.h](#).

18.17.2.2 `#define StartModel`

Macro defining the start of the model.

Definition at line 593 of file [bspline_tissue_model.h](#).

18.17.3 Typedef Documentation

18.17.3.1 `typedef util::Palette::Color Color`

Typedef to easy the use of colors.

Definition at line 193 of file [bspline_tissue_model.h](#).

18.18 vvelib/factory/complex_grid.h File Reference

Functions to initialise a complex as a regular grid.

```
#include <config.h>
#include <util/vector.h>
#include <factory/grid.h>
#include <list>
#include <algorithms/insert.h>
#include <algorithms/complex.h>
#include <cassert>
#include <cmath>
```

Classes

- struct [complex_factory::HexFiller](#)< Complex, Model >
For internal use only!
- struct [complex_factory::SquareFiller](#)< Complex, Model >
For internal use only!

Namespaces

- namespace [complex_factory](#)
Contains functions to initialise a complex.

Defines

- #define **TF_BOTTOM_CORNER**(i, j) bottom_corner[(i)*(M+1)+(j)]
- #define **TF_CORNER**(i, j) corners[(i)*(M+1)+(j)]
- #define **TF_TOP_CORNER**(i, j) top_corner[(((i)>N)?N:(i))*(M+1)+(j)]

Typedefs

- typedef [util::Vector](#)< 2, size_t > **complex_factory::Point2u**
- typedef [util::Vector](#)< 3, double > **complex_factory::Point3d**

Functions

- `template<class Graph >`
`void complex_factory::connectJunction (const typename Graph::vertex_t &j,`
`const typename Graph::vertex_t &nj, const typename Graph::vertex_t &pj,`
`Graph &S, bool direct=true)`
Connect j to nj knowing that pj is the junction before j in the current cell.
- `template<typename Complex , class Model >`
`bool complex_factory::hex_grid (size_t N, size_t M, Complex &T, Model`
`&model, const Point3d &bottom_left=Point3d(0, 0, 0), const Point3d`
`&u=Point3d(1, 0, 0), const Point3d &v=Point3d(0, 1, 0))`
Create a hexagonal grid with N lines and M columns.
- `template<typename Complex , class Model >`
`bool complex_factory::square_grid (size_t N, size_t M, Complex &T, Model`
`&model, const Point3d &bottom_left=Point3d(0, 0, 0), const Point3d &shift_`
`right=Point3d(1, 0, 0), const Point3d &shift_up=Point3d(0, 1, 0))`
Create a square grid with N lines and M columns.
- `template<typename Complex , class Model >`
`SquareFiller< Complex, Model > complex_factory::squareFiller (const Point3d`
`&bottom_left, const Point3d &shift_right, const Point3d &shift_up, Complex`
`&T, Model &model)`
Create a square filler.

18.18.1 Detailed Description

Functions to initialise a complex as a regular grid.

Definition in file [complex_grid.h](#).

18.19 vvelib/factory/grid.h File Reference

Contains factories to generate grids.

```
#include <config.h>
```

```
#include <list>
```

Namespaces

- namespace [factory](#)
Contains predefined ways to initialize a graph.

Functions

- `template<typename Graph , class Model >`
`bool factory::hex_grid (size_t N, size_t M, Graph &G, Model &model, bool torus=false)`
Build a hexagonal grid with N lines and M columns.
- `template<typename Graph , class Model >`
`bool factory::square_grid (size_t N, size_t M, Graph &G, Model &model, bool torus=false, bool connect8=false)`
Create a square grid with N lines and M columns.

18.19.1 Detailed Description

Contains factories to generate grids.

Definition in file [grid.h](#).

18.20 vvelib/factory/objreader.h File Reference

Contains factories to generate a cell complex from an obj file.

```
#include <config.h>
#include <string>
#include <util/vector.h>
#include <QIODevice>
#include <QTextStream>
#include <QString>
#include <QFile>
#include <iostream>
#include <algorithms/complex.h>
```

Classes

- class [complex_factory::ObjReaderError](#)
Error object returned by the [complex_factory::objreader](#) functions.

Namespaces

- namespace [complex_factory](#)
Contains functions to initialise a complex.

Functions

- template<class CellComplex , class Model >
ObjReaderError [complex_factory::objreader](#) (std::string filename, CellComplex &T, [Model](#) &model)
Convenience methods that read a Wavefront OBJ file.
- template<class CellComplex , class Model >
ObjReaderError [complex_factory::objreader](#) (const **QString** &filename, CellComplex &T, [Model](#) &model)
Convenience methods that read a Wavefront OBJ file.
- template<class CellComplex , class Model >
ObjReaderError [complex_factory::objreader](#) (**QTextStream** &content, CellComplex &T, [Model](#) &model)
Read a Wavefront OBJ file describing a surface and return the complex representing it.

18.20.1 Detailed Description

Contains factories to generate a cell complex from an obj file.

Definition in file [objreader.h](#).

18.21 vvelib/geometry/area.h File Reference

Algorithms for areas of various shapes.

```
#include <config.h>
#include <geometry/geometry.h>
#include <vector>
```

Namespaces

- namespace [geometry](#)
Algorithmic geometry.

Functions

- Point2d [geometry::centroid](#) (const std::vector< Point2d > &polygon)
Center of mass of a 2D non-overlapping polygon.
- template<size_t N>
double [geometry::polygonArea](#) (const std::vector< [util::Vector](#)< N, double > > &polygon)
Area of a planar polygon.
- template<size_t N>
double [geometry::triangleArea](#) (const [util::Vector](#)< N, double > &a, const [util::Vector](#)< N, double > &b, const [util::Vector](#)< N, double > &c)
Area of a triangle in 2D or 3D.

18.21.1 Detailed Description

Algorithms for areas of various shapes.

Definition in file [area.h](#).

18.22 vvelib/geometry/coordinates.h File Reference

Includes functions to convert between different type of coordinate systems.

```
#include <config.h>
#include <geometry/geometry.h>
```

Namespaces

- namespace [geometry](#)
Algorithmic geometry.

Functions

- [Point3d geometry::barycentricToCartesian](#) (const [Point3d](#) &barycentric, const [Point3d](#) &p1, const [Point3d](#) &p2, const [Point3d](#) &p3)
Convert barycentric coordinates to cartesian.
- [Matrix3d geometry::barycentricToCartesian_matrix](#) (const [Point3d](#) &p1, const [Point3d](#) &p2, const [Point3d](#) &p3)
Convert cartesian coordinates to barycentric coordinates in a triangle.
- [Point3d geometry::cartesianToBarycentric](#) (const [Point3d](#) &pos, const [Point3d](#) &p1, const [Point3d](#) &p2, const [Point3d](#) &p3)
Convert cartesian coordinates to barycentric coordinates in a triangle.
- [Matrix3d geometry::cartesianToBarycentric_matrix](#) (const [Point3d](#) &p1, const [Point3d](#) &p2, const [Point3d](#) &p3, int &index_one)
Convert cartesian coordinates to barycentric coordinates in a triangle.

18.22.1 Detailed Description

Includes functions to convert between different type of coordinate systems.

Definition in file [coordinates.h](#).

18.23 vvelib/geometry/geometry.h File Reference

Common definitions and utilities for all geometry algorithms.

```
#include <config.h>
#include <util/vector.h>
#include <util/matrix.h>
#include <util/floats.h>
```

Namespaces

- namespace [geometry](#)
Algorithmic geometry.

Typedefs

- typedef [util::Matrix](#)< 2, 2, double > [geometry::Matrix2d](#)
Type of 2x2 matrix.
- typedef [util::Matrix](#)< 3, 3, double > [geometry::Matrix3d](#)
Type of 3x3 matrix.
- typedef [util::Matrix](#)< 4, 4, double > [geometry::Matrix4d](#)
Type of a 4x4 matrix.
- typedef [util::Vector](#)< 2, double > [geometry::Point2d](#)
Type of a 2D point.
- typedef [util::Vector](#)< 3, double > [geometry::Point3d](#)
Type of a 3D point.
- typedef [util::Vector](#)< 4, double > [geometry::Point4d](#)
Type of a 4D point.

18.23.1 Detailed Description

Common definitions and utilities for all geometry algorithms.

Definition in file [geometry.h](#).

18.24 vvelib/geometry/intersection.h File Reference

Algorithms to compute intersections.

```
#include <config.h>
#include <geometry/geometry.h>
#include <vector>
```

Namespaces

- namespace [geometry](#)
Algorithmic geometry.

Functions

- bool [geometry::lineLineIntersection](#) (Point2d &u, const Point2d &p1, const Point2d &u1, const Point2d &p2, const Point2d &u2)
Find if the segment [p1,p2] intersect the line going through p of normal n.
- bool [geometry::lineSegmentIntersection](#) (Point2d &u, const Point2d &p1, const Point2d &p2, const Point2d &p, const Point2d &n)
Find if the segment [p1,p2] intersect the line going through p of normal n.
- bool [geometry::lineTriangleIntersection](#) (Point3d &u, double &s, const Point3d &p1, const Point3d &p2, const Point3d &tr1, const Point3d &tr2, const Point3d &tr3)
Line-triangle intersection.
- bool [geometry::planeLineIntersection](#) (Point3d &u, double &s, const Point3d &p, const Point3d &n, const Point3d &u1, const Point3d &u2)
Plane-Line Intersection.
- template<typename PointContainer >
bool [geometry::pointInPolygon](#) (const Point2d &p, const PointContainer &polygon)
• bool [geometry::pointInPolygon](#) (const Point2d &p, const std::vector< Point2d > &polygon)
Find if the point p is inside the polygon using the winding number.
- bool [geometry::pointInTriangle](#) (Point3d &u, double &s, const Point3d &p, const Point3d &p1, const Point3d &p2, const Point3d &p3)
Point in triangle test (3D).
- bool [geometry::pointInTriangle](#) (const Point2d &p, const Point2d &p1, const Point2d &p2, const Point2d &p3)

Point in triangle test (2D).

- bool [geometry::segmentSegmentIntersection](#) (Point2d &u, const Point2d &p1, const Point2d &p2, const Point2d &q1, const Point2d &q2)

Find if two 2D segments intersect.

18.24.1 Detailed Description

Algorithms to compute intersections.

Definition in file [intersection.h](#).

18.25 vvelib/geometry/projection.h File Reference

Algorithms to project on lines/planes/.

```
#include <config.h>
#include <geometry/geometry.h>
```

Namespaces

- namespace [geometry](#)
Algorithmic geometry.

Functions

- `template<size_t N>`
`util::Vector< N, double > geometry::projectPointOnLine (const util::Vector< N, double > &pt, const util::Vector< N, double > &u1, const util::Vector< N, double > &u2, bool strict=false)`
Project a point on a line.
- `Point3d geometry::projectPointOnPlane (const Point3d &pos, const Point3d &p, const Point3d &n)`
Project a point into a plane defined by a point and a normal to the plane.
- `Point3d geometry::projectPointOnTriangle (const Point3d &pt, const Point3d &p1, const Point3d &p2, const Point3d &p3, double &s, bool *in_triangle=0)`
Project a point into a triangle.

18.25.1 Detailed Description

Algorithms to project on lines/planes/. ...

Definition in file [projection.h](#).

18.26 vvelib/geometry/quaternion.h File Reference

Implements the quaternion object.

```
#include <config.h>
```

```
#include <geometry/geometry.h>
```

Classes

- class [geometry::Quaternion](#)
Implements the quaternion operations.

Namespaces

- namespace [geometry](#)
Algorithmic geometry.
- namespace [std](#)
STL namespace.

Functions

- **Quaternion** [geometry::operator*](#) (const **Quaternion** &q, double s)
- **Quaternion** [geometry::operator*](#) (double s, const **Quaternion** &q)

18.26.1 Detailed Description

Implements the quaternion object.

Definition in file [quaternion.h](#).

18.27 vvelib/graph/edge.h File Reference

Define the [graph::Edge](#) class to be used with the [graph::VVGraph](#) class.

```
#include <config.h>
#include <util/unorderedmap.h>
#include <storage/fwd.h>
#include <QtCore/QHash>
```

Classes

- struct [graph::Arc](#)< [EdgeContent](#) >
Type of a undirected edge (or arc).
- class [graph::Edge](#)< [EdgeContent](#) >
Edge of a vv graph.

Namespaces

- namespace [graph](#)
Contains all the classes related to the graphs.

Typedefs

- typedef uintptr_t [graph::edge_identity_t](#)
Type of the identifier of an edge.

Functions

- template<typename T >
void [graph::copy_symmetric](#) (T &e2, const T &e1)
Default copy of the arc uses the = operator.
- template<typename EdgeContent >
uint [qHash](#) (const [graph::Edge](#)< EdgeContent > &e)

18.27.1 Detailed Description

Define the [graph::Edge](#) class to be used with the [graph::VVGraph](#) class.

Definition in file [edge.h](#).

18.28 vvelib/graph/vertex.h File Reference

This file contain the definition of the [graph::Vertex](#) class.

```
#include <QtCore/QHash>
#include <config.h>
#include <map>
#include <utility>
#include <util/unorderedmap.h>
#include <util/mangling.h>
#include <iostream>
#include <typeinfo>
#include <stdint.h>
#include <util/assert.h>
#include <storage/fwd.h>
```

Classes

- struct [graph::CountedContent](#)< Content >
Type of the reference counted content.
- class [graph::Vertex](#)< VertexContent >
Vertex of a vv graph.
- class [graph::WeakVertex](#)< VertexContent >
Weak pointer on a vertex.

Namespaces

- namespace [graph](#)
Contains all the classes related to the graphs.

Defines

- #define **TEMPLATE_VERTEX** typename VertexContent
- #define **VERTEX_ARGS** VertexContent
- #define **VVERTEX_NO_CACHE**

Typedefs

- typedef uintptr_t [graph::vertex_identity_t](#)

Type of the identifier of a vertex.

Functions

- template<TEMPLATE_VERTEX , typename charT >
std::basic_ostream< charT > & **graph::operator**<< (std::basic_ostream< charT > &ss, const Vertex< VERTEX_ARGS > &v)
- template<TEMPLATE_VERTEX >
uint **qHash** (const [graph::Vertex](#)< VERTEX_ARGS > &v)

18.28.1 Detailed Description

This file contain the definition of the [graph::Vertex](#) class. For now, the [graph::Vertex](#) class can only be used with the [graph::VVGraph](#) class.

Definition in file [vertex.h](#).

18.29 vvelib/graph/vvbigraph.h File Reference

Contain the definition of the VVBiGraph template class for bipartite graphs.

```
#include <config.h>
#include <graph/vertex.h>
#include <graph/edge.h>
#include <graph/vvgraph.h>
#include <util/tie.h>
#include <util/member_iterator.h>
#include <util/circ_iterator.h>
#include <util/set_vector.h>
#include <storage/fwd.h>
#include <utility>
#include <memory>
#include <algorithm>
#include <iterator>
#include <list>
#include <set>
#include <map>
#include <iostream>
```

Classes

- struct [graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::search_result_t< Neighborhood, Iterator >](#)
Type of the result of the search for a vertex in a neighborhood.
- struct [graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >::single_neighborhood_t< VertexSrcContent, VertexTgtContent, EdgeSrcContent >](#)
Type of the neighborhood of a vertex.
- class [graph::VVBiGraph< Vertex1Content, Vertex2Content, Edge1Content, Edge2Content_, compact >](#)
Class representing a VV graph.

Namespaces

- namespace [graph](#)

Contains all the classes related to the graphs.

- namespace [std](#)
STL namespace.

Defines

- #define **BIGRAPH_ARGS** Vertex1Content,Vertex2Content,Edge1Content,Edge2Content,compact
- #define **BIGRAPH_TEMPLATE** typename Vertex1Content, typename Vertex2Content, typename Edge1Content, typename Edge2Content, bool compact
- #define **VVBIGRAPH_COMPACT_EDGE** compact
- #define **VVBIGRAPH_COMPACT_VERTEX** compact

Functions

- template<typename BiGraph >
void **graph::create_bigraph_methods** (BiGraph &G)
- template<BIGRAPH_TEMPLATE >
void **std::swap** ([graph::VVBiGraph](#)< BIGRAPH_ARGS > &g1,
[graph::VVBiGraph](#)< BIGRAPH_ARGS > &g2)

18.29.1 Detailed Description

Contain the definition of the VVBiGraph template class for bipartite graphs.

Definition in file [vvbigraph.h](#).

18.30 vvelib/graph/vvgraph.h File Reference

Contain the definition of the VVGraph template class.

```
#include <config.h>
#include <graph/vertex.h>
#include <graph/edge.h>
#include <util/tie.h>
#include <util/member_iterator.h>
#include <util/circ_iterator.h>
#include <util/set_vector.h>
#include <util/range.h>
#include <storage/fwd.h>
#include <utility>
#include <memory>
#include <algorithm>
#include <iterator>
#include <list>
#include <vector>
#include <set>
#include <map>
#include <iostream>
```

Classes

- class [graph::_EmptyEdgeContent](#)
Empty class used as default for edge content.
- struct [graph::VVGraph< VertexContent, EdgeContent, compact >::neighbor_t](#)
Structure maintaining the data for a single neighbor.
- struct [graph::VVGraph< VertexContent, EdgeContent, compact >::search_result_t< Neighborhood, Iterator >](#)
Type of the result of the search for a vertex in a neighborhood.
- struct [graph::VVGraph< VertexContent, EdgeContent, compact >::single_neighborhood_t](#)
Type of the neighborhood of a vertex.

- class [graph::VVGraph](#)< VertexContent, EdgeContent, compact >
Class representing a VV graph.

Namespaces

- namespace [graph](#)
Contains all the classes related to the graphs.
- namespace [std](#)
STL namespace.

Defines

- #define **GRAPH_ARGS** VertexContent,EdgeContent,compact
- #define **GRAPH_TEMPLATE** typename VertexContent, typename EdgeContent, bool compact
- #define **VVGRAPH_COMPACT_EDGE** compact
- #define **VVGRAPH_COMPACT_VERTEX** compact

Functions

- template<typename Graph >
void **graph::create_graph_methods** (Graph &G)
- intptr_t **graph::getNextGraphId** ()
- template<GRAPH_TEMPLATE >
void [std::swap](#) ([graph::VVGraph](#)< GRAPH_ARGS > &g1, [graph::VVGraph](#)< GRAPH_ARGS > &g2)
Specialize the swap algorithm for two graphs.

18.30.1 Detailed Description

Contain the definition of the VVGraph template class.

Definition in file [vvgraph.h](#).

18.31 vvelib/model.h File Reference

Defines the [Model](#) class.

```
#include <config.h>
#include <string>
#include <set>
#include <QString>
#include <QStringList>
#include <QObject>
#include <QList>
#include <QMetaMethod>
#include <util/watchdog.h>
```

Classes

- class [Model](#)
Simulation class.

Namespaces

- namespace [storage](#)
Namespace containing all functions and classes related to VVE persistence.

18.31.1 Detailed Description

Defines the [Model](#) class.

Definition in file [model.h](#).

18.32 vvelib/shape/quadric.h File Reference

Include the definitions to draw spheres, cylinders and disks.

```
#include <config.h>
#include <geometry/geometry.h>
```

Namespaces

- namespace [shape](#)
Define a set of shape to be drawn using OpenGL.

Functions

- void [shape::cylinder](#) (const [Point3d](#) &base_center, const [Point3d](#) &axis, double radius_base, double radius_top, double length, int slices, int stacks, bool closed=true)
Draw a cylinder.
- void [shape::disk](#) (const [Point3d](#) ¢er, const [Point3d](#) &normal, double inner_radius, double outer_radius, int slices, int loops)
Draw a disk.
- void [shape::sphere](#) (const [Point3d](#) ¢er, double radius, int slices, int stacks)
Draw a sphere.

18.32.1 Detailed Description

Include the definitions to draw spheres, cylinders and disks.

Definition in file [quadric.h](#).

18.33 vvelib/storage/storage.h File Reference

This file contains the base class and functions to handle storage.

```
#include <config.h>
#include <QString>
#include <cstdint>
#include <string>
#include <vector>
#include <list>
#include <map>
#include <set>
#include <util/unorderedmap.h>
```

Classes

- class [storage::VVEStorage](#)
Abstract base class defining the interactions with the persistence module.

Namespaces

- namespace [storage](#)
Namespace containing all functions and classes related to VVE persistence.

Enumerations

- enum [storage::Options](#) { [storage::TypeChecking](#) }
Common options for reader/writer.
- enum [storage::STORAGE_ERROR](#) {
 [storage::NO__ERROR](#) = 0, [storage::IO_ERROR](#), [storage::BAD_CONTENT](#),
 [storage::TYPE_CHECK_ERROR](#),
 [storage::TYPE_CONVERSION_ERROR](#), [storage::NO_FIELD_ERROR](#),
 [storage::REFERENCE_ERROR](#), [storage::USER_ERROR](#),
 [storage::UNKNOWN_ERROR](#) }
Enumeration describing the basic errors.
- enum [storage::TypeCheckingOption](#) {
 [storage::TCO_Exact](#), [storage::TCO_Strict](#), [storage::TCO_Permissive](#),
 [storage::TCO_PermissiveNoWarning](#),

[storage::TCO_NoCheck](#) }

Enumeration describing how the type of the elements should be handled.

Functions

- `template<typename Ptr >`
`Ptr storage::create_object (Ptr const &p)`
- `template<typename Ptr >`
`ReferencePtr storage::make_reference (Ptr &p)`
- `template<typename T >`
`bool storage::serialization (VVEStorage &storage, T &value)`
Function implementing the serialization.
- `int storage::versionNumber (const QString &file_version)`
Helper function to parse version number.

18.33.1 Detailed Description

This file contains the base class and functions to handle storage.

Definition in file [storage.h](#).

18.34 vvelib/storage/storage_xml.h File Reference

This file contains the BIN implementation of the storage class.

```
#include <config.h>
#include <storage/storage.h>
#include <QDomDocument>
#include <list>
#include <storage/types.h>
```

Classes

- class [storage::VVEStorage_XMLReader](#)
Implementation reading an XML file.
- class [storage::VVEStorage_XMLWriter](#)
Implementation writing an XML file.

Namespaces

- namespace [storage](#)
Namespace containing all functions and classes related to VVE persistence.

Defines

- #define **DEF_FIELD**(T) virtual bool field(const **QString**& name, T& value);
- #define **DEF_FIELD**(T) virtual bool field(const **QString**& name, T& value);

18.34.1 Detailed Description

This file contains the BIN implementation of the storage class. This file contains the XML implementation of the storage class.

Definition in file [storage_xml.h](#).

18.35 vvelib/storage/types.h File Reference

This file contains all the utilities to convert from type to type.

```
#include <config.h>
#include <QString>
#include <string>
```

Classes

- struct [storage::ConvertType< From, To >](#)
Convert object of type from to object of type to.
- struct [storage::TypeTraits< T >](#)
Defines main traits for types directly handled.

Namespaces

- namespace [storage](#)
Namespace containing all functions and classes related to VVE persistence.

Defines

- #define **CONVERT_FROM_STRING**(type) template <> struct ConvertType<std::string,type> { bool operator()(const std::string&, type&) const; };
- #define **CONVERT_FROM_STRING**(type) template <> struct ConvertType<**QString**,type> { bool operator()(const **QString**&, type&) const; };
- #define **CONVERT_TO_STRING**(type) template <> struct ConvertType<type,std::string> { bool operator()(const type& from, std::string& to) const; };
- #define **CONVERT_TO_STRING**(type) template <> struct ConvertType<type,**QString**> { bool operator()(const type& from, **QString**& to) const; };
- #define **FOR_ALL_TYPEIDS**(macro)
- #define **FOR_ALL_TYPES**(macro)
- #define **FOR_ALL_TYPES_NOSTRING**(macro)
- #define **NB_STORAGE_TYPES** 17

Enumerations

- enum `NUMBER_CLASS` {
`SIGNED_INTEGER`, `UNSIGNED_INTEGER`, `FLOATING_POINT`,
`CHAR`,
`NOT_A_NUMBER` }
- enum `TYPES` {
`T_INVALID` = -1, `T_BOOL` = 0, `T_WCHART`, `T_CHAR`,
`T_SIGNED_CHAR`, `T_UNSIGNED_CHAR`, `T_SHORT`, `T_UNSIGNED_SHORT`,
`T_INT`, `T_UNSIGNED_INT`, `T_LONG`, `T_UNSIGNED_LONG`,
`T_LONG_LONG`, `T_UNSIGNED_LONG_LONG`, `T_FLOAT`, `T_DOUBLE`,
`T_LONG_DOUBLE`, `T_STRING` }

Functions

- template<typename From , typename To >
bool [storage::convert_type](#) (const From &from, To &to)
Function converting from to to.
- `TYPES` [storage::find_type](#) (`NUMBER_CLASS` klass, size_t bytes)
- bool [storage::isStrictConversion](#) (`TYPES` from, `TYPES` to)
- template<typename T >
`TYPES` [storage::type_id](#) (const T &=T())
- template<typename T >
const `QString` & [storage::type_name](#) (const T &=T())
- const `QString` & [storage::typeid_to_name](#) (`TYPES` id)
- `TYPES` [storage::typename_to_id](#) (const `QString` &name)

18.35.1 Detailed Description

This file contains all the utilities to convert from type to type.

Definition in file [types.h](#).

18.35.2 Define Documentation

18.35.2.1 #define FOR_ALL_TYPEIDS(macro)

Value:

```
macro (T_BOOL, bool) \
    macro (T_WCHART, wchar_t) \
    macro (T_CHAR, char) \
    macro (T_SIGNED_CHAR, signed char) \
```

```
macro(T_UNSIGNED_CHAR,unsigned char) \  
macro(T_SHORT,short) \  
macro(T_UNSIGNED_SHORT,unsigned short) \  
macro(T_INT,int) \  
macro(T_UNSIGNED_INT,unsigned int) \  
macro(T_LONG,long) \  
macro(T_UNSIGNED_LONG,unsigned long) \  
macro(T_LONG_LONG,long long) \  
macro(T_UNSIGNED_LONG_LONG,unsigned long long) \  
macro(T_FLOAT,float) \  
macro(T_DOUBLE,double) \  
macro(T_LONG_DOUBLE,long double) \  
macro(T_STRING,QString)
```

Definition at line 80 of file types.h.

18.35.2.2 #define FOR_ALL_TYPES(macro)

Value:

```
macro(bool) \  
macro(wchar_t) \  
macro(char) \  
macro(signed char) \  
macro(unsigned char) \  
macro(short) \  
macro(unsigned short) \  
macro(int) \  
macro(unsigned int) \  
macro(long) \  
macro(unsigned long) \  
macro(long long) \  
macro(unsigned long long) \  
macro(float) \  
macro(double) \  
macro(long double) \  
macro(std::string) \  
macro(QString)
```

Definition at line 42 of file types.h.

18.35.2.3 #define FOR_ALL_TYPES_NOSTRING(macro)

Value:

```
macro(bool) \  
macro(wchar_t) \  
macro(char) \  
macro(signed char) \  
macro(unsigned char) \  
macro(short) \  
macro(unsigned short) \  
macro(int) \  
macro(unsigned int) \  
macro(long) \  
macro(unsigned long) \  
macro(long long) \  
macro(long long)
```

```
macro(unsigned long long) \  
macro(float) \  
macro(double) \  
macro(long double)
```

Definition at line 62 of file types.h.

18.36 vvelib/storage/vector.h File Reference

```
#include <config.h>
#include <storage/storage.h>
#include <util/vector.h>
```

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Functions

- `template<size_t dim, class T >`
`bool util::serialization (storage::VVEStorage &store, Vector< dim, T > &vec)`

18.36.1 Detailed Description

Definition in file [vector.h](#).

18.37 vvelib/util/vector.h File Reference

Defines the [util::Vector](#) class template.

```
#include <config.h>
#include <iostream>
#include <cassert>
#include <cstdarg>
#include <QTextStream>
#include <util/static_assert.h>
#include <storage/fwd.h>
#include <cmath>
```

Classes

- struct [util::CrossProductType< 2, T >](#)
For 2d -> a scalar.
- struct [util::CrossProductType< 3, T >](#)
For 3d -> a vector.
- class [util::Vector< dim, T >](#)
Vector class supporting all classic classic vector operations.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Functions

- template<size_t dim, typename T, typename T1 >
Vector< dim, T > **util::map** (T(*fct)(const T1 &), const Vector< dim, T1 > &v)
- template<size_t dim, typename T, typename T1 >
Vector< dim, T > **util::map** (const T &(*fct)(const T1 &), const Vector< dim, T1 > &v)
- template<size_t dim, typename T >
Vector< dim, T > **util::map** (T(*fct)(T), const Vector< dim, T > &v)
- template<size_t dim, typename T >
Vector< dim, T > **util::map** (T(*fct)(const T &), const Vector< dim, T > &v)

- `template<size_t dim, typename T >`
 `Vector< dim, T > util::map (const T &(*fct)(const T &), const Vector< dim, T`
 `> &v)`

18.37.1 Detailed Description

Defines the [util::Vector](#) class template.

Definition in file [vector.h](#).

18.38 vvelib/test/graph.vvh File Reference

File containing macros and functions to test properties on graphs.

```
#include <test/test.h>
#include <util/forall.h>
#include <util/unorderedset.h>
```

Namespaces

- namespace [test](#)
Namespace containing test facilities.

Defines

- #define [CHECK_NEIGHBORS_BIGRAPH](#)(G) TEST_RESULT &= test::check_neighborsBiGraph(G, __LINE__)
Check if the neighbors of the vertices of G are in G, if they are not twice in the neighborhood and if a vertex is not a neighbor of itself.
- #define [CHECK_NEIGHBORS_GRAPH](#)(G) TEST_RESULT &= test::check_neighborsGraph(G, __LINE__)
Check if the neighbors of the vertices of G are in G, if they are not twice in the neighborhood and if a vertex is not a neighbor of itself.
- #define [CHECK_SYMMETRIC_BIGRAPH](#)(G) TEST_RESULT &= test::check_symmetricBiGraph(G, __LINE__)
- #define [CHECK_SYMMETRIC_GRAPH](#)(G) TEST_RESULT &= test::check_symmetricGraph(G, __LINE__)
Check if the graph G is symmetric and update result accordingly.

Functions

- template<typename BiGraph >
bool **test::check_neighborsBiGraph** (const BiGraph &G, size_t line_nb)
- template<typename Graph >
bool **test::check_neighborsGraph** (const Graph &G, size_t line_nb)
- template<typename BiGraph >
bool **test::check_symmetricBiGraph** (const BiGraph &G, size_t line_nb)
- template<typename Graph >
bool **test::check_symmetricGraph** (const Graph &G, size_t line_nb)
- template<typename VertexContent >
QString **test::toStr** (const [graph::Vertex](#)< VertexContent > &v)

18.38.1 Detailed Description

File containing macros and functions to test properties on graphs.

Definition in file [graph.vvh](#).

18.38.2 Define Documentation

18.38.2.1 **#define CHECK_NEIGHBORS_BIGRAPH(G) TEST_RESULT &= test::check_neighborsBiGraph(G, __LINE__)**

Check if the neighbors of the vertices of G are in G, if they are not twice in the neighborhood and if a vertex is not a neighbor of itself.

Definition at line 30 of file graph.vvh.

18.38.2.2 **#define CHECK_NEIGHBORS_GRAPH(G) TEST_RESULT &= test::check_neighborsGraph(G, __LINE__)**

Check if the neighbors of the vertices of G are in G, if they are not twice in the neighborhood and if a vertex is not a neighbor of itself.

Definition at line 21 of file graph.vvh.

18.38.2.3 **#define CHECK_SYMMETRIC_GRAPH(G) TEST_RESULT &= test::check_symmetricGraph(G, __LINE__)**

Check if the graph G is symmetric and update `result` accordingly.

18.39 vvelib/test/test.h File Reference

File containing macros and functions to for main testing.

```
#include <config.h>
#include <QTextStream>
#include <util/random.h>
#include <util/vector.h>
```

Namespaces

- namespace [test](#)
Namespace containing test facilities.

Defines

- #define **CHECK**(tst) TEST_RESULT &= test::check(tst, __LINE__, #tst)
- #define [CHECK_EQUAL](#)(tst, value) TEST_RESULT &= test::check_equal(tst, value, __LINE__, #tst, #value)
check if tst is equal to value.
- #define [CHECK_FLOAT](#)(tst, value, epsilon) TEST_RESULT &= test::check_float(tst, value, epsilon, __LINE__, #tst, #value)
Check if tst is equal to value with a relative error of epsilon.
- #define [CHECK_FLOAT_DIFF](#)(tst, value, epsilon) TEST_RESULT &= test::check_float_diff(tst, value, epsilon, __LINE__, #tst, #value)
Check if tst is different to value with a relative error of epsilon.
- #define [CHECK_NOTEQUAL](#)(tst, value) TEST_RESULT &= test::check_notequal(tst, value, __LINE__, #tst, #value)
check if tst is equal to value.
- #define [CHECK_ZERO](#)(tst, epsilon) TEST_RESULT &= test::check_zero(tst, epsilon, __LINE__, #tst)
Check if tst is equal to zero with an error of epsilon.
- #define **FAILED_FUNCTION**(line) test::out << "Test line " << line << " " << __PRETTY_FUNCTION__ << "() failed as ";
- #define **FAILED_TEST**(line, tst) test::out << "Test line " << line << " " << tst << " failed as ";
- #define [TEST_FCT](#)(tst, failed)
Launch the function test_tst (replacing tst with the argument), expect it to return a boolean and display a message indicating if the test was successful or not.

Functions

- bool **test::check** (bool tst, size_t line_nb, const char *tst_str)
- template<typename T1 , typename T2 >
bool **test::check_equal** (const T1 &tst, const T2 &value, size_t line_nb, const char *tst_str, const char *value_str)
- template<typename T1 , typename T2 , typename T3 >
bool **test::check_float** (const T1 &tst, const T2 &value, const T3 &epsilon, size_t line_nb, const char *tst_str, const char *value_str)
- template<typename T1 , typename T2 , typename T3 >
bool **test::check_float_diff** (const T1 &tst, const T2 &value, const T3 &epsilon, size_t line_nb, const char *tst_str, const char *value_str)
- template<typename T1 , typename T2 >
bool **test::check_notequal** (const T1 &tst, const T2 &value, size_t line_nb, const char *tst_str, const char *value_str)
- template<typename T1 , typename T2 >
bool **test::check_zero** (const T1 &tst, const T2 &epsilon, size_t line_nb, const char *tst_str)
- void **test::resetResult** ()
- template<size_t N, typename T >
void **test::shuffle** (T array[])

18.39.1 Detailed Description

File containing macros and functions to for main testing.

Definition in file [test.h](#).

18.39.2 Define Documentation

18.39.2.1 #define CHECK_EQUAL(tst, value) TEST_RESULT &= test::check_equal(tst, value, __LINE__, #tst, #value)

check if tst is equal to value.

The operator == must exist between the two. If not, output the line of the error, show the content of both member (i.e. the output stream operator must exist) and set the variable "TEST_RESULT" to false.

Definition at line 41 of file test.h.

18.39.2.2 #define CHECK_FLOAT(tst, value, epsilon) TEST_RESULT &= test::check_float(tst, value, epsilon, __LINE__, #tst, #value)

Check if tst is equal to value with a relative error of epsilon.

This function is used for floating points number. The absolute epsilon is computed as the relative epsilon multiplied by the biggest absolute values of tst and value;

Definition at line 65 of file test.h.

18.39.2.3 **#define CHECK_FLOAT_DIFF(tst, value, epsilon) TEST_RESULT** **&= test::check_float_diff(tst, value, epsilon, __LINE__, #tst, #value)**

Check if tst is different to value with a relative error of epsilon.

This function is used for floating points number. The absolute epsilon is computed as the relative epsilon multiplied by the biggest absolute values of tst and value;

Definition at line 76 of file test.h.

18.39.2.4 **#define CHECK_NOTEQUAL(tst, value) TEST_RESULT &=** **test::check_notequal(tst, value, __LINE__, #tst, #value)**

check if tst is equal to value.

The operator == must exist between the two. If not, output the line of the error, show the content of both member (i.e. the output stream operator must exist) and set the variable "TEST_RESULT" to false.

Definition at line 54 of file test.h.

18.39.2.5 **#define CHECK_ZERO(tst, epsilon) TEST_RESULT &=** **test::check_zero(tst, epsilon, __LINE__, #tst)**

Check if tst is equal to zero with an error of epsilon.

This function is used for floating points number.

Definition at line 85 of file test.h.

18.39.2.6 **#define TEST_FCT(tst, failed)**

Value:

```
{ \
  bool _11532fd4s85fds_test_failed = !test_##tst(); \
  if(_11532fd4s85fds_test_failed) \
  { \
    failed = true; \
    test::out << #tst " test failed" << endl; \
    setExitCode(10); \
  } \
  else \
    test::out << #tst " test passed" << endl;\
}
```

Launch the function test_tst (replacing tst with the argument), expect it to return a boolean and display a message indicating if the test was successful or not.

If the test failed, the boolean variable put as second argument is set to true. the variable is not changed otherwise.

Definition at line 98 of file test.h.

18.40 vvelib/tissue_model.h File Reference

This files include the tissue model helper.

```
#include <vve.h>
#include <util/parms.h>
#include <util/palette.h>
#include <iostream>
#include <util/watchdog.h>
#include <algorithms/tissue.h>
#include <geometry/geometry.h>
```

Classes

- struct [tissue_model::TissueModel< RealModel, TissueClass >::CompareSize](#)
Operator class to compare the size of cells.
- struct [tissue_model::TissueModel< RealModel, TissueClass >](#)
Base class for the [tissue_model](#) helper.

Namespaces

- namespace [tissue_model](#)
Namespace containing the base class for the [tissue](#) model helper.

Defines

- `#define CellAttributes`
- `#define CellEdgeAttributes`
- `#define CellJunctionEdgeAttributes`
- `#define EndModel }; DEFINE_MODEL(ModelClass)`
- `#define JunctionAttributes`
- `#define JunctionCellEdgeAttributes`
- `#define ModelInit`
- `#define StartModel`
- `#define WallAttributes`

Typedefs

- typedef [util::Palette::Color](#) **Color**

Variables

- const double `tissue_model::epsilon` = 0.0001

Relative constant for geometrical error.

18.40.1 Detailed Description

This files include the tissue model helper. This helper define a tissue that is already drawn.

The namespace `tissue_model` contains the base class for this helper.

To use the helper, simply include `<tissue_model.h>`. Any extra attributes for vertices or edges should be defined before including this file.

Once included, the model starts with

```
StartModel
```

and ends with

```
EndModel
```

The tissue defined in the model uses cells which contains these attributes:

```
double area;           // Area of the cell if the vertex is of type
Point3d pos;           // Position of the vertex
Point3d normal;        // Normal to the surface
double value;          // Value used to draw the vertex
```

and junctions which contains these attributes:

```
Point3d pos;           // Position of the junction
Point3d normal;        // Normal to the surface
Point2d uv;            // Local coordinate of the junction on the bspline
```

In addition, the user can add its own attributes by defining the macro `VertexAttributes` **before** including this file:

```
#define CellAttribute \
    double attr1; \
    int attr2; \
    bool attr3;
#define JunctionAttribute \
    double attr1; \
    bool attr2; \
    int attr3;

#include <tissue_model.h>
```

Note

As this is a macro, to span over many lines you need to end the inner lines with a backslash.

Similarly, the user can add attributes to the cell and tissue edges by defining the macros `CellEdgeAttributes`, `WallAttributes`, `CellJunctionEdgeAttributes` and `JunctionCellEdgeAttributes`.

Within the model class, named `ModelClass`, all the methods and variable of the class `tissue_model::TissueModel` are available. Reference to the parent class can be done using the `ParentClass` typedef.

For example, if yo want to augment the drawing method, you might write:

```
void draw(Viewer* viewer)
{
    ParentClass::draw(viewer);
    // my other commands
}
```

As a small example, here is a code that creates a single cell and change its color when the user click on it.

```
#define CellAttributes \
    bool clicked; // Is the current cell clicked?

#include <tissue_model.h>

StartModel

// Define
int clickedColor, unclickedColor;

// Read the extra parameter
void readParam(util::Parms& parms)
{
    parms("View", "ClickedColor", clickedColor);
    parms("View", "UnclickedColor", unclickedColor);
}

// Initialize the tissue
void initialize()
{
    // First, create the cell
    cell c;
    junction j1, j2, j3, j4;
    j1->pos = Point3d(-1,-1,0);
    j2->pos = Point3d(1,-1,0);
    j3->pos = Point3d(1,1,0);
    j4->pos = Point3d(-1,1,0);
    c->pos = Point3d(0,0,0);
    T.addCell(c, j1, j2, j3, j4);
    // Then, initialise the content of it
    c->clicked = false;
}

// Update the status of the cell after user interaction
void postSelection(const QPoint&, Viewer* viewer)
{
    const cell& c = cellFromId(viewer->selectedName());
    // If a cell was selected
    if(c)
    {
        c->clicked ^= true; // Invert its state
    }
}
```



```
}

// Define the color depending on the clicked state
Color getCellColor(const cell& c)
{
    if(c->clicked)
        return palette.getColor(clickedColor);
    else
        return palette.getColor(unclickedColor);
}

// Define the color depending on the clicked state
Color getCellCenterColor(const cell& c)
{
    if(c->clicked)
        return palette.getColor(clickedColor);
    else
        return palette.getColor(unclickedColor);
}

// All the normals are the same and upward
Point3d normal(const cell&)
{
    return Point3d(0,0,1);
}
Point3d normal(const junction&)
{
    return Point3d(0,0,1);
}

EndModel
```

Definition in file [tissue_model.h](#).

18.41 vvelib/util/assert.h File Reference

Graphical (or textual) assertion utility.

```
#include <config.h>
```

```
#include <cassert>
```

Namespaces

- namespace [util](#)

Various utility classes for generic programming.

Defines

- #define [vvassert](#)(expr)

If NDEBUG is not defined and expr is false, warn the user of the program of the failed assertion.

- #define [vvassert_msg](#)(expr, msg)

Functions

- void [util::__assert_fail](#) (const [QString](#) &assertion, const char *file, unsigned int line, const char *function)
- void [util::__assert_fail_txt](#) (const [QString](#) &assertion, const char *file, unsigned int line, const char *function)

18.41.1 Detailed Description

Graphical (or textual) assertion utility. Define the macro [vvassert](#) (expr)

Definition in file [assert.h](#).

18.41.2 Define Documentation

18.41.2.1 #define vvassert(expr)

Value:

```
((expr) \
                                     ? static_cast<void>(0) \
                                     : util::__assert_fail(#expr, __FILE__, __LINE__, __PR
ETTY_FUNCTION__))
```

If NDEBUG is not defined and `expr` is false, warn the user of the program of the failed assertion.

If TEXT_VVASSERT is defined, `vvassert` falls back onto the default `assert` function, otherwise, it shows a Qt message box with the assert message, then abort the program.

Definition at line 40 of file `assert.h`.

Referenced by `vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false >::adjacentCells()`, `algorithms::TriangleSurface::position()`, and `graph::Vertex< VertexContent >::Vertex()`.

18.41.2.2 #define vvassert_msg(expr, msg)

Value:

```
((expr) \
                                     ? static_cast<void>(0) \
                                     : util::__assert_fail(msg, __FILE__, __LINE__, __PRETTY_FUNCTION__))
```

Definition at line 43 of file `assert.h`.

18.42 vvelib/util/buffer.h File Reference

Defines the [util::Buffer](#) class.

```
#include <config.h>
#include <stdexcept>
#include <iostream>
```

Classes

- class [util::Buffer< T, size >](#)
A ranged checked array.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Functions

- `template<class T , unsigned int size>`
`std::ostream & util::operator<< (std::ostream &os, Buffer< T, size > &b)`
Stream output to the array.
- `template<class T , unsigned int size>`
`std::istream & util::operator>> (std::istream &is, Buffer< T, size > &b)`
Stream input to the array.

18.42.1 Detailed Description

Defines the [util::Buffer](#) class.

Definition in file [buffer.h](#).

18.43 vvelib/util/clamp.h File Reference

Defines the [util::clamp](#) function.

```
#include <config.h>
```

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Functions

- `template<class T >`
`T util::clamp (const T &val, const T &min, const T &max)`
A function to clamp a value to a range.

18.43.1 Detailed Description

Defines the [util::clamp](#) function.

Definition in file [clamp.h](#).

18.44 vvelib/util/color.h File Reference

Defines the [util::Color](#) class template.

```
#include <config.h>
#include <util/vector.h>
#include <util/clamp.h>
#include <QtGui/QColor>
```

Classes

- class [util::Color< T >](#)
A utility class to encapsulate color data.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Typedefs

- typedef [Color< double >](#) **util::Colord**
- typedef [Color< float >](#) **util::Colorf**

Functions

- void **util::convertFromQColor** ([Color< long long >](#) &c, const **QColor** &col)
- void **util::convertFromQColor** ([Color< long >](#) &c, const **QColor** &col)
- void **util::convertFromQColor** ([Color< int >](#) &c, const **QColor** &col)
- void **util::convertFromQColor** ([Color< short >](#) &c, const **QColor** &col)
- void **util::convertFromQColor** ([Color< char >](#) &c, const **QColor** &col)
- void **util::convertFromQColor** ([Color< unsigned long long >](#) &c, const **QColor** &col)
- void **util::convertFromQColor** ([Color< unsigned long >](#) &c, const **QColor** &col)
- void **util::convertFromQColor** ([Color< unsigned int >](#) &c, const **QColor** &col)
- void **util::convertFromQColor** ([Color< unsigned short >](#) &c, const **QColor** &col)
- void **util::convertFromQColor** ([Color< unsigned char >](#) &c, const **QColor** &col)
- void **util::convertFromQColor** ([Color< long double >](#) &c, const **QColor** &col)

- void **util::convertFromQColor** ([Color](#)< double > &c, const **QColor** &col)
- void **util::convertFromQColor** ([Color](#)< float > &c, const **QColor** &col)
- template<class T >
[Color](#)< T > **util::convertHSVtoRGB** (T h, T s, T v)

Return a color based on HSV values.

- **QColor util::convertToQColor** (const [Color](#)< long long > &c)
- **QColor util::convertToQColor** (const [Color](#)< long > &c)
- **QColor util::convertToQColor** (const [Color](#)< int > &c)
- **QColor util::convertToQColor** (const [Color](#)< short > &c)
- **QColor util::convertToQColor** (const [Color](#)< char > &c)
- **QColor util::convertToQColor** (const [Color](#)< unsigned long long > &c)
- **QColor util::convertToQColor** (const [Color](#)< unsigned long > &c)
- **QColor util::convertToQColor** (const [Color](#)< unsigned int > &c)
- **QColor util::convertToQColor** (const [Color](#)< unsigned short > &c)
- **QColor util::convertToQColor** (const [Color](#)< unsigned char > &c)
- **QColor util::convertToQColor** (const [Color](#)< long double > &c)
- **QColor util::convertToQColor** (const [Color](#)< double > &c)
- **QColor util::convertToQColor** (const [Color](#)< float > &c)

18.44.1 Detailed Description

Defines the [util::Color](#) class template.

Definition in file [color.h](#).

18.45 vvelib/util/contour.h File Reference

Defines the [util::Contour](#) class.

```
#include <config.h>
#include <string>
#include <vector>
#include <util/vector.h>
#include <util/watchdog.h>
```

Classes

- class [util::Contour](#)
Contour utility class.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

18.45.1 Detailed Description

Defines the [util::Contour](#) class.

Definition in file [contour.h](#).

18.46 vvelib/util/dir.h File Reference

Defines functions related to directory handling.

```
#include <config.h>
```

```
#include <string>
```

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Functions

- `std::string util::absoluteDir (std::string filename, std::string current_dir=std::string())`
Returns the absolute path of `filename`.

18.46.1 Detailed Description

Defines functions related to directory handling.

Definition in file [dir.h](#).

18.47 vvelib/util/forall.h File Reference

This file contains the defines the forall loops.

```
#include <config.h>
#include <iterator>
#include <utility>
```

Namespaces

- namespace [std](#)
STL namespace.
- namespace [util](#)
Various utility classes for generic programming.

Defines

- #define [forall](#)(typed_var, cont) forall_range(typed_var, util::ForAll::make_range(cont))
Macro iterating over a standard container from cont.begin() to cont.end() or cont.first to cont.second if cont is a pair.
- #define [forall_named](#)(typed_var, cont, name) forall_range(typed_var, std::make_pair((cont).begin_##name(), (cont).end_##name()))
Macro iterating avec a a container but from cont.begin_name() to cont.end_name().
- #define [forall_range](#)(typed_var, range)
Macro allowing for automatic iteration over range.first -> range.second (range being a std::pair).
- #define [forall_reverse](#)(typed_var, cont) forall_range(typed_var, util::ForAll::make_reverse_range(cont))
Macro iterating over a standard reversible container from cont.rbegin() to cont.rend() ot from std::reverse_iterator(cont.second) to std::reverse_iterator(cont.first) if cont is a pair.

Functions

- template<typename T >
const T & **std::begin** (const pair< T, T > &p)
- template<typename T >
const T & **std::end** (const pair< T, T > &p)

18.47.1 Detailed Description

This file contains the defines the forall loops. Forall allows to iterate on STL container or range.

Note

A STL range is a pair of iterator where the first element of the pair is the beginning the second element of the pair is the end.

Use examples:

```
std::vector<int> va;
// ... Filling in va ...
forall(int a, va)
{
    cout << a << endl;
}
```

This example shows how to iterate on a container returned by value by a function:

```
std::vector<int> create_vector();
// Now put the output of the function in a local variable
std::vector<int> va = create_vector();
forall(int a, va)
{
    cout << a << endl;
}
```

Note

You should not iterate directly on it! The loop do not keep the container alive.

Use example with range:

```
typedef std::vector<int>::iterator iterator;
std::pair<iterator,iterator> extract(std::vector<int> v, int first, int last);
std::vector<int> va;
// ... Filling in va ...
forall(int& i, extract(va, 2, 10))
{
    i += 5;
    cout << i << endl;
}
```

In this example, the values in the vector are changed too.

Definition in file [forall.h](#).

18.47.2 Define Documentation

18.47.2.1 #define forall(typed_var, cont) forall_range(typed_var, util::ForAll::make_range(cont))

Macro iterating over a standard container from `cont.begin()` to `cont.end()` or `cont.first` to `cont.second` if `cont` is a pair.

Warning

The container has to be maintained for the whole duration of the loop. All functions of VV are safe to that respect. But you should not use it to iterate on a container returned by value.

Definition at line 212 of file forall.h.

```

Referenced by vvcomplex::VVComplex< MANDATORY_COMPLEX_
TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue<
ALL_COMPLEX_TEMPLATE_ARGS >)>::addCellUnsorted(),
vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue<
ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_
EmptyEdgeContent, graph::_EmptyEdgeContent, false >::adjacentCells(),
util::Parms::all(), vvcomplex::VVComplex< MANDATORY_COMPLEX_
TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_
COMPLEX_TEMPLATE_ARGS >)>::border(), vvcomplex::VVComplexGraph<
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_
ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_
EmptyEdgeContent, false >::border(), tissue::Tissue< Model, CellContent, Junction
Content, WallContent, CellEdgeContent, CellJunctionContent, JunctionCellContent,
compact, LeafClass >::calcQuads(), vvcomplex::VVComplex< MANDATORY_
COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue<
ALL_COMPLEX_TEMPLATE_ARGS >)>::connectFromJunctions(), algo-
rithms::TriangleSurface::contour(), vvcomplex::VVComplexGraph< RESOLVE_
LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS
>), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent,
false >::deleteCell(), vvcomplex::VVComplex< MANDATORY_
COMPLEX_TEMPLATE_ARGS, RESOLVE_LEAF_CLASS(LeafClass,
Tissue< ALL_COMPLEX_TEMPLATE_ARGS >)>::deleteCellInGraphs(),
vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_ARGS,
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_
ARGS >)>::divideCell(), vvcomplex::VVComplexGraph< RESOLVE_
LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS
>), JunctionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent,
false >::divideCell(), tissue_model::TissueModel< RealModel, TissueClass
>::draw(), bspline_tissue_model::TissueModel< RealModel, TissueClass
>::draw(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass
>::drawCell(), tissue::Tissue< Model, CellContent, JunctionContent, WallContent,
CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, LeafClass
>::drawCellContour(), tissue::Tissue< Model, CellContent, JunctionContent, Wall
Content, CellEdgeContent, CellJunctionContent, JunctionCellContent, compact, Leaf
Class >::drawSimplifiedCell(), algorithms::drawSymetricVVGraph(), tissue::Tissue<
Model, CellContent, JunctionContent, WallContent, CellEdgeContent, CellJunction
Content, JunctionCellContent, compact, LeafClass >::drawWalledCell(), vvcom-
plex::findCenter(), tissue::findDivisionPoints(), cell_system::findDivisionPoints(),
vvcomplex::FindOppositeWall(), solver::Solver< nb_vars, identifier
>::FindPartials(), complex_factory::hex_grid(), algorithms::TriangleGrowth::init(),
vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue<
ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_

```

```

EmptyEdgeContent, graph::_EmptyEdgeContent, false >::interfaceWall(),
vvcomplex::VVComplexGraph< RESOLVE_LEAF_CLASS(LeafClass, Tissue<
ALL_COMPLEX_TEMPLATE_ARGS >), JunctionContent, graph::_
EmptyEdgeContent, graph::_EmptyEdgeContent, false >::interfaceWallSpan(),
vvcomplex::VVComplex< MANDATORY_COMPLEX_TEMPLATE_
ARGS, RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_
TEMPLATE_ARGS >)>::mergeCells(), vvcomplex::VVComplexGraph<
RESOLVE_LEAF_CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_
ARGS >), JunctionContent, graph::_EmptyEdgeContent, graph::_
EmptyEdgeContent, false >::mergeCells(), complex_factory::objreader(),
algorithms::TriangleSurface::position(), algorithms::TriangleGrowth::readOBJS(),
storage::VVEStorage_XMLReader::setLastError(), solver::Solver< nb_
vars, identifier >::solveAdaptiveCrankNicholson(), solver::Solver< nb_
vars, identifier >::solveAdaptiveEuler(), solver::Solver< nb_vars, identifier
>::solveAdaptiveRungeKutta(), solver::Solver< nb_vars, identifier >::solveEuler(),
solver::Solver< nb_vars, identifier >::solveFixedpoint(), solver::Solver<
nb_vars, identifier >::solveMidpoint(), solver::Solver< nb_vars, identifier
>::solveRungeKutta(), vvcomplex::VVComplexGraph< RESOLVE_LEAF_
CLASS(LeafClass, Tissue< ALL_COMPLEX_TEMPLATE_ARGS >), Junc-
tionContent, graph::_EmptyEdgeContent, graph::_EmptyEdgeContent, false
>::splitWall(), complex_factory::square_grid(), cell_system::CellSystem< Tis-
sueClass, RealModel >::step_cellsystem_division(), cell_system::CellSystem<
TissueClass, RealModel >::step_cellsystem_meca(), tissue_model::TissueModel<
RealModel, TissueClass >::updateCellsArea(), bspline_tissue_model::TissueModel<
RealModel, TissueClass >::updateCellsArea(), cell_system::CellSystem< Tis-
sueClass, RealModel >::updateCellsArea(), bspline_tissue_model::TissueModel<
RealModel, TissueClass >::updatePositions(), and util::WatchDog::watch().

```

18.47.2.2 #define forall_named(typed_var, cont, name) forall_range(typed_var, std::make_pair((cont).begin_##name(), (cont).end_##name()))

Macro iterating avec a a container but from `cont.begin_name()` to `cont.end_name()`.

Warning

In the current implementation, `cont` is evaluated twice!

Definition at line 234 of file `forall.h`.

Referenced by `tissue_model::TissueModel< RealModel, TissueClass >::draw()`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::draw()`, `tissue_model::TissueModel< RealModel, TissueClass >::drawWithNames()`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::drawWithNames()`, `complex_factory::objreader()`, `cell_system::CellSystem< TissueClass, RealModel >::step_cellsystem_meca()`, `tissue_model::TissueModel< RealModel, TissueClass >::updateCellsArea()`, `bspline_tissue_model::TissueModel< RealModel, TissueClass >::updateCellsArea()`, and `cell_system::CellSystem< TissueClass, RealModel >::updateCellsArea()`.

18.47.2.3 #define forall_range(typed_var, range)

Value:

```
for( const util::ForAll::BaseForwardIter& iter = util::ForAll::forwardIter( range
    , forall_pointer( (range).first ) ) ; \
!util::ForAll::castForwardIter( &iter, forall_pointer( (range).first ) )->is_end(
    ) ; \
++( util::ForAll::castForwardIter( &iter, forall_pointer( (range).first ) )->it )
    ) \
for( typed_var = util::ForAll::castForwardIter( &iter, forall_pointer( (range).fi
    rst ) )->value() ; \
util::ForAll::castForwardIter( &iter, forall_pointer( (range).first ) )->brk ; \
--( util::ForAll::castForwardIter( &iter, forall_pointer( (range).first ) )->brk
    ) )
```

Macro allowing for automatic iteration over `range.first -> range.second` (range being a `std::pair`).

Definition at line 191 of file `forall.h`.

18.47.2.4 #define forall_reverse(typed_var, cont) forall_range(typed_var, util::ForAll::make_reverse_range(cont))

Macro iterating over a standard reversible container from `cont.rbegin()` to `cont.rend()` or from `std::reverse_iterator(cont.second)` to `std::reverse_iterator(cont.first)` if `cont` is a pair.

Warning

The container has to be maintained for the whole duration of the loop. All functions of VV are safe to that respect. But you should not use it to iterate on a container returned by value.

Definition at line 226 of file `forall.h`.

18.48 vvelib/util/function.h File Reference

Defines the [util::Function](#) class.

```
#include <config.h>
#include <string>
#include <vector>
#include <util/watchdog.h>
#include <util/vector.h>
```

Classes

- class [util::Function](#)
A utility class for functions.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

18.48.1 Detailed Description

Defines the [util::Function](#) class.

Definition in file [function.h](#).

18.49 vvelib/util/glerrorcheck.h File Reference

Define the `OPENGL_ERROR_CHECK` macro.

```
#include <config.h>
#include <iostream>
```

Defines

- `#define OPENGL_ERROR_CHECK(cmd)`
The following macro checks if an OpenGL has occurred.

18.49.1 Detailed Description

Define the `OPENGL_ERROR_CHECK` macro.

Definition in file [glerrorcheck.h](#).

18.49.2 Define Documentation

18.49.2.1 `#define OPENGL_ERROR_CHECK(cmd)`

Value:

```
cmd; \
{ \
    GLenum error = GL_NO_ERROR; \
    while ((error = glGetError()) != GL_NO_ERROR) { \
        std::cerr << "OpenGL error found. [ " << __FILE__ \
            << " : " << __LINE__ \
            << " : " << #cmd \
            << " ] \"" << gluErrorString(error) \
            << "\"." << std::endl; \
    } \
}
```

The following macro checks if an OpenGL has occurred.

Parameters

cmd The command to check.

This macro is based on that found in the notes for SIGGRAPH 2003 Course 26. It is modified here to use C++ (it was originally in C) and in the output format.

Definition at line 22 of file `glerrorcheck.h`.

18.50 vvelib/util/graph_inset.h File Reference

Defines the [util::GraphInset](#) class.

```
#include <config.h>
#include <vector>
#include <list>
#include <string>
#include <graph/vertex.h>
#include <util/vector.h>
#include <util/parms.h>
```

Classes

- struct [util::GraphInset::Data](#)
Data structure used to store the time stamped values.
- class [util::GraphInset](#)
Class used to draw a graph of time stamped data in the model window.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

18.50.1 Detailed Description

Defines the [util::GraphInset](#) class.

Definition in file [graph_inset.h](#).

18.51 vvelib/util/leaf_class.h File Reference

Define templates and macros to figure out, at compile time, what class is the leaf class of a chain of inheritance.

```
#include <config.h>
```

Namespaces

- namespace [template_utils](#)
Namespace for meta-programming utils using templates.

Defines

- #define **RESOLVE_LEAF_CLASS**(T, Self) typename template_utils::resolve_this<T,Self>::type

18.51.1 Detailed Description

Define templates and macros to figure out, at compile time, what class is the leaf class of a chain of inheritance.

Definition in file [leaf_class.h](#).

18.52 vvelib/util/materials.h File Reference

Defines the [util::Materials](#) class.

```
#include <config.h>
#include <string>
#include <qgl.h>
#include <util/watchdog.h>
```

Classes

- struct [util::Materials::Material](#)
The material data structure.
- class [util::Materials](#)
A utility class for materials.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

18.52.1 Detailed Description

Defines the [util::Materials](#) class.

Definition in file [materials.h](#).

18.53 vvelib/util/member_iterator.h File Reference

Defines the [util::SelectMemberIterator](#) class template.

```
#include <config.h>
#include <iterator>
```

Classes

- class [util::SelectMemberIterator](#)< [Iterator](#), [T](#), [member](#), [Reference](#), [Pointer](#) >
Iterate over a container of structure, dereferencing only a member of it.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

18.53.1 Detailed Description

Defines the [util::SelectMemberIterator](#) class template.

Definition in file [member_iterator.h](#).

18.54 vvelib/util/palette.h File Reference

Defines the [util::Palette](#) class.

```
#include <config.h>
#include <string>
#include <util/gl.h>
#include <util/color.h>
#include <util/watchdog.h>
```

Classes

- class [util::Palette](#)
A utility class for palettes.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

18.54.1 Detailed Description

Defines the [util::Palette](#) class.

Definition in file [palette.h](#).

18.55 vvelib/util/parms.h File Reference

Defines the [util::Parms](#) class.

```
#include <config.h>
#include <string>
#include <map>
#include <sstream>
#include <iostream>
#include <vector>
#include <set>
#include <list>
#include <util/forall.h>
#include <QString>
#include <iterator>
#include <QTextStream>
#include <QStringList>
```

Classes

- class [util::Parms](#)
A utility class to parse L-Studio like parameter files.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Functions

- [QTextStream & util::operator>>](#) ([QTextStream &ss](#), bool b)

18.55.1 Detailed Description

Defines the [util::Parms](#) class.

Definition in file [parms.h](#).

18.56 vvelib/util/point.h File Reference

Defines the (deprecated) [util::Point](#) class template.

```
#include <config.h>
#include <iostream>
#include <cmath>
#include <util/vector.h>
```

Classes

- class [util::Point< T >](#)
A 3D point/vector class.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Functions

- template<class T >
Point< T > [util::operator*](#) (const T &s, const Point< T > &p)
Scalar multiplication operator.
- template<class T >
Point< T > [util::operator*](#) (const Point< T > &p, const T &s)
Scalar multiplication operator.

18.56.1 Detailed Description

Defines the (deprecated) [util::Point](#) class template. New code should use [util::Vector](#) instead.

Definition in file [point.h](#).

18.57 vvelib/util/random.h File Reference

Defines various functions to generate random numbers.

```
#include <config.h>
#include <util/vector.h>
```

Namespaces

- namespace `util`
Various utility classes for generic programming.

Functions

- `template<size_t dim>`
`Vector< dim, double > util::gaussRan (const Vector< dim, double > &mean,`
`const Vector< dim, double > &sigma)`
Generate a random vector with gaussian distribution.
- `double util::gaussRan (double mean, double sigma)`
Generate a random number with gaussian distribution.
- `template<size_t dim>`
`Vector< dim, double > util::ran (const Vector< dim, double > &V)`
Generate a random vector uniformly distributed between Vector(0) and V.
- `double util::ran (double M)`
Generate a random number uniformly distributed between 0 and M.
- `template<size_t dim>`
`Vector< dim, long int > util::random (const Vector< dim, long int > &n)`
Returns a vector of random numbers.
- `long int util::random (long int n)`
Returns a random number between 0 and n (excluded), for $n \leq RAND_MAX$.
- `long int util::random ()`
Returns a random number between 0 and $RAND_MAX$.
- `void util::sran (unsigned int seed)`
Initialize the random number generator.
- `unsigned int util::sran_time ()`
Initialize the random number with the current time of the day (in microsecond).

18.57.1 Detailed Description

Defines various functions to generate random numbers.

Definition in file [random.h](#).

18.58 vvelib/util/range.h File Reference

Defines the range util template and various functions to deal with numerical range.

```
#include <config.h>
#include <utility>
#include <iterator>
```

Classes

- class [util::range< Iterator >](#)
Class representing a range of values from two iterators.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Functions

- `template<typename Iterator >`
`range< Iterator > util::make_range (const std::pair< Iterator, Iterator > &p)`
[Function](#) to create a range from a pair of iterators.
- `template<typename Iterator >`
`range< Iterator > util::make_range (const Iterator &first, const Iterator &last)`
[Function](#) to create a range from two iterators.
- `template<typename T >`
`bool util::range_c (const T &min, const T &max, const T &val)`
Check if a value is in a C-style range (closed on lower bound and open on the upper).
- `template<typename T >`
`bool util::range_closed (const T &min, const T &max, const T &val)`
Check if a value is in an closed range.
- `template<typename T >`
`bool util::range_open (const T &min, const T &max, const T &val)`
Check if a value is in an open range.

18.58.1 Detailed Description

Defines the range util template and various functions to deal with numerical range.

Definition in file [range.h](#).

18.59 vvelib/util/static_assert.h File Reference

Define the `STATIC_ASSERT` macro.

```
#include <config.h>
```

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Defines

- #define **DO_JOIN**(X, Y) DO_JOIN2(X,Y)
- #define **DO_JOIN2**(X, Y) X##Y
- #define **JOIN**(X, Y) DO_JOIN(X, Y)
- #define **STATIC_ASSERT**(B, txt)

Assertion that works at compile time.

18.59.1 Detailed Description

Define the `STATIC_ASSERT` macro.

Definition in file [static_assert.h](#).

18.59.2 Define Documentation

18.59.2.1 #define STATIC_ASSERT(B, txt)

Value:

```
typedef util::static_assert_test<\
    sizeof(util::STATIC_ASSERTION_FAILURE< (bool) ( B ) >)>\
    JOIN(static_assert_typedef_, __LINE__)
```

Assertion that works at compile time.

If the parameters evaluate to false, the program won't compile. Useful to check that a template method is not used when it is invalid.

A typical example can be found in the implementation of the [util::Vector](#) class to make sure the constructor with n (fixed) values are used only if the vector is of dimension n:

```
template <size_t N, typename T>
Vector<N,T>::Vector( const T& v1, const T& v2)
{
    STATIC_ASSERT(N == 2); // Compilation will fail if N != 2
    // ...
}
```

Note

This will work because, as specified by the standard, a (non-virtual) method of class template is instantiated only if used. So, as long as the user do not try to construct a vector with two values, this specific constructor won't even exist.

Definition at line 48 of file static_assert.h.

Referenced by `util::Vector< nCols, double >::cross()`, `util::Vector< nCols, double >::i()`, `util::Vector< nCols, double >::j()`, `util::Vector< nCols, double >::k()`, `util::Vector< nCols, double >::l()`, `util::Vector< nCols, double >::projectXY()`, `util::Vector< nCols, double >::set()`, `util::Vector< nCols, double >::t()`, `util::Vector< nCols, double >::Vector()`, `util::Vector< nCols, double >::x()`, `util::Vector< nCols, double >::y()`, and `util::Vector< nCols, double >::z()`.

18.60 vvelib/util/tensor.h File Reference

Defines the [util::Tensor](#) class template.

```
#include <config.h>
#include <iostream>
#include <cmath>
#include <util/point.h>
#include <util/vector.h>
```

Classes

- class [util::Tensor< T >](#)
A growth tensor class.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Functions

- `template<class T >`
`std::ostream & util::operator<< (std::ostream &os, Tensor< T > &tensor)`
- `template<class T >`
`std::istream & util::operator>> (std::istream &is, Tensor< T > &tensor)`

18.60.1 Detailed Description

Defines the [util::Tensor](#) class template.

Definition in file [tensor.h](#).

18.61 vvelib/util/texture.h File Reference

Defines the [util::Texture1D](#) and [util::Texture2D](#) classes.

```
#include <config.h>
```

```
#include <string>
```

```
#include <qgl.h>
```

Classes

- class [util::Texture1D](#)
A utility class for one-dimensional textures.
- class [util::Texture2D](#)
A utility class for two-dimensional textures.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

18.61.1 Detailed Description

Defines the [util::Texture1D](#) and [util::Texture2D](#) classes.

Definition in file [texture.h](#).

18.62 vvelib/util/tie.h File Reference

Defines the [util::tie](#) function.

```
#include <config.h>
```

```
#include <utility>
```

Classes

- class [util::refpair](#)< T, U >
Class used to hold references for the [util::tie\(\)](#) function.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

Functions

- template<typename T , typename U >
 [refpair](#)< T, U > [util::tie](#) (T &x, U &y)
 Tie two variables to the values of a pair.

18.62.1 Detailed Description

Defines the [util::tie](#) function.

Definition in file [tie.h](#).

18.63 vvelib/util/watchdog.h File Reference

Defines the [util::WatchDog](#) and [util::FileObject](#) classes.

```
#include <config.h>
#include <string>
#include <set>
#include <map>
#include <QObject>
#include <QString>
#include <util/forall.h>
```

Classes

- class [util::FileObject](#)
This class is the base class for any object that might be watched by the [WatchDog](#).
- class [util::WatchDog](#)
Watch the modifications of file objects for you.

Namespaces

- namespace [util](#)
Various utility classes for generic programming.

18.63.1 Detailed Description

Defines the [util::WatchDog](#) and [util::FileObject](#) classes.

Definition in file [watchdog.h](#).

Index

- ~Arc
 - graph::Arc, [190](#)
- ~Buffer
 - util::Buffer, [204](#)
- ~Model
 - Model, [368](#)
- ~Point
 - util::Point, [416](#)
- ~Tensor
 - util::Tensor, [506](#)
- ~Texture1D
 - util::Texture1D, [510](#)
- ~Texture2D
 - util::Texture2D, [516](#)
- ~VVComplex
 - vvcomplex::VVComplex, [802](#)
- ~VVEStorage
 - storage::VVEStorage, [857](#)
- ~Vertex
 - graph::Vertex, [664](#)
- ~Viewer
 - Viewer, [681](#)
- _content
 - graph::Edge, [287](#)
 - graph::Vertex, [672](#)
- _numberOfLines
 - util::GraphInset, [322](#)
- _source
 - graph::Edge, [287](#)
- _target
 - graph::Edge, [287](#)
- a
 - util::Color, [244](#), [245](#)
- absoluteDir
 - util, [159](#)
- acquire
 - graph::Vertex, [665](#)
- actions
 - Model, [368](#)
- AdaptiveCrankNicholson
 - solver, [132](#)
- AdaptiveEuler
 - solver, [132](#)
- AdaptiveRungeKutta
 - solver, [132](#)
- addAction
 - Model, [368](#)
- addActions
 - Model, [368](#)
- addCell
 - vvcomplex::VVComplex, [803–807](#)
 - vvcomplex::VVComplexGraph, [838–841](#)
- addCellExtra
 - vvcomplex::VVComplex, [808](#)
- addCellUnsorted
 - vvcomplex::VVComplex, [808](#)
- addData
 - util::GraphInset, [312–314](#)
- addMenuItem
 - Model, [369](#)
- addObject
 - util::WatchDog, [975](#)
- addToolBar
 - Viewer, [681](#)
- addViewer
 - Viewer, [682](#)
- adjacentCell
 - vvcomplex::VVComplex, [809](#)
 - vvcomplex::VVComplexGraph, [841](#)
- adjacentCells
 - vvcomplex::VVComplex, [810](#)
 - vvcomplex::VVComplexGraph, [842](#)
- AEulerHighTol
 - solver::Solver, [494](#)
- AEulerIncDt
 - solver::Solver, [494](#)
- AEulerLowTol
 - solver::Solver, [494](#)
- AEulerResDt
 - solver::Solver, [494](#)

- AEulerResTol
 - solver::Solver, [494](#)
- AEulerTolType
 - solver::Solver, [495](#)
- algorithms, [73](#)
 - drawSymetricVVGraph, [75](#)
 - drawVVBIGraphConnections1, [75](#)
 - drawVVBIGraphConnections2, [76](#)
 - drawVVGraphConnections, [76](#)
 - insertAfter, [76](#)
 - insertBefore, [77](#)
 - shortest_paths_FloydWarshall, [77](#)
 - split, [77](#)
- algorithms/complex.h
 - EXPORT_COMPLEX_EDGES, [995](#)
 - EXPORT_COMPLEX_EDGES_-PREFIX, [995](#)
 - EXPORT_COMPLEX_GRAPHS, [996](#)
 - EXPORT_COMPLEX_GRAPHS_-PREFIX, [996](#)
 - EXPORT_COMPLEX_TYPES, [996](#)
 - EXPORT_COMPLEX_TYPES_-PREFIX, [997](#)
 - EXPORT_COMPLEX_VERTICES, [997](#)
 - EXPORT_COMPLEX_-VERTICES_PREFIX, [997](#)
 - IMPORT_COMPLEX_EDGES, [998](#)
 - IMPORT_COMPLEX_GRAPHS, [998](#)
 - IMPORT_COMPLEX_MODEL, [998](#)
 - IMPORT_COMPLEX_TYPES, [998](#)
 - IMPORT_COMPLEX_VERTICES, [999](#)
- algorithms::Insert, [329](#)
- algorithms::TriangleGrowth, [588](#)
 - contour, [589](#)
 - endTime, [590](#)
 - errorMsg, [590](#)
 - init, [590](#)
 - readOBJS, [593](#)
 - setTime, [600](#)
 - size, [601](#)
 - startTime, [601](#)
 - surface, [601](#), [602](#)
 - time, [602](#)
 - timePos, [602](#)
 - valid, [603](#)
- algorithms::TriangleSurface, [604](#)
 - CellComplex, [606](#)
 - center3d, [606](#)
 - centerPosition, [606](#)
 - clear, [607](#)
 - contour, [607](#)
 - faces, [612](#)
 - normal, [609](#)
 - pmin, [612](#)
 - point3d, [609](#)
 - position, [609](#), [610](#)
 - S, [613](#)
 - smoothNormal, [611](#)
 - time, [613](#)
 - vector, [611](#), [612](#)
 - vector3d, [612](#)
 - vertices, [613](#)
- algorithms::TriangleSurface::Cell, [205](#)
 - id, [205](#)
 - normal, [205](#)
- algorithms::TriangleSurface::Position, [419](#)
- algorithms::TriangleSurface::Vertex, [656](#)
 - id, [656](#)
 - normal, [656](#)
 - pos, [656](#)
- all
 - util::Parms, [404–409](#)
- angle
 - cell_
 - system::CellSystemDivisionParams, [230](#)
 - cell_system::DivisionParams, [276](#)
 - geometry::Quaternion, [424](#)
 - util::Vector, [645](#), [646](#)
- angle_theta
 - util::Tensor, [506](#)
- animationPeriod
 - Model, [370](#)
- any
 - graph::VVGraph, [922](#)
- any_cell
 - vvcomplex::VComplexGraph, [843](#)
- any_junction
 - vvcomplex::VComplexGraph, [843](#)
- any_vertex1
 - graph::VVBIGraph, [729](#)
- any_vertex2
 - graph::VVBIGraph, [729](#)

- anyIn
 - graph::VVBIGraph, [730](#)
 - graph::VVGraph, [923](#)
- approx
 - util, [160](#)
- Arc
 - graph::Arc, [190](#)
- arc
 - graph::VVGraph, [924](#)
- arc_t
 - graph::VVGraph, [916](#)
- area
 - cell_system::CellSystemCell, [228](#)
- arguments
 - Model, [370](#)
- ARungeHighTol
 - solver::Solver, [495](#)
- ARungeIncDt
 - solver::Solver, [495](#)
- ARungeLowTol
 - solver::Solver, [495](#)
- ARungeResDt
 - solver::Solver, [495](#)
- ARungeResTol
 - solver::Solver, [496](#)
- ARungeTolType
 - solver::Solver, [496](#)
- assert.h
 - vvassert, [1060](#)
 - vvassert_msg, [1061](#)
- autoOpenClose
 - util::GraphInset, [322](#)
- axis
 - geometry::Quaternion, [424](#)
- b
 - util::Color, [245](#)
- backColor
 - bspline_tissue_model::TissueModel, [584](#)
 - tissue_model::TissueModel, [570](#)
 - util::GraphInset, [322](#)
- backgroundOffset
 - util::GraphInset, [322](#)
- BAD_CONTENT
 - storage, [139](#)
- barycentricToCartesian
 - geometry, [114](#)
- barycentricToCartesian_matrix
 - geometry, [114](#)
- base
 - util::SelectMemberIterator, [452](#)
- base_iterator
 - util::SelectMemberIterator, [450](#)
- begin
 - graph::VVGraph, [924](#), [925](#)
 - util::range, [432](#)
 - util::Vector, [627](#), [628](#)
- begin_vertex1
 - graph::VVBIGraph, [731](#)
- begin_vertex2
 - graph::VVBIGraph, [731](#)
- bind
 - util::Texture1D, [511](#)
 - util::Texture2D, [517](#)
- blend
 - util::Materials, [334](#)
 - util::Palette, [394](#)
 - util::Texture1D, [511](#)
 - util::Texture2D, [517](#)
- blending
 - tissue::Tissue, [552](#)
- border
 - vvcomplex::VVComplex, [810–812](#)
 - vvcomplex::VVComplexGraph, [843](#), [844](#)
- BoundingBox
 - util::Shapes::BSplineSurface, [196](#)
- bspline_tissue_model, [79](#)
 - epsilon, [79](#)
- bspline_tissue_model.h
 - Color, [1019](#)
 - EndModel, [1019](#)
 - StartModel, [1019](#)
- bspline_tissue_model::TissueModel, [572](#)
 - backColor, [584](#)
 - cellFromId, [576](#)
 - cellInitWalls, [584](#)
 - draw, [576](#)
 - drawNeighborhood, [584](#)
 - drawWithNames, [577](#)
 - dt, [585](#)
 - getCellCenterColor, [578](#)
 - getCellColor, [578](#)
 - growthStartTime, [585](#)
 - initDraw, [578](#)
 - initialize, [578](#)
 - initTissue, [579](#)
 - leaf, [585](#)
 - leaves, [585](#)

- modifiedFiles, [579](#)
- ordered_cells_t, [575](#)
- palette, [586](#)
- postDraw, [580](#)
- preDraw, [580](#)
- readParam, [581](#)
- readTissueParam, [581](#)
- registerFiles, [582](#)
- SetPos, [582](#)
- step, [583](#)
- T, [586](#)
- time, [586](#)
- TissueModel, [575](#)
- updateCellsArea, [583](#)
- updatePositions, [583](#)
- bspline_tissue_-
 - model::TissueModel::CompareSizeell, [248](#)
- Buffer
 - util::Buffer, [203](#)
- C
 - vvcomplex::VVComplex, [827](#)
- c_data
 - util::Buffer, [204](#)
 - util::Matrix, [347](#)
 - util::Point, [417](#)
 - util::Vector, [628](#)
- cache
 - graph::Vertex, [673](#)
- cache_source
 - graph::Vertex, [673](#)
- calcQuads
 - tissue::Tissue, [528](#)
- cameraRecordingStart
 - Viewer, [682](#)
- cameraRecordingStop
 - Viewer, [682](#)
- cartesianToBarycentric
 - geometry, [114](#)
- cartesianToBarycentric_matrix
 - geometry, [114](#)
- cbegin
 - util::range, [433](#)
- cell
 - vvcomplex, [180](#)
 - vvcomplex::VVComplex, [796](#)
- cell_arc
 - vvcomplex::VVComplex, [796](#)
- CELL_DIVISION_ALGORITHM
 - tissue, [146](#)
- cell_edge
 - vvcomplex, [180](#)
 - vvcomplex::VVComplex, [796](#)
- cell_graph
 - tissue, [145](#)
 - vvcomplex, [180](#)
 - vvcomplex::VVComplex, [796](#)
- cell_junction_edge
 - vvcomplex, [180](#)
 - vvcomplex::VVComplex, [797](#)
- cell_system, [80](#)
 - findDivisionPoints, [81](#), [83](#)
 - label_t, [81](#)
 - removeWhitespace, [84](#)
 - volumeLeftCell, [84](#)
- cell_system::CellSystem, [208](#)
 - CellSystem, [212](#)
 - current_div, [222](#)
 - damping, [222](#)
 - division_rules, [222](#)
 - division_rules_t, [212](#)
 - dt, [223](#)
 - ke, [223](#)
 - ki, [223](#)
 - label, [212](#)
 - label_change_rules, [223](#)
 - label_change_rules_t, [212](#)
 - label_names, [223](#)
 - label_numbers, [224](#)
 - mass, [224](#)
 - pressure, [224](#)
 - readMechanicParam, [213](#)
 - readParam, [213](#)
 - readRules, [214](#)
 - readSymbols, [216](#)
 - registerSymbol, [217](#)
 - restLength, [224](#)
 - serialize, [217](#)
 - showInterval, [225](#)
 - showTime, [225](#)
 - stability, [225](#)
 - step, [218](#)
 - step_cellsystem, [218](#)
 - step_cellsystem_division, [218](#)
 - step_cellsystem_meca, [219](#)
 - T, [225](#)
 - time, [225](#)
 - updateCellsArea, [220](#)
 - version, [221](#)

- versionNumber, [221](#)
 - viewRulesApplication, [226](#)
 - viewRulesDefinitions, [226](#)
- cell_system::CellSystemCell, [227](#)
 - area, [228](#)
 - label, [228](#)
 - pos, [228](#)
 - serialize, [227](#)
- cell_system::CellSystemDivisionParams, [229](#)
 - angle, [230](#)
 - CellSystemDivisionParams, [230](#)
 - direction, [230](#)
 - epsilon, [231](#)
 - minCellWall, [231](#)
 - pinching, [231](#)
 - pinchingParam, [231](#)
 - ratio, [231](#)
- cell_system::CellSystemJunction, [233](#)
 - force, [234](#)
 - pos, [234](#)
 - serialize, [233](#)
 - velocity, [234](#)
- cell_system::DivisionParams, [276](#)
 - angle, [276](#)
 - left_label, [276](#)
 - ratio, [277](#)
 - right_label, [277](#)
- CellComplex
 - algorithms::TriangleSurface, [606](#)
- cellDivAlg
 - tissue::Tissue, [552](#)
- cellFromId
 - bspline_tissue_model::TissueModel, [576](#)
 - tissue_model::TissueModel, [564](#)
- cellInitWalls
 - bspline_tissue_model::TissueModel, [584](#)
- cellMaxPinch
 - tissue::CellPinchingParams, [207](#)
 - tissue::Tissue, [553](#)
- cellPinch
 - tissue::CellPinchingParams, [207](#)
 - tissue::Tissue, [553](#)
- cellPinching
 - tissue, [146](#)
- CellPinchingParams
 - tissue::CellPinchingParams, [207](#)
- CellSystem
 - cell_system::CellSystem, [212](#)
- CellSystemDivisionParams
 - cell_
 - system::CellSystemDivisionParams, [230](#)
- cellWallCorner
 - tissue::Tissue, [553](#)
- cellWallMin
 - tissue::ClosestMidAlgoParams, [239](#)
 - tissue::ClosestWallAlgoParams, [241](#)
 - tissue::ShortWallAlgoParams, [460](#)
 - tissue::Tissue, [554](#)
- cellWallSample
 - tissue::ShortWallAlgoParams, [460](#)
 - tissue::Tissue, [554](#)
- cellWallWidth
 - tissue::Tissue, [555](#)
- cend
 - util::range, [433](#)
- center3d
 - algorithms::TriangleSurface, [606](#)
- center_blending
 - tissue::Tissue, [555](#)
- centerPosition
 - algorithms::TriangleSurface, [606](#)
- centroid
 - geometry, [115](#)
- changed
 - util::GraphInset::Data, [267](#)
- changedExitCode
 - Model, [370](#)
- changeFilename
 - util::WatchDog, [975](#)
- changeSourceColor
 - util::GraphInset, [322](#)
- changeSourceOffset
 - util::GraphInset, [323](#)
- changeSourceWidth
 - util::GraphInset, [323](#)
- check
 - util::GraphInset, [315](#)
- CHECK_EQUAL
 - test.h, [1054](#)
- CHECK_FLOAT
 - test.h, [1054](#)
- CHECK_FLOAT_DIFF
 - test.h, [1054](#)
- CHECK_NEIGHBORS_BIGRAPH
 - graph.vvh, [1052](#)
- CHECK_NEIGHBORS_GRAPH

- graph.vvh, [1052](#)
- CHECK_NOTEQUAL
 - test.h, [1055](#)
- CHECK_SYMMETRIC_GRAPH
 - graph.vvh, [1052](#)
- CHECK_ZERO
 - test.h, [1055](#)
- checkNextField
 - storage::old::VVEStorage_
BINReader, [869](#)
 - storage::VVEStorage, [857](#)
 - storage::VVEStorage_BINReader,
[877](#)
 - storage::VVEStorage_BINWriter,
[887](#)
 - storage::VVEStorage_XMLReader,
[894](#)
 - storage::VVEStorage_XMLWriter,
[902](#)
- circ_neighbor_iterator
 - graph::VVGraph, [916](#)
- circ_neighbor_iterator1
 - graph::VVBIGraph, [713](#)
- circ_neighbor_iterator1_range
 - graph::VVBIGraph, [713](#)
- circ_neighbor_iterator2
 - graph::VVBIGraph, [713](#)
- circ_neighbor_iterator2_range
 - graph::VVBIGraph, [714](#)
- circ_neighbor_iterator_range
 - graph::VVGraph, [916](#)
- CircIterator
 - util::CircIterator, [236](#)
- clamp
 - util, [160](#)
 - util::Texture1D, [511](#)
 - util::Texture2D, [517](#)
- cleanData
 - util::GraphInset, [316](#)
- clear
 - algorithms::TriangleSurface, [607](#)
 - graph::Edge, [282](#)
 - graph::VVBIGraph, [732](#), [733](#)
 - graph::VVGraph, [925](#), [926](#)
 - storage::VVEStorage, [857](#)
 - util::GraphInset, [316](#)
 - vvcomplex::VVComplex, [812](#)
- clearLastError
 - storage::VVEStorage, [857](#)
- clearOldGraphs
 - vvcomplex::VVComplex, [813](#)
- closed
 - util::GraphInset, [323](#)
- closeGraph
 - util::GraphInset, [316](#)
- ClosestMidAlgoParams
 - tissue::ClosestMidAlgoParams, [239](#)
- ClosestWallAlgoParams
 - tissue::ClosestWallAlgoParams, [241](#)
- cofactor
 - util::Matrix, [354](#)
- Color
 - bspline_tissue_model.h, [1019](#)
 - util::Color, [244](#)
- colorBegin
 - tissue::Tissue, [556](#)
- colorCenter
 - tissue::Tissue, [556](#)
- colorEnd
 - tissue::Tissue, [557](#)
- complex_factory, [86](#)
 - connectJunction, [87](#)
 - hex_grid, [88](#)
 - objreader, [91](#), [92](#)
 - square_grid, [98](#)
 - squareFiller, [101](#)
- complex_factory::HexFiller, [327](#)
- complex_factory::ObjReaderError, [390](#)
 - operator bool, [391](#)
 - string, [391](#)
 - Type, [391](#)
 - type, [391](#)
- complex_factory::SquareFiller, [504](#)
- complex_graph
 - vvcomplex, [180](#)
 - vvcomplex::VVComplex, [797](#)
- configOpenGL
 - Viewer, [682](#)
- ConjGradMaxSteps
 - solver::Solver, [496](#)
- ConjGradTol
 - solver::Solver, [496](#)
- ConjGradTolType
 - solver::Solver, [496](#)
- conjugate
 - geometry::Quaternion, [425](#)
- connectFromJunctions
 - vvcomplex::VVComplex, [813](#)
- connectJunction
 - complex_factory, [87](#)

- const_cell_edge
 - vvcomplex, 180
 - vvcomplex::VVComplex, 797
- const_cell_junction_edge
 - vvcomplex, 181
 - vvcomplex::VVComplex, 798
- const_circ_neighbor_iterator
 - graph::VVGraph, 917
- const_circ_neighbor_iterator1
 - graph::VVBIGraph, 714
- const_circ_neighbor_iterator1_range
 - graph::VVBIGraph, 714
- const_circ_neighbor_iterator2
 - graph::VVBIGraph, 714
- const_circ_neighbor_iterator2_range
 - graph::VVBIGraph, 715
- const_circ_neighbor_iterator_range
 - graph::VVGraph, 917
- const_edge1_t
 - graph::VVBIGraph, 715
- const_edge2_t
 - graph::VVBIGraph, 715
- const_edge_t
 - graph::VVGraph, 917
- const_iterator
 - graph::VVGraph, 917
- const_iterator1
 - graph::VVBIGraph, 715
 - vvcomplex::VVComplexGraph, 837
- const_iterator2
 - graph::VVBIGraph, 716
 - vvcomplex::VVComplexGraph, 837
- const_junction_cell_edge
 - vvcomplex, 181
 - vvcomplex::VVComplex, 798
- const_neighbor1_found_t
 - graph::VVBIGraph, 716
- const_neighbor2_found_t
 - graph::VVBIGraph, 716
- const_neighbor_found_t
 - graph::VVGraph, 917
- const_neighbor_iterator
 - graph::VVGraph, 918
- const_neighbor_iterator1
 - graph::VVBIGraph, 717
- const_neighbor_iterator1_range
 - graph::VVBIGraph, 717
- const_neighbor_iterator2
 - graph::VVBIGraph, 717
- const_neighbor_iterator2_range
 - graph::VVBIGraph, 717
- const_neighbor_iterator_range
 - graph::VVGraph, 918
- const_range_vertex1
 - graph::VVBIGraph, 718
- const_range_vertex2
 - graph::VVBIGraph, 718
- const_wall
 - vvcomplex, 181
 - vvcomplex::VVComplex, 798
- ConstNbPartials
 - solver::Solver, 497
- contains
 - graph::VVBIGraph, 734
 - graph::VVGraph, 926
- content
 - graph::Vertex, 665
- content_t
 - graph::Edge, 280
 - graph::Vertex, 661
 - graph::WeakVertex, 980
- Contour
 - util::Contour, 251
- contour
 - algorithms::TriangleGrowth, 589
 - algorithms::TriangleSurface, 607
- contourColor
 - tissue::Tissue, 557
 - util::GraphInset, 323
- contourOffset
 - util::GraphInset, 323
- contourWidth
 - util::GraphInset, 323
- convert_type
 - storage, 140
- convertHSVtoRGB
 - util, 160
- copy_symmetric
 - graph, 123
- count
 - graph::CountedContent, 264
 - graph::VVBIGraph, 734
 - graph::VVGraph, 927
- counted_content_t
 - graph::Vertex, 661
- CRAvgCPU
 - solver::Solver, 497
- createViewer
 - Viewer, 683
- CRIncDt

- solver::Solver, [497](#)
- CRMinCPU
 - solver::Solver, [497](#)
- cross
 - util::Vector, [628](#)
- CRResDt
 - solver::Solver, [497](#)
- cube
 - shape, [126](#)
- current_div
 - cell_system::CellSystem, [222](#)
- current_method
 - solver::Solver, [498](#)
- cylinder
 - shape, [126](#)
- damping
 - cell_system::CellSystem, [222](#)
- data
 - util::GraphInset, [323](#)
 - util::Matrix, [348](#)
 - util::Vector, [628](#)
- decals
 - util::Texture1D, [511](#)
 - util::Texture2D, [518](#)
- deleteCell
 - vvcomplex::VVComplex, [815](#)
 - vvcomplex::VVComplexGraph, [845](#)
- deleteCellInGraphs
 - vvcomplex::VVComplex, [815](#)
- deleteJunction
 - vvcomplex::VVComplex, [816](#)
- deleteJunctionExtra
 - vvcomplex::VVComplex, [816](#)
- demangle
 - util, [161](#)
- det
 - util::Matrix, [354](#)
- difference_type
 - util::SelectMemberIterator, [450](#)
- direction
 - cell_
 - system::CellSystemDivisionParams, [230](#)
- disk
 - shape, [127](#)
- divide
 - util::Vector, [646](#)
- divide_at_u1
 - vvcomplex::DivisionData, [273](#)
- divide_at_v1
 - vvcomplex::DivisionData, [273](#)
- divideCell
 - tissue::Tissue, [529–533](#)
 - vvcomplex::VVComplex, [817–820](#)
 - vvcomplex::VVComplexGraph, [845](#)
- division_data
 - tissue::Tissue, [525](#)
 - vvcomplex::VVComplex, [799](#)
 - vvcomplex::VVComplexGraph, [837](#)
- division_rules
 - cell_system::CellSystem, [222](#)
- division_rules_t
 - cell_system::CellSystem, [212](#)
- DivisionData
 - vvcomplex::DivisionData, [271, 272](#)
- doc/ Directory Reference, [60](#)
- doc/examples/ Directory Reference, [61](#)
- doc/examples/context_menu.cpp, [985](#)
- doc/examples/hello_world.cpp, [986](#)
- doc/examples/select.cpp, [987](#)
- doc/examples/simple_model.cpp, [988](#)
- domElement
 - Viewer, [683](#)
- Draw
 - util::Shapes::BSplineSurface, [196](#)
- draw
 - bspline_tissue_model::TissueModel, [576](#)
 - Model, [370, 371](#)
 - tissue_model::TissueModel, [565](#)
 - util::GraphInset, [317](#)
 - Viewer, [684](#)
- drawBorders
 - tissue::Tissue, [558](#)
- drawCell
 - tissue::Tissue, [534, 535](#)
- drawCellContour
 - tissue::Tissue, [537](#)
- drawFrame
 - util::GraphInset, [318](#)
- drawInsides
 - tissue::Tissue, [558](#)
- drawLight
 - Viewer, [684](#)
- drawNeighborhood
 - bspline_tissue_model::TissueModel, [584](#)
 - tissue_model::TissueModel, [570](#)
- drawSimplifiedCell

- tissue::Tissue, [538](#)
- drawSymetricVVGrahp
 - algorithms, [75](#)
- drawVVBIGraphConnections1
 - algorithms, [75](#)
- drawVVBIGraphConnections2
 - algorithms, [76](#)
- drawVVGrahpConnections
 - algorithms, [76](#)
- drawWalledCell
 - tissue::Tissue, [539](#), [540](#)
- drawWithName
 - util::GraphInset, [318](#)
- drawWithNames
 - bspline_tissue_model::TissueModel, [577](#)
 - Model, [371](#)
 - tissue_model::TissueModel, [566](#)
 - Viewer, [685](#)
- dt
 - bspline_tissue_model::TissueModel, [585](#)
 - cell_system::CellSystem, [223](#)
 - solver::Solver, [498](#)
- Dx
 - solver::Solver, [498](#)
- dx
 - tissue::ShortWallAlgoParams, [461](#)
- Edge
 - graph::Edge, [281](#)
- edge
 - graph::VVBIGraph, [735](#), [736](#)
 - graph::VVGrahp, [927](#), [928](#)
- edge1_t
 - graph::VVBIGraph, [718](#)
- edge2_t
 - graph::VVBIGraph, [718](#)
- edge_identity_t
 - graph, [122](#)
- edge_list1_t
 - graph::VVBIGraph, [719](#)
- edge_list2_t
 - graph::VVBIGraph, [719](#)
- edge_list_t
 - graph::VVGrahp, [918](#)
 - graph::VVGrahp::neighbor_t, [388](#)
- edge_t
 - graph::VVGrahp, [918](#)
- EdgeInternals
 - solver::Solver, [473](#)
- edges
 - graph::VVBIGraph::single_neighborhood_t, [464](#)
 - graph::VVGrahp::single_neighborhood_t, [466](#)
- empty
 - graph::VVBIGraph, [736](#), [737](#)
 - graph::VVGrahp, [929](#)
- end
 - graph::VVGrahp, [930](#)
 - util::CircIterator, [237](#)
 - util::range, [433](#)
 - util::Vector, [629](#)
- end_vertex1
 - graph::VVBIGraph, [737](#)
- end_vertex2
 - graph::VVBIGraph, [738](#)
- endCompound
 - storage::old::VVEStorage_BINReader, [869](#)
 - storage::VVEStorage, [858](#)
 - storage::VVEStorage_BINReader, [877](#)
 - storage::VVEStorage_BINWriter, [887](#)
 - storage::VVEStorage_XMLReader, [894](#)
 - storage::VVEStorage_XMLWriter, [902](#)
- EndModel
 - bspline_tissue_model.h, [1019](#)
- endReference
 - storage::old::VVEStorage_BINReader, [869](#)
 - storage::VVEStorage, [858](#)
 - storage::VVEStorage_BINReader, [878](#)
 - storage::VVEStorage_BINWriter, [887](#)
 - storage::VVEStorage_XMLReader, [894](#)
 - storage::VVEStorage_XMLWriter, [902](#)
- endTime
 - algorithms::TriangleGrowth, [590](#)
- epsilon
 - bspline_tissue_model, [79](#)
 - cell_system::CellSystemDivisionParams,

- 231
- tissue_model, 153
- erase
 - graph::VVBIGraph, 738, 739
 - graph::VVGraph, 930, 931
- eraseAllEdges
 - graph::VVBIGraph, 740–742
 - graph::VVGraph, 932, 933
- eraseEdge
 - graph::VVBIGraph, 743–745
 - graph::VVGraph, 934, 935
- errorMsg
 - algorithms::TriangleGrowth, 590
- Euler
 - solver, 132
- EulerDt
 - solver::Solver, 498
- EXPORT_COMPLEX_EDGES
 - algorithms/complex.h, 995
- EXPORT_COMPLEX_EDGES_ -
 - PREFIX
 - algorithms/complex.h, 995
- EXPORT_COMPLEX_GRAPHS
 - algorithms/complex.h, 996
- EXPORT_COMPLEX_GRAPHS_ -
 - PREFIX
 - algorithms/complex.h, 996
- EXPORT_COMPLEX_TYPES
 - algorithms/complex.h, 996
- EXPORT_COMPLEX_TYPES_PREFIX
 - algorithms/complex.h, 997
- EXPORT_COMPLEX_VERTICES
 - algorithms/complex.h, 997
- EXPORT_COMPLEX_VERTICES_ -
 - PREFIX
 - algorithms/complex.h, 997
- faces
 - algorithms::TriangleSurface, 612
- factory, 102
 - hex_grid, 102
 - square_grid, 105
- field
 - storage::VVEStorage, 858–861
- filename
 - storage::old::VVEStorage_ -
 - BINReader, 869
 - storage::VVEStorage, 861
 - storage::VVEStorage_BINReader, 878
 - storage::VVEStorage_BINWriter, 888
 - storage::VVEStorage_XMLReader, 895
 - storage::VVEStorage_XMLWriter, 903
- FileObject
 - util::FileObject, 290, 291
- fileObjects
 - util::WatchDog, 977
- fileRegister
 - Model, 372
- fileUnregister
 - Model, 372
- fileVersion
 - storage::VVEStorage, 862
- filter
 - util::Texture1D, 511
 - util::Texture2D, 518
- finalizeDraw
 - Model, 372
- finalizePrint
 - Model, 372
- find
 - graph::VVBIGraph, 745, 746
 - graph::VVGraph, 936
- find_type
 - storage, 140
- FindCenter
 - vvcomplex, 182
- findCenter
 - vvcomplex, 182
- findDivisionPoints
 - cell_system, 81, 83
 - tissue, 147, 149, 150
 - vvcomplex, 183
- findIn
 - graph::VVBIGraph, 747, 748
 - graph::VVGraph, 936, 937
- findInVertex
 - graph::VVBIGraph, 748–750
 - graph::VVGraph, 937, 938
- FindOppositeWall
 - vvcomplex, 183
- FindPartials
 - solver::Solver, 475
- findVertex
 - graph::VVBIGraph, 751, 752
 - graph::VVGraph, 940
- FindWallMin

- vvcomplex, 185
- first
 - util::repair, 436
- Fixedpoint
 - solver, 132
- FixedPointDt
 - solver::Solver, 498
- FixedPointMaxSteps
 - solver::Solver, 499
- FixedPointTol
 - solver::Solver, 499
- FixedPointTolType
 - solver::Solver, 499
- flag
 - graph::VVBiGraph, 754
 - graph::VVGraph, 942
- flagged
 - graph::VVBiGraph, 754, 755
 - graph::VVBiGraph::single_ - neighborhood_t, 464
 - graph::VVGraph, 943
 - graph::VVGraph::single_ - neighborhood_t, 466
- FOR_ALL_TYPEIDS
 - types.h, 1045
- FOR_ALL_TYPES
 - types.h, 1046
- FOR_ALL_TYPES_NOSTRING
 - types.h, 1046
- forall
 - forall.h, 1069
- forall.h
 - forall, 1069
 - forall_named, 1071
 - forall_range, 1071
 - forall_reverse, 1072
- forall_named
 - forall.h, 1071
- forall_range
 - forall.h, 1071
- forall_reverse
 - forall.h, 1072
- force
 - cell_system::CellSystemJunction, 234
- found
 - graph::VVBiGraph::search_result_t, 444
 - graph::VVGraph::search_result_t, 440
- frame_stack
 - storage::VVEStorage_BINReader, 884
 - storage::VVEStorage_BINWriter, 891
- Function
 - util::Function, 297
- g
 - util::Color, 245, 246
- gaussRan
 - util, 161, 162
- geometry, 111
 - barycentricToCartesian, 114
 - barycentricToCartesian_matrix, 114
 - cartesianToBarycentric, 114
 - cartesianToBarycentric_matrix, 114
 - centroid, 115
 - lineLineIntersection, 115
 - lineSegmentIntersection, 115
 - lineTriangleIntersection, 115
 - Matrix2d, 113
 - Matrix3d, 113
 - Matrix4d, 113
 - planeLineIntersection, 116
 - Point2d, 113
 - Point3d, 114
 - Point4d, 114
 - pointInPolygon, 116
 - pointInTriangle, 117
 - polygonArea, 117
 - projectPointOnLine, 118
 - projectPointOnPlane, 118
 - projectPointOnTriangle, 119
 - segmentSegmentIntersection, 119
 - triangleArea, 120
- geometry::Quaternion, 420
 - angle, 424
 - axis, 424
 - conjugate, 425
 - inverse, 425
 - inverseRotate, 425
 - operator*, 425
 - operator*=: 425, 426
 - operator+, 426
 - operator+=: 426
 - operator/, 426
 - operator/=: 427
 - operator=, 427
 - Quaternion, 422, 423

- rotate, 427
- setAxisAngle, 428
- setMatrix, 428
- w, 429
- get_cell
 - vvcomplex::VVComplexGraph, 847
- get_junction
 - vvcomplex::VVComplexGraph, 848
- get_vertex1
 - graph::VVBIGraph, 755
- get_vertex2
 - graph::VVBIGraph, 756
- getCellCenterColor
 - bspline_tissue_model::TissueModel, 578
 - tissue_model::TissueModel, 566
- getCellColor
 - bspline_tissue_model::TissueModel, 578
 - tissue_model::TissueModel, 566
- getCellPinchingParams
 - tissue::Tissue, 543
- getClosestMidAlgoParams
 - tissue::Tissue, 543
- getCloseWallAlgoParams
 - tissue::Tissue, 544
- getColor
 - util::Palette, 395
- getFilename
 - util::FileObject, 291
- getMaterial
 - util::Materials, 335
- getMax
 - util::Contour, 252
 - util::Function, 298
- getMin
 - util::Contour, 252
 - util::Function, 298
- getShortWallAlgoParams
 - tissue::Tissue, 545
- glerrorcheck.h
 - OPENGL_ERROR_CHECK, 1074
- graph, 121
 - copy_symmetric, 123
 - edge_identity_t, 122
 - vertex_counter, 123
 - vertex_identity_t, 122
- graph.vvh
 - CHECK_NEIGHBORS_BIGRAPH, 1052
 - CHECK_NEIGHBORS_GRAPH, 1052
 - CHECK_SYMMETRIC_GRAPH, 1052
 - graph::_EmptyEdgeContent, 187
 - graph::Arc, 188
 - ~Arc, 190
 - Arc, 190
 - identity_t, 189
 - inv, 191
 - isNull, 191
 - operator bool, 191
 - operator edge_t, 191
 - operator*, 191
 - operator-, 192
 - operator->, 192
 - source, 192
 - sync, 192
 - target, 193
 - graph::CountedContent, 264
 - count, 264
 - graph::Edge, 278
 - _content, 287
 - _source, 287
 - _target, 287
 - clear, 282
 - content_t, 280
 - Edge, 281
 - identity_t, 280
 - isNull, 282
 - operator bool, 282
 - operator<, 284
 - operator<=, 284
 - operator>, 285
 - operator>=, 285
 - operator*, 283
 - operator->, 283
 - operator->*, 283, 284
 - operator=, 284
 - operator==, 285
 - pointer, 280
 - serialize, 286
 - source, 286
 - target, 286
 - graph::Vertex, 657
 - ~Vertex, 664
 - _content, 672
 - acquire, 665
 - cache, 673
 - cache_source, 673

- content, 665
- content_t, 661
- counted_content_t, 661
- id, 665
- identity_t, 661
- isNull, 666
- isWeakRef, 666
- null, 673
- num, 667
- operator bool, 667
- operator<, 669
- operator<=, 669
- operator>, 670
- operator>=, 671
- operator*, 668
- operator->, 668
- operator->*, 668
- operator=, 669
- operator==, 670
- pointer, 661
- real_pointer, 661
- release, 671
- serialize, 672
- Vertex, 662–664
- weakRef, 672
- graph::VVBiGraph, 698
 - any_vertex1, 729
 - any_vertex2, 729
 - anyIn, 730
 - begin_vertex1, 731
 - begin_vertex2, 731
 - circ_neighbor_iterator1, 713
 - circ_neighbor_iterator1_range, 713
 - circ_neighbor_iterator2, 713
 - circ_neighbor_iterator2_range, 714
 - clear, 732, 733
 - const_circ_neighbor_iterator1, 714
 - const_circ_neighbor_iterator1_range, 714
 - const_circ_neighbor_iterator2, 714
 - const_circ_neighbor_iterator2_range, 715
 - const_edge1_t, 715
 - const_edge2_t, 715
 - const_iterator1, 715
 - const_iterator2, 716
 - const_neighbor1_found_t, 716
 - const_neighbor2_found_t, 716
 - const_neighbor_iterator1, 717
 - const_neighbor_iterator1_range, 717
 - const_neighbor_iterator2, 717
 - const_neighbor_iterator2_range, 717
 - const_range_vertex1, 718
 - const_range_vertex2, 718
 - contains, 734
 - count, 734
 - edge, 735, 736
 - edge1_t, 718
 - edge2_t, 718
 - edge_list1_t, 719
 - edge_list2_t, 719
 - empty, 736, 737
 - end_vertex1, 737
 - end_vertex2, 738
 - erase, 738, 739
 - eraseAllEdges, 740–742
 - eraseEdge, 743–745
 - find, 745, 746
 - findIn, 747, 748
 - findInVertex, 748–750
 - findVertex, 751, 752
 - flag, 754
 - flagged, 754, 755
 - get_vertex1, 755
 - get_vertex2, 756
 - graph_id, 786
 - iAnyIn, 756
 - iEmpty, 757
 - in_edges1_t, 719
 - in_edges2_t, 719
 - ineighbor_iterator1, 720
 - ineighbor_iterator1_range, 720
 - ineighbor_iterator2, 720
 - ineighbor_iterator2_range, 721
 - iNeighbors, 758
 - insert, 759, 760
 - insertEdge, 761
 - insertInEdge, 762
 - int_const_neighbor_iterator1, 721
 - int_const_neighbor_iterator2, 721
 - int_neighbor_iterator1, 721
 - int_neighbor_iterator2, 722
 - iterator1, 722
 - iterator2, 722
 - iValence, 763
 - lookup1, 786
 - lookup1_t, 723
 - lookup2, 786
 - lookup2_t, 723
 - nb_vertices1, 763

- nb_vertices2, [764](#)
- neighbor1_found_t, [723](#)
- neighbor1_t, [723](#)
- neighbor2_found_t, [724](#)
- neighbor2_t, [724](#)
- neighbor_iterator1, [724](#)
- neighbor_iterator1_range, [724](#)
- neighbor_iterator2, [725](#)
- neighbor_iterator2_range, [725](#)
- neighborhood1, [787](#)
- neighborhood1_t, [725](#)
- neighborhood1_value_type, [725](#)
- neighborhood2, [788](#)
- neighborhood2_t, [726](#)
- neighborhood2_value_type, [726](#)
- neighbors, [764–767](#)
- nextTo, [768](#)
- no_vertex1, [769](#)
- no_vertex2, [769](#)
- operator=, [770](#)
- operator==, [771](#)
- prevTo, [773](#)
- range_vertex1, [726](#)
- range_vertex2, [726](#)
- reference, [774](#)
- replace, [775](#)
- single_neighborhood1_t, [727](#)
- single_neighborhood2_t, [727](#)
- size, [776](#)
- size_type, [727](#)
- source, [777](#)
- spliceAfter, [777](#), [778](#)
- spliceBefore, [778](#), [779](#)
- swap, [780](#)
- target, [780](#), [781](#)
- undoInEdge, [781](#)
- updateEdgeCache, [782](#)
- updateVertexCache, [783](#)
- valence, [783](#), [784](#)
- vertex1_t, [728](#)
- vertex2_t, [728](#)
- vertices1, [784](#), [785](#)
- vertices2, [785](#)
- VVBiGraph, [728](#)
- graph::VVBiGraph::search_result_t, [442](#)
 - found, [444](#)
 - it, [445](#)
 - neighborhood, [445](#)
 - operator bool, [444](#)
 - search_result_t, [443](#)
- graph::VVBiGraph::single_neighborhood_t, [462](#)
 - edges, [464](#)
 - flagged, [464](#)
 - in_edges, [464](#)
 - operator==, [463](#)
- graph::VVGraph, [907](#)
 - any, [922](#)
 - anyIn, [923](#)
 - arc, [924](#)
 - arc_t, [916](#)
 - begin, [924](#), [925](#)
 - circ_neighbor_iterator, [916](#)
 - circ_neighbor_iterator_range, [916](#)
 - clear, [925](#), [926](#)
 - const_circ_neighbor_iterator, [917](#)
 - const_circ_neighbor_iterator_range, [917](#)
 - const_edge_t, [917](#)
 - const_iterator, [917](#)
 - const_neighbor_found_t, [917](#)
 - const_neighbor_iterator, [918](#)
 - const_neighbor_iterator_range, [918](#)
 - contains, [926](#)
 - count, [927](#)
 - edge, [927](#), [928](#)
 - edge_list_t, [918](#)
 - edge_t, [918](#)
 - empty, [929](#)
 - end, [930](#)
 - erase, [930](#), [931](#)
 - eraseAllEdges, [932](#), [933](#)
 - eraseEdge, [934](#), [935](#)
 - find, [936](#)
 - findIn, [936](#), [937](#)
 - findInVertex, [937](#), [938](#)
 - findVertex, [940](#)
 - flag, [942](#)
 - flagged, [943](#)
 - graph_id, [970](#)
 - iAnyIn, [943](#)
 - iEmpty, [944](#)
 - in_edges_t, [918](#)
 - ineighbor_iterator, [919](#)
 - ineighbor_iterator_range, [919](#)
 - iNeighbors, [944](#)
 - insert, [945](#), [946](#)
 - insertEdge, [947](#)
 - insertEdges, [948](#)
 - insertInEdge, [948](#)

- int_const_neighbor_iterator, 919
- int_neighbor_iterator, 919
- iterator, 920
- iValence, 949
- lookup, 970
- lookup_t, 920
- neighbor_found_t, 920
- neighbor_iterator, 920
- neighbor_iterator_range, 920
- neighborhood, 971
- neighborhood_t, 921
- neighborhood_value_type, 921
- neighbors, 949–952
- nextTo, 952
- operator=, 953
- operator==, 954
- prevTo, 956
- reference, 957
- replace, 958
- size, 959
- size_type, 921
- source, 960
- spliceAfter, 960, 961
- spliceBefore, 962, 963
- store_vertices, 964
- subgraph, 965
- swap, 967
- target, 967
- undoInEdge, 968
- updateEdgeCache, 968
- updateVertexCache, 969
- valence, 969
- value_type, 921
- vertex_t, 921
- VVGraph, 922
- graph::VVGraph::neighbor_t, 387
 - edge_list_t, 388
 - neighbor_t, 388
 - operator==, 388
 - target, 389
- graph::VVGraph::search_result_t, 438
 - found, 440
 - it, 440
 - neighborhood, 440
 - operator bool, 440
 - search_result_t, 439
- graph::VVGraph::single_neighborhood_t, 465
 - edges, 466
 - flagged, 466
 - in_edges, 466
 - operator==, 466
- graph::WeakVertex, 979
 - content_t, 980
 - identity_t, 980
 - isNull, 982
 - operator=, 982
 - pointer, 980
 - strong_ref, 981
 - WeakVertex, 981, 982
- graph_id
 - graph::VVBIGraph, 786
 - graph::VVGraph, 970
- graphClosed
 - util::GraphInset, 319
- GraphInset
 - util::GraphInset, 312
- greater
 - util, 162
- grow
 - util::Tensor, 506
- growthStartTime
 - bspline_tissue_model::TissueModel, 585
- guarded
 - util::WatchDog, 976
- helpMenu
 - Viewer, 685
- helpString
 - Model, 373
 - Viewer, 685
- hex_grid
 - complex_factory, 88
 - factory, 102
- i
 - util::Vector, 629, 630
- iAnyIn
 - graph::VVBIGraph, 756
 - graph::VVGraph, 943
- id
 - algorithms::TriangleSurface::Cell, 205
 - algorithms::TriangleSurface::Vertex, 656
 - graph::Vertex, 665
 - storage::TypeTraits, 614
- identity
 - util::Matrix, 348

- identity_t
 - graph::Arc, [189](#)
 - graph::Edge, [280](#)
 - graph::Vertex, [661](#)
 - graph::WeakVertex, [980](#)
- iEmpty
 - graph::VVBIGraph, [757](#)
 - graph::VVGraph, [944](#)
- ignore
 - storage::VVEStorage, [862](#)
 - storage::VVEStorage_BINReader, [878](#)
- IMPORT_COMPLEX_EDGES
 - algorithms/complex.h, [998](#)
- IMPORT_COMPLEX_GRAPHS
 - algorithms/complex.h, [998](#)
- IMPORT_COMPLEX_MODEL
 - algorithms/complex.h, [998](#)
- IMPORT_COMPLEX_TYPES
 - algorithms/complex.h, [998](#)
- IMPORT_COMPLEX_VERTICES
 - algorithms/complex.h, [999](#)
- in_edges
 - graph::VVBIGraph::single_neighborhood_t, [464](#)
 - graph::VVGraph::single_neighborhood_t, [466](#)
- in_edges1_t
 - graph::VVBIGraph, [719](#)
- in_edges2_t
 - graph::VVBIGraph, [719](#)
- in_edges_t
 - graph::VVGraph, [918](#)
- index
 - Viewer, [686](#)
- ineighbor_iterator
 - graph::VVGraph, [919](#)
- ineighbor_iterator1
 - graph::VVBIGraph, [720](#)
- ineighbor_iterator1_range
 - graph::VVBIGraph, [720](#)
- ineighbor_iterator2
 - graph::VVBIGraph, [720](#)
- ineighbor_iterator2_range
 - graph::VVBIGraph, [721](#)
- ineighbor_iterator_range
 - graph::VVGraph, [919](#)
- iNeighbors
 - graph::VVBIGraph, [758](#)
 - graph::VVGraph, [944](#)
- init
 - algorithms::TriangleGrowth, [590](#)
 - Viewer, [686](#)
- initDraw
 - bspline_tissue_model::TissueModel, [578](#)
 - Model, [373](#), [374](#)
 - tissue_model::TissueModel, [567](#)
- initFromDOMEElement
 - Viewer, [686](#)
- InitialDt
 - solver::Solver, [499](#)
- initialize
 - bspline_tissue_model::TissueModel, [578](#)
 - solver::Solver, [476](#)
 - tissue_model::TissueModel, [567](#)
- initialized
 - solver::Solver, [499](#)
 - util::GraphInset, [324](#)
- initPrint
 - Model, [374](#)
- initTissue
 - bspline_tissue_model::TissueModel, [579](#)
- insert
 - graph::VVBIGraph, [759](#), [760](#)
 - graph::VVGraph, [945](#), [946](#)
- insertAction
 - Model, [374](#)
- insertActions
 - Model, [375](#)
- insertAfter
 - algorithms, [76](#)
- insertBefore
 - algorithms, [77](#)
- insertEdge
 - graph::VVBIGraph, [761](#)
 - graph::VVGraph, [947](#)
- insertEdges
 - graph::VVGraph, [948](#)
- insertInEdge
 - graph::VVBIGraph, [762](#)
 - graph::VVGraph, [948](#)
- int_const_neighbor_iterator
 - graph::VVGraph, [919](#)
- int_const_neighbor_iterator1
 - graph::VVBIGraph, [721](#)
- int_const_neighbor_iterator2
 - graph::VVBIGraph, [721](#)

- int_neighbor_iterator
 - graph::VVGraph, [919](#)
- int_neighbor_iterator1
 - graph::VVBIGraph, [721](#)
- int_neighbor_iterator2
 - graph::VVBIGraph, [722](#)
- interfaceWall
 - vvcomplex::VVComplex, [822](#)
 - vvcomplex::VVComplexGraph, [848](#)
- interfaceWallSpan
 - vvcomplex::VVComplex, [822](#)
 - vvcomplex::VVComplexGraph, [848](#)
- inv
 - graph::Arc, [191](#)
- inverse
 - geometry::Quaternion, [425](#)
 - util::Matrix, [355](#)
- inverseRotate
 - geometry::Quaternion, [425](#)
- IO_ERROR
 - storage, [139](#)
- isInitIterator
 - util::CircIterator, [237](#)
- isLoaded
 - util::Parms, [410](#)
- isNull
 - graph::Arc, [191](#)
 - graph::Edge, [282](#)
 - graph::Vertex, [666](#)
 - graph::WeakVertex, [982](#)
- isStrictConversion
 - storage, [140](#)
- IsTextured
 - util::Shapes::BSplineSurface, [200](#)
- isWeakRef
 - graph::Vertex, [666](#)
- it
 - graph::VVBIGraph::search_result_t, [445](#)
 - graph::VVGraph::search_result_t, [440](#)
 - util::SelectMemberIterator, [455](#)
- iterator
 - graph::VVGraph, [920](#)
 - util::GraphInset, [311](#)
- iterator1
 - graph::VVBIGraph, [722](#)
 - vvcomplex::VVComplexGraph, [837](#)
- iterator2
 - graph::VVBIGraph, [722](#)
- vvcomplex::VVComplexGraph, [838](#)
- iterator_category
 - util::SelectMemberIterator, [450](#)
- iValence
 - graph::VVBIGraph, [763](#)
 - graph::VVGraph, [949](#)
- j
 - util::Vector, [630](#)
- junction
 - vvcomplex, [181](#)
 - vvcomplex::DivisionData, [271](#)
 - vvcomplex::VVComplex, [799](#)
- junction_cell_edge
 - vvcomplex, [181](#)
 - vvcomplex::VVComplex, [800](#)
- k
 - util::Vector, [631](#)
- ke
 - cell_system::CellSystem, [223](#)
- keepAllData
 - util::GraphInset, [324](#)
- keyPressEvent
 - Viewer, [688](#)
- ki
 - cell_system::CellSystem, [223](#)
- l
 - util::Vector, [631](#), [632](#)
- label
 - cell_system::CellSystem, [212](#)
 - cell_system::CellSystemCell, [228](#)
- label_change_rules
 - cell_system::CellSystem, [223](#)
- label_change_rules_t
 - cell_system::CellSystem, [212](#)
- label_names
 - cell_system::CellSystem, [223](#)
- label_numbers
 - cell_system::CellSystem, [224](#)
- label_t
 - cell_system, [81](#)
- lastError
 - storage::VVEStorage, [862](#)
- lastErrorString
 - storage::VVEStorage, [862](#)
- leaf
 - bspline_tissue_model::TissueModel, [585](#)

- leafs
 - bspline_tissue_model::TissueModel, 585
- left_label
 - cell_system::DivisionParams, 276
- length
 - util::Contour, 252
- less
 - util, 162
- lineLineIntersection
 - geometry, 115
- linesColor
 - util::GraphInset, 324
- lineSegmentIntersection
 - geometry, 115
- linesMinMaxView
 - util::GraphInset, 324
- linesOffset
 - util::GraphInset, 324
- linesWidth
 - util::GraphInset, 324
- lineTriangleIntersection
 - geometry, 115
- Load
 - util::Shapes::BSplineSurface, 200
- loadingSnapshot
 - Model, 375
- loadSnapshot
 - Model, 375, 376
- lookup
 - graph::VVGraph, 970
- lookup1
 - graph::VVBIGraph, 786
- lookup1_t
 - graph::VVBIGraph, 723
- lookup2
 - graph::VVBIGraph, 786
- lookup2_t
 - graph::VVBIGraph, 723
- lookup_t
 - graph::VVGraph, 920
- mainWindow
 - Viewer, 688
- make_range
 - util, 162, 163
- map
 - util::Matrix, 355–360
 - util::Vector, 646–648
- mass
 - cell_system::CellSystem, 224
- Mat
 - solver::Solver, 473
- Materials
 - util::Materials, 334
- Matrix
 - util::Matrix, 345–347
- Matrix2d
 - geometry, 113
- Matrix3d
 - geometry, 113
- Matrix4d
 - geometry, 113
- max
 - util::Vector, 649
- MAX_COMPONENT
 - solver, 132
- MaxDt
 - solver::Solver, 500
- maximum
 - util, 163
- MEAN_COMPONENT
 - solver, 132
- menuActionInserted
 - Model, 376
- menuActionRemoved
 - Model, 376
- menuBar
 - Viewer, 689
- menuItem
 - Model, 376
- mergeCells
 - vvcomplex::VVComplex, 822
 - vvcomplex::VVComplexGraph, 849
- method
 - solver::Solver, 500
- Midpoint
 - solver, 132
- MidPointDt
 - solver::Solver, 500
- min
 - util::Vector, 649
- minCellWall
 - cell_ -
 - system::CellSystemDivisionParams, 231
- MinDt
 - solver::Solver, 500
- minimum
 - util, 163

- Model, 362
 - ~Model, 368
 - actions, 368
 - addAction, 368
 - addActions, 368
 - addMenuItem, 369
 - animationPeriod, 370
 - arguments, 370
 - changedExitCode, 370
 - draw, 370, 371
 - drawWithNames, 371
 - fileRegister, 372
 - fileUnregister, 372
 - finalizeDraw, 372
 - finalizePrint, 372
 - helpString, 373
 - initDraw, 373, 374
 - initPrint, 374
 - insertAction, 374
 - insertActions, 375
 - loadingSnapshot, 375
 - loadSnapshot, 375, 376
 - menuActionInserted, 376
 - menuActionRemoved, 376
 - menuItem, 376
 - Model, 367
 - modifiedFiles, 377
 - postDraw, 377
 - postSelection, 378
 - preDraw, 378
 - print, 379
 - registerFile, 379
 - removeAction, 379
 - removeMenuItem, 380
 - reread, 381
 - restart, 381
 - restartModel, 381
 - run, 381
 - runModel, 381
 - saveNextSnapshot, 382
 - saveSnapshot, 382
 - savingNextSnapshot, 382
 - savingScreenshot, 382
 - savingSnapshot, 382
 - screenshot, 383
 - serialize, 383
 - setAnimationPeriod, 384
 - setExitCode, 384
 - setStatusMessage, 384
 - statusMessage, 384
 - step, 385
 - stop, 385
 - stopModel, 385
 - unregisterFile, 385
 - version, 385
 - versionNumber, 386
- model
 - util::WatchDog, 977
 - Viewer, 689
 - vvcomplex::VVComplex, 828
- modelRecordingStart
 - Viewer, 689
- modelRecordingStop
 - Viewer, 689
- modified
 - util::FileObject, 291
- modifiedFiles
 - bspline_tissue_model::TissueModel, 579
 - Model, 377
 - tissue_model::TissueModel, 567
- modulate
 - util::Texture1D, 512
 - util::Texture2D, 518
- mult
 - util::Vector, 632
- multiply
 - util::Vector, 649
- name
 - storage::TypeTraits, 614
- names
 - util::WatchDog, 978
- nb_cells
 - vvcomplex::VVComplexGraph, 850
- nb_junctions
 - vvcomplex::VVComplexGraph, 850
- nb_vertices1
 - graph::VVBIGraph, 763
- nb_vertices2
 - graph::VVBIGraph, 764
- nbColumns
 - util::Matrix, 348
- nbRows
 - util::Matrix, 348
- neighbor1_found_t
 - graph::VVBIGraph, 723
- neighbor1_t
 - graph::VVBIGraph, 723
- neighbor2_found_t

- graph::VVBIGraph, 724
- neighbor2_t
 - graph::VVBIGraph, 724
- neighbor_found_t
 - graph::VVGraph, 920
- neighbor_iterator
 - graph::VVGraph, 920
- neighbor_iterator1
 - graph::VVBIGraph, 724
- neighbor_iterator1_range
 - graph::VVBIGraph, 724
- neighbor_iterator2
 - graph::VVBIGraph, 725
- neighbor_iterator2_range
 - graph::VVBIGraph, 725
- neighbor_iterator_range
 - graph::VVGraph, 920
- neighbor_t
 - graph::VVGraph::neighbor_t, 388
- neighborhood
 - graph::VVBIGraph::search_result_t, 445
 - graph::VVGraph, 971
 - graph::VVGraph::search_result_t, 440
- neighborhood1
 - graph::VVBIGraph, 787
- neighborhood1_t
 - graph::VVBIGraph, 725
- neighborhood1_value_type
 - graph::VVBIGraph, 725
- neighborhood2
 - graph::VVBIGraph, 788
- neighborhood2_t
 - graph::VVBIGraph, 726
- neighborhood2_value_type
 - graph::VVBIGraph, 726
- neighborhood_t
 - graph::VVGraph, 921
- neighborhood_value_type
 - graph::VVGraph, 921
- neighbors
 - graph::VVBIGraph, 764–767
 - graph::VVGraph, 949–952
- NewtMaxSteps
 - solver::Solver, 501
- NewtTol
 - solver::Solver, 501
- NewtTolType
 - solver::Solver, 501
- newViewer
 - Viewer, 690
- nextTo
 - graph::VVBIGraph, 768
 - graph::VVGraph, 952
- NO__ERROR
 - storage, 139
- NO_FIELD_ERROR
 - storage, 139
- no_cell
 - vvcomplex::VVComplexGraph, 851
- no_junction
 - vvcomplex::VVComplexGraph, 851
- no_vertex1
 - graph::VVBIGraph, 769
- no_vertex2
 - graph::VVBIGraph, 769
- norm
 - util::Matrix, 360
 - util::Vector, 632, 650
- normal
 - algorithms::TriangleSurface, 609
 - algorithms::TriangleSurface::Cell, 205
 - algorithms::TriangleSurface::Vertex, 656
 - util::Contour, 253
- normalize
 - util::Vector, 632
- normalized
 - util::Vector, 633, 650, 651
- normalizeX
 - util::Function, 298
- normalizeY
 - util::Function, 298
- normsq
 - util::Matrix, 360
 - util::Vector, 633, 651
- notequal
 - util, 164
- null
 - graph::Vertex, 673
- num
 - graph::Vertex, 667
- numberOfLines
 - util::GraphInset, 319
- objreader
 - complex_factory, 91, 92
- OldC

- vvcomplex::VVComplex, 828
- OldS
 - vvcomplex::VVComplex, 829
- OldW
 - vvcomplex::VVComplex, 829
- OpenGL_ERROR_CHECK
 - glerrorcheck.h, 1074
- openGraph
 - util::GraphInset, 319
- openOpenGLConfig
 - Viewer, 690
- openScreenshotFormatDialog
 - Viewer, 690
- operator bool
 - complex_factory::ObjReaderError, 391
 - graph::Arc, 191
 - graph::Edge, 282
 - graph::Vertex, 667
 - graph::VVBiGraph::search_result_t, 444
 - graph::VVGraph::search_result_t, 440
 - vvcomplex::DivisionData, 272
- operator edge_t
 - graph::Arc, 191
- operator<
 - graph::Edge, 284
 - graph::Vertex, 669
 - util::Vector, 637
- operator<<
 - util, 165
- operator<=
 - graph::Edge, 284
 - graph::Vertex, 669
 - util::Vector, 638
- operator>
 - graph::Edge, 285
 - graph::Vertex, 670
 - util::Vector, 639
- operator>>
 - util, 165
- operator>=
 - graph::Edge, 285
 - graph::Vertex, 671
 - util::Vector, 639
- operator*
 - geometry::Quaternion, 425
 - graph::Arc, 191
 - graph::Edge, 283
 - graph::Vertex, 668
 - util, 164
 - util::Matrix, 349, 361
 - util::SelectMemberIterator, 452
 - util::Vector, 634, 652
- operator*=
 - geometry::Quaternion, 425, 426
 - util::Point, 417
 - util::Vector, 634
- operator~
 - util::Matrix, 352
- operator^
 - util::Vector, 653, 654
- operator()
 - solver::Solver, 477
 - storage::ConvertType, 263
 - util::Contour, 253
 - util::Function, 299
 - util::Matrix, 348, 349
 - util::Parms, 410–413
- operator+
 - geometry::Quaternion, 426
 - util::Matrix, 350
 - util::Vector, 635, 652
- operator++
 - util::SelectMemberIterator, 452, 453
- operator+=
 - geometry::Quaternion, 426
 - util::SelectMemberIterator, 453
 - util::Vector, 635
- operator-
 - graph::Arc, 192
 - util::Matrix, 350
 - util::Point, 417
 - util::SelectMemberIterator, 455
 - util::Vector, 635, 653
- operator->
 - graph::Arc, 192
 - graph::Edge, 283
 - graph::Vertex, 668
 - util::SelectMemberIterator, 454
- operator->*
 - graph::Edge, 283, 284
 - graph::Vertex, 668
- operator--
 - util::SelectMemberIterator, 453, 454
- operator-=
 - util::SelectMemberIterator, 454
 - util::Vector, 636
- operator/

- geometry::Quaternion, [426](#)
 - util::Matrix, [350](#)
 - util::Point, [417](#)
 - util::Vector, [636](#)
- operator/=
 - geometry::Quaternion, [427](#)
 - util::Point, [417](#)
 - util::Vector, [637](#)
- operator=
 - geometry::Quaternion, [427](#)
 - graph::Edge, [284](#)
 - graph::Vertex, [669](#)
 - graph::VVBIGraph, [770](#)
 - graph::VVGraph, [953](#)
 - graph::WeakVertex, [982](#)
 - util::Color, [246](#)
 - util::Contour, [254](#)
 - util::FileObject, [291](#)
 - util::Function, [300](#)
 - util::Matrix, [351](#)
 - util::range, [433](#), [434](#)
 - util::repair, [436](#)
 - util::SelectMemberIterator, [455](#)
 - util::Texture1D, [512](#)
 - util::Texture2D, [518](#)
 - util::Vector, [638](#)
- operator==
 - graph::Edge, [285](#)
 - graph::Vertex, [670](#)
 - graph::VVBIGraph, [771](#)
 - graph::VVBIGraph::single_-neighborhood_t, [463](#)
 - graph::VVGraph, [954](#)
 - graph::VVGraph::neighbor_t, [388](#)
 - graph::VVGraph::single_-neighborhood_t, [466](#)
 - util::Vector, [639](#)
- operator%
 - util::Vector, [651](#), [652](#)
- Options
 - storage, [138](#)
- ordered_cells_t
 - bspline_tissue_model::TissueModel, [575](#)
 - tissue_model::TissueModel, [564](#)
- orthogonal
 - util::Vector, [654](#)
- Palette
 - util::Palette, [393](#)
- palette
 - bspline_tissue_model::TissueModel, [586](#)
 - tissue::Tissue, [559](#)
 - tissue_model::TissueModel, [570](#)
- parallel, [124](#)
 - threadEval, [125](#)
- parallel::ThreadEval, [520](#)
- ParallelControler, [399](#)
- parallelepiped
 - shape, [127](#)
- Parms
 - util::Parms, [403](#)
- pdt
 - solver::Solver, [501](#)
- peff
 - solver::Solver, [501](#)
- pinching
 - cell_-system::CellSystemDivisionParams, [231](#)
- pinchingParam
 - cell_-system::CellSystemDivisionParams, [231](#)
- planeLineIntersection
 - geometry, [116](#)
- pmin
 - algorithms::TriangleSurface, [612](#)
- Point
 - util::Point, [416](#)
- Point2d
 - geometry, [113](#)
- Point3d
 - geometry, [114](#)
- point3d
 - algorithms::TriangleSurface, [609](#)
- Point4d
 - geometry, [114](#)
- pointer
 - graph::Edge, [280](#)
 - graph::Vertex, [661](#)
 - graph::WeakVertex, [980](#)
 - util::GraphInset, [311](#)
 - util::SelectMemberIterator, [450](#)
- pointInPolygon
 - geometry, [116](#)
- pointInTriangle
 - geometry, [117](#)
- polygonArea

- geometry, [117](#)
- pos
 - algorithms::TriangleSurface::Vertex, [656](#)
 - cell_system::CellSystemCell, [228](#)
 - cell_system::CellSystemJunction, [234](#)
- position
 - algorithms::TriangleSurface, [609](#), [610](#)
- postDraw
 - bspline_tissue_model::TissueModel, [580](#)
 - Model, [377](#)
 - tissue::Tissue, [545](#)
 - tissue_model::TissueModel, [568](#)
 - Viewer, [690](#)
- postSelection
 - Model, [378](#)
 - Viewer, [691](#)
- preDraw
 - bspline_tissue_model::TissueModel, [580](#)
 - Model, [378](#)
 - tissue::Tissue, [546](#)
 - tissue_model::TissueModel, [568](#)
 - Viewer, [691](#)
- pressure
 - cell_system::CellSystem, [224](#)
- prevTo
 - graph::VVBIGraph, [773](#)
 - graph::VVGraph, [956](#)
- print
 - Model, [379](#)
- PrintMatrix
 - solver::Solver, [502](#)
- PrintStats
 - solver::Solver, [502](#)
- proj
 - util::Point, [418](#)
- proj_length
 - util::Point, [418](#)
- projectPointOnLine
 - geometry, [118](#)
- projectPointOnPlane
 - geometry, [118](#)
- projectPointOnTriangle
 - geometry, [119](#)
- projectXY
 - util::Vector, [640](#)
- pu
 - vvcomplex::DivisionData, [273](#)
- pv
 - vvcomplex::DivisionData, [274](#)
- qdemangle
 - util, [166](#)
- Quaternion
 - geometry::Quaternion, [422](#), [423](#)
- r
 - util::Color, [247](#)
- ran
 - util, [166](#)
- random
 - util, [166](#), [167](#)
- range
 - util::range, [431](#), [432](#)
- range_c
 - util, [167](#)
- range_closed
 - util, [168](#)
- range_open
 - util, [168](#)
- range_vertex1
 - graph::VVBIGraph, [726](#)
- range_vertex2
 - graph::VVBIGraph, [726](#)
- ratio
 - cell_
 - system::CellSystemDivisionParams, [231](#)
 - cell_system::DivisionParams, [277](#)
- reader
 - storage::old::VVEStorage_
 - BINReader, [870](#)
 - storage::VVEStorage, [862](#)
 - storage::VVEStorage_BINReader, [879](#)
 - storage::VVEStorage_BINWriter, [888](#)
 - storage::VVEStorage_XMLReader, [895](#)
 - storage::VVEStorage_XMLWriter, [903](#)
- readMechanicParam
 - cell_system::CellSystem, [213](#)
- readOBJS
 - algorithms::TriangleGrowth, [593](#)
- readParam

- bspline_tissue_model::TissueModel, 581
- cell_system::CellSystem, 213
- tissue_model::TissueModel, 568
- readParms
 - solver::Solver, 478
 - tissue::Tissue, 546
 - util::GraphInset, 319
- readRules
 - cell_system::CellSystem, 214
- readSymbols
 - cell_system::CellSystem, 216
- readTissueParam
 - bspline_tissue_model::TissueModel, 581
 - tissue_model::TissueModel, 569
- readViewParms
 - tissue::Tissue, 548
- real_pointer
 - graph::Vertex, 661
- Recompute
 - util::Shapes::BSplineSurface, 201
- reconnectGraphs
 - vvcomplex::VVComplex, 823
- reference
 - graph::VVBIGraph, 774
 - graph::VVGraph, 957
 - storage::VVEStorage, 863
 - util::GraphInset, 311
 - util::SelectMemberIterator, 450
- REFERENCE_ERROR
 - storage, 139
- reference_t
 - storage::old::VVEStorage_ - BINReader, 868
 - storage::VVEStorage, 857
 - storage::VVEStorage_BINReader, 877
 - storage::VVEStorage_BINWriter, 886
 - storage::VVEStorage_XMLReader, 893
 - storage::VVEStorage_XMLWriter, 901
- refpair
 - util::refpair, 435, 436
- registerFile
 - Model, 379
- registerFiles
 - bspline_tissue_model::TissueModel, 582
- registerSymbol
 - cell_system::CellSystem, 217
- release
 - graph::Vertex, 671
- removeAction
 - Model, 379
- removeMenuItem
 - Model, 380
- removeObject
 - util::WatchDog, 976
- removeWhitespace
 - cell_system, 84
- replace
 - graph::VVBIGraph, 775
 - graph::VVGraph, 958
 - util::Texture1D, 513
 - util::Texture2D, 519
- Reread
 - util::Shapes::BSplineSurface, 201
- reread
 - Model, 381
 - util::Contour, 254
 - util::FileObject, 292
 - util::Function, 301
 - util::KeyFramer, 331
 - util::Materials, 336
 - util::Palette, 396
- restart
 - Model, 381
- restartModel
 - Model, 381
- restLength
 - cell_system::CellSystem, 224
- reverse_iterator
 - util::GraphInset, 311
- rex
 - tissue_model::TissueModel, 571
- right_label
 - cell_system::DivisionParams, 277
- rotate
 - geometry::Quaternion, 427
- rotation
 - util::Matrix, 352, 353
- run
 - Model, 381
- RungeKutta
 - solver, 132
- RungeKuttaDt

- solver::Solver, [502](#)
- runModel
 - Model, [381](#)
- S
 - algorithms::TriangleSurface, [613](#)
 - vvcomplex::VVComplex, [830](#)
- sampleDx
 - tissue::Tissue, [559](#)
- save_parameters
 - vvcomplex::VVComplex, [831](#)
- saveNextSnapshot
 - Model, [382](#)
- saveScreenshot
 - Viewer, [691](#)
- saveSnapshot
 - Model, [382](#)
- saveSubgraphs
 - vvcomplex::VVComplex, [824](#)
- savingNextSnapshot
 - Model, [382](#)
- savingScreenshot
 - Model, [382](#)
- savingSnapshot
 - Model, [382](#)
- scale_s
 - util::Tensor, [506](#)
- scale_t
 - util::Tensor, [507](#)
- scaleX
 - util::Function, [304](#)
- scaleY
 - util::Function, [304](#), [305](#)
- screenshot
 - Model, [383](#)
- screenshotCounter
 - Viewer, [692](#)
- screenshotFileName
 - Viewer, [692](#)
- screenshotFormat
 - Viewer, [692](#)
- screenshotQuality
 - Viewer, [693](#)
- search_result_t
 - graph::VVBiGraph::search_result_t, [443](#)
 - graph::VVGraph::search_result_t, [439](#)
- second
 - util::refpair, [436](#)
- segmentSegmentIntersection
 - geometry, [119](#)
- select
 - util::Palette, [396](#)
- selectColor
 - util::Palette, [397](#)
- SelectMemberIterator
 - util::SelectMemberIterator, [451](#)
- serialization
 - storage, [140](#)
- serialize
 - cell_system::CellSystem, [217](#)
 - cell_system::CellSystemCell, [227](#)
 - cell_system::CellSystemJunction, [233](#)
 - graph::Edge, [286](#)
 - graph::Vertex, [672](#)
 - Model, [383](#)
 - storage::old::VVEStorage_ - BINReader, [870](#)
 - storage::VVEStorage, [864](#)
 - storage::VVEStorage_BINReader, [879](#)
 - storage::VVEStorage_BINWriter, [888](#)
 - storage::VVEStorage_XMLReader, [895](#)
 - storage::VVEStorage_XMLWriter, [903](#)
 - vvcomplex::VVComplex, [825](#)
- set
 - util::Tensor, [507](#)
 - util::Vector, [640](#), [641](#)
- setAnimationPeriod
 - Model, [384](#)
- setAxisAngle
 - geometry::Quaternion, [428](#)
- setBegin
 - util::range, [434](#)
- setCellPinchingParams
 - tissue::Tissue, [549](#)
- setEnd
 - util::range, [434](#)
- setExitCode
 - Model, [384](#)
- setFilename
 - util::FileObject, [292](#)
- setIndex
 - Viewer, [693](#)
- setLastError

- storage::old::VVEStorage_ -
 - BINReader, [871](#)
- storage::VVEStorage, [864](#)
- storage::VVEStorage_BINReader, [881](#)
- storage::VVEStorage_XMLReader, [896](#)
- setMatrix
 - geometry::Quaternion, [428](#)
- setModel
 - Viewer, [693](#)
- setNumberOfLines
 - util::GraphInset, [320](#)
- setOption
 - storage::old::VVEStorage_ -
 - BINReader, [871](#)
 - storage::VVEStorage, [864](#)
 - storage::VVEStorage_BINReader, [881](#)
 - storage::VVEStorage_XMLReader, [897](#)
 - storage::VVEStorage_XMLWriter, [904](#)
- SetPos
 - bspline_tissue_model::TissueModel, [582](#)
- setSamples
 - util::Function, [305](#)
- setScreenshotCounter
 - Viewer, [693](#)
- setScreenshotFileName
 - Viewer, [694](#)
- setScreenshotFormat
 - Viewer, [694](#)
- setScreenshotQuality
 - Viewer, [694](#)
- setStatusMessage
 - Model, [384](#)
 - Viewer, [695](#)
- setTime
 - algorithms::TriangleGrowth, [600](#)
 - util::GraphInset, [321](#)
- shape, [126](#)
 - cube, [126](#)
 - cylinder, [126](#)
 - disk, [127](#)
 - parallelepiped, [127](#)
 - sphere, [127](#)
- shiftX
 - util::Function, [305](#)
- shiftY
 - util::Function, [306](#)
- shortest_paths_FloydWarshall
 - algorithms, [77](#)
- ShortWallAlgoParams
 - tissue::ShortWallAlgoParams, [460](#)
- showAll
 - util::GraphInset, [324](#)
- showEvent
 - Viewer, [695](#)
- showIndices
 - util::GraphInset, [325](#)
- showInterval
 - cell_system::CellSystem, [225](#)
- showTime
 - cell_system::CellSystem, [225](#)
- single_neighborhood1_t
 - graph::VVBIGraph, [727](#)
- single_neighborhood2_t
 - graph::VVBIGraph, [727](#)
- size
 - algorithms::TriangleGrowth, [601](#)
 - graph::VVBIGraph, [776](#)
 - graph::VVGGraph, [959](#)
 - util::GraphInset, [321](#)
 - util::Matrix, [353](#)
 - util::range, [434](#)
 - util::Vector, [642](#)
- size_type
 - graph::VVBIGraph, [727](#)
 - graph::VVGGraph, [921](#)
 - vvcomplex::VVComplexGraph, [838](#)
- smoothNormal
 - algorithms::TriangleSurface, [611](#)
- solveAdaptiveCrankNicholson
 - solver::Solver, [480](#)
- solveAdaptiveEuler
 - solver::Solver, [485](#)
- solveAdaptiveRungeKutta
 - solver::Solver, [486](#)
- solveEuler
 - solver::Solver, [489](#)
- solveFixedpoint
 - solver::Solver, [490](#)
- solveMidpoint
 - solver::Solver, [491](#)
- Solver
 - solver::Solver, [474](#)
- solver, [129](#)
 - AdaptiveCrankNicholson, [132](#)

- AdaptiveEuler, 132
- AdaptiveRungeKutta, 132
- Euler, 132
- Fixedpoint, 132
- MAX_COMPONENT, 132
- MEAN_COMPONENT, 132
- Midpoint, 132
- RungeKutta, 132
- SolvingMethod, 131
- ToleranceType, 132
- solver::Solver, 468
 - AEulerHighTol, 494
 - AEulerIncDt, 494
 - AEulerLowTol, 494
 - AEulerResDt, 494
 - AEulerResTol, 494
 - AEulerTolType, 495
 - ARungeHighTol, 495
 - ARungeIncDt, 495
 - ARungeLowTol, 495
 - ARungeResDt, 495
 - ARungeResTol, 496
 - ARungeTolType, 496
 - ConjGradMaxSteps, 496
 - ConjGradTol, 496
 - ConjGradTolType, 496
 - ConstNbPartials, 497
 - CRAvgCPU, 497
 - CRIncDt, 497
 - CRMinCPU, 497
 - CRResDt, 497
 - current_method, 498
 - dt, 498
 - Dx, 498
 - EdgeInternals, 473
 - EulerDt, 498
 - FindPartials, 475
 - FixedPointDt, 498
 - FixedPointMaxSteps, 499
 - FixedPointTol, 499
 - FixedPointTolType, 499
 - InitialDt, 499
 - initialize, 476
 - initialized, 499
 - Mat, 473
 - MaxDt, 500
 - method, 500
 - MidPointDt, 500
 - MinDt, 500
 - NewtonMaxSteps, 501
 - NewtonTol, 501
 - NewtonTolType, 501
 - operator(), 477
 - pdt, 501
 - peff, 501
 - PrintMatrix, 502
 - PrintStats, 502
 - readParms, 478
 - RungeKuttaDt, 502
 - solveAdaptiveCrankNicholson, 480
 - solveAdaptiveEuler, 485
 - solveAdaptiveRungeKutta, 486
 - solveEuler, 489
 - solveFixedpoint, 490
 - solveMidpoint, 491
 - Solver, 474
 - solveRungeKutta, 492
 - step, 502
 - tag_t, 473
 - Vec, 473
 - VertexInternals, 474
- solveRungeKutta
 - solver::Solver, 492
- SolvingMethod
 - solver, 131
- source
 - graph::Arc, 192
 - graph::Edge, 286
 - graph::VVBiGraph, 777
 - graph::VVGraph, 960
- sphere
 - shape, 127
- spliceAfter
 - graph::VVBiGraph, 777, 778
 - graph::VVGraph, 960, 961
- spliceBefore
 - graph::VVBiGraph, 778, 779
 - graph::VVGraph, 962, 963
- split
 - algorithms, 77
- splitWall
 - vvcomplex::VVComplex, 826
 - vvcomplex::VVComplexGraph, 851, 852
- splitWallExtra
 - vvcomplex::VVComplex, 827
- square_grid
 - complex_factory, 98
 - factory, 105
- squareFiller

- complex_factory, 101
- sran
 - util, 168
- sran_time
 - util, 168
- stability
 - cell_system::CellSystem, 225
- startCompound
 - storage::old::VVEStorage_
BINReader, 872
 - storage::VVEStorage, 865
 - storage::VVEStorage_BINReader,
882
 - storage::VVEStorage_BINWriter,
890
 - storage::VVEStorage_XMLReader,
897
 - storage::VVEStorage_XMLWriter,
905
- StartModel
 - bspline_tissue_model.h, 1019
- startRecordingCameraAnimation
 - Viewer, 695
- startRecordingModelAnimation
 - Viewer, 696
- startReference
 - storage::old::VVEStorage_
BINReader, 873
 - storage::VVEStorage, 865
 - storage::VVEStorage_BINReader,
882
 - storage::VVEStorage_BINWriter,
890
 - storage::VVEStorage_XMLReader,
898
 - storage::VVEStorage_XMLWriter,
905
- startTime
 - algorithms::TriangleGrowth, 601
- STATIC_ASSERT
 - static_assert.h, 1086
- static_assert.h
 - STATIC_ASSERT, 1086
- statusMessage
 - Model, 384
 - Viewer, 696
- std, 133
 - swap, 133
- step
 - bspline_tissue_model::TissueModel,
583
 - cell_system::CellSystem, 218
 - Model, 385
 - solver::Solver, 502
 - tissue_model::TissueModel, 569
- step_cellsystem
 - cell_system::CellSystem, 218
- step_cellsystem_division
 - cell_system::CellSystem, 218
- step_cellsystem_meca
 - cell_system::CellSystem, 219
- stepDrawFinished
 - Viewer, 696
- stop
 - Model, 385
- stopModel
 - Model, 385
- stopRecordingCameraAnimation
 - Viewer, 696
- stopRecordingModelAnimation
 - Viewer, 696
- storage, 134
 - BAD_CONTENT, 139
 - convert_type, 140
 - find_type, 140
 - IO_ERROR, 139
 - isStrictConversion, 140
 - NO__ERROR, 139
 - NO_FIELD_ERROR, 139
 - Options, 138
 - REFERENCE_ERROR, 139
 - serialization, 140
 - STORAGE_ERROR, 138
 - TCO_Exact, 139
 - TCO_NoCheck, 140
 - TCO_Permissive, 140
 - TCO_PermissiveNoWarning, 140
 - TCO_Strict, 139
 - TYPE_CHECK_ERROR, 139
 - TYPE_CONVERSION_ERROR,
139
 - TypeChecking, 138
 - TypeCheckingOption, 139
 - typeid_to_name, 141
 - typename_to_id, 141
 - UNKNOWN_ERROR, 139
 - USER_ERROR, 139
 - versionNumber, 141
- storage::ConvertType, 263

- operator(), 263
- storage::Frame, 294
- storage::old::VVEStorage_BINReader, 867
 - checkNextField, 869
 - endCompound, 869
 - endReference, 869
 - filename, 869
 - reader, 870
 - reference_t, 868
 - serialize, 870
 - setLastError, 871
 - setOption, 871
 - startCompound, 872
 - startReference, 873
 - writer, 874
- storage::TypeTraits, 614
 - id, 614
 - name, 614
 - type, 614
- storage::VVEStorage, 853
 - ~VVEStorage, 857
 - checkNextField, 857
 - clear, 857
 - clearLastError, 857
 - endCompound, 858
 - endReference, 858
 - field, 858–861
 - filename, 861
 - fileVersion, 862
 - ignore, 862
 - lastError, 862
 - lastErrorString, 862
 - reader, 862
 - reference, 863
 - reference_t, 857
 - serialize, 864
 - setLastError, 864
 - setOption, 864
 - startCompound, 865
 - startReference, 865
 - version, 865
 - writer, 866
- storage::VVEStorage_BINReader, 875
 - checkNextField, 877
 - endCompound, 877
 - endReference, 878
 - filename, 878
 - frame_stack, 884
 - ignore, 878
 - reader, 879
 - reference_t, 877
 - serialize, 879
 - setLastError, 881
 - setOption, 881
 - startCompound, 882
 - startReference, 882
 - top_level_compounds, 884
 - writer, 883
- storage::VVEStorage_BINWriter, 885
 - checkNextField, 887
 - endCompound, 887
 - endReference, 887
 - filename, 888
 - frame_stack, 891
 - reader, 888
 - reference_t, 886
 - serialize, 888
 - startCompound, 890
 - startReference, 890
 - writer, 891
- storage::VVEStorage_XMLReader, 892
 - checkNextField, 894
 - endCompound, 894
 - endReference, 894
 - filename, 895
 - reader, 895
 - reference_t, 893
 - serialize, 895
 - setLastError, 896
 - setOption, 897
 - startCompound, 897
 - startReference, 898
 - writer, 899
- storage::VVEStorage_XMLWriter, 900
 - checkNextField, 902
 - endCompound, 902
 - endReference, 902
 - filename, 903
 - reader, 903
 - reference_t, 901
 - serialize, 903
 - setOption, 904
 - startCompound, 905
 - startReference, 905
 - writer, 906
- STORAGE_ERROR
 - storage, 138
- store_vertices
 - graph::VVGraph, 964

- strictCellWallMin
 - tissue::ClosestMidAlgoParams, [239](#)
 - tissue::ClosestWallAlgoParams, [241](#)
 - tissue::ShortWallAlgoParams, [461](#)
 - tissue::Tissue, [560](#)
- string
 - complex_factory::ObjReaderError, [391](#)
- strong_ref
 - graph::WeakVertex, [981](#)
- subgraph
 - graph::VVGraph, [965](#)
- surface
 - algorithms::TriangleGrowth, [601](#), [602](#)
- swap
 - graph::VVBIGraph, [780](#)
 - graph::VVGraph, [967](#)
 - std, [133](#)
- sync
 - graph::Arc, [192](#)
- T
 - bspline_tissue_model::TissueModel, [586](#)
 - cell_system::CellSystem, [225](#)
 - tissue_model::TissueModel, [571](#)
- t
 - util::Vector, [642](#)
- tag_t
 - solver::Solver, [473](#)
- tangent
 - util::Contour, [261](#)
- target
 - graph::Arc, [193](#)
 - graph::Edge, [286](#)
 - graph::VVBIGraph, [780](#), [781](#)
 - graph::VVGraph, [967](#)
 - graph::VVGraph::neighbor_t, [389](#)
- TCO_Exact
 - storage, [139](#)
- TCO_NoCheck
 - storage, [140](#)
- TCO_Permissive
 - storage, [140](#)
- TCO_PermissiveNoWarning
 - storage, [140](#)
- TCO_Strict
 - storage, [139](#)
- template_utils, [142](#)
- Tensor
 - util::Tensor, [505](#)
- test, [143](#)
- test.h
 - CHECK_EQUAL, [1054](#)
 - CHECK_FLOAT, [1054](#)
 - CHECK_FLOAT_DIFF, [1054](#)
 - CHECK_NOTEQUAL, [1055](#)
 - CHECK_ZERO, [1055](#)
 - TEST_FCT, [1055](#)
- TEST_FCT
 - test.h, [1055](#)
- testDivisionOnVertices
 - vvcomplex, [185](#)
- Texture1D
 - util::Texture1D, [509](#)
- Texture2D
 - util::Texture2D, [515](#), [516](#)
- TextureId
 - util::Shapes::BSplineSurface, [201](#)
- threadEval
 - parallel, [125](#)
- tie
 - util, [169](#)
- time
 - algorithms::TriangleGrowth, [602](#)
 - algorithms::TriangleSurface, [613](#)
 - bspline_tissue_model::TissueModel, [586](#)
 - cell_system::CellSystem, [225](#)
 - util::GraphInset, [325](#)
 - util::GraphInset::Data, [267](#)
- timePos
 - algorithms::TriangleGrowth, [602](#)
- timeSpan
 - util::GraphInset, [325](#)
- Tissue
 - tissue::Tissue, [526](#), [527](#)
- tissue, [144](#)
 - CELL_DIVISION_ALGORITHM, [146](#)
 - cell_graph, [145](#)
 - cellPinching, [146](#)
 - findDivisionPoints, [147](#), [149](#), [150](#)
 - tissue_graph, [145](#)
 - vertex, [145](#)
- tissue::CellPinchingParams, [206](#)
 - cellMaxPinch, [207](#)
 - cellPinch, [207](#)
 - CellPinchingParams, [207](#)

- tissue::ClosestMidAlgoParams, 238
 - cellWallMin, 239
 - ClosestMidAlgoParams, 239
 - strictCellWallMin, 239
- tissue::ClosestWallAlgoParams, 240
 - cellWallMin, 241
 - ClosestWallAlgoParams, 241
 - strictCellWallMin, 241
- tissue::ShortWallAlgoParams, 459
 - cellWallMin, 460
 - cellWallSample, 460
 - dx, 461
 - ShortWallAlgoParams, 460
 - strictCellWallMin, 461
- tissue::Tissue, 521
 - blending, 552
 - calcQuads, 528
 - cellDivAlg, 552
 - cellMaxPinch, 553
 - cellPinch, 553
 - cellWallCorner, 553
 - cellWallMin, 554
 - cellWallSample, 554
 - cellWallWidth, 555
 - center_blending, 555
 - colorBegin, 556
 - colorCenter, 556
 - colorEnd, 557
 - contourColor, 557
 - divideCell, 529–533
 - division_data, 525
 - drawBorders, 558
 - drawCell, 534, 535
 - drawCellContour, 537
 - drawInsides, 558
 - drawSimplifiedCell, 538
 - drawWalledCell, 539, 540
 - getCellPinchingParams, 543
 - getClosestMidAlgoParams, 543
 - getCloseWallAlgoParams, 544
 - getShortWallAlgoParams, 545
 - palette, 559
 - postDraw, 545
 - preDraw, 546
 - readParms, 546
 - readViewParms, 548
 - sampleDx, 559
 - setCellPinchingParams, 549
 - strictCellWallMin, 560
 - Tissue, 526, 527
 - valueCenterColor, 549
 - valueColor, 550
- tissue_graph
 - tissue, 145
- tissue_model, 153
 - epsilon, 153
- tissue_model::TissueModel, 561
 - backColor, 570
 - cellFromId, 564
 - draw, 565
 - drawNeighborhood, 570
 - drawWithNames, 566
 - getCellCenterColor, 566
 - getCellColor, 566
 - initDraw, 567
 - initialize, 567
 - modifiedFiles, 567
 - ordered_cells_t, 564
 - palette, 570
 - postDraw, 568
 - preDraw, 568
 - readParam, 568
 - readTissueParam, 569
 - rex, 571
 - step, 569
 - T, 571
 - TissueModel, 564
 - updateCellsArea, 569
- tissue_model::TissueModel::CompareSize, 249
- TissueModel
 - bspline_tissue_model::TissueModel, 575
 - tissue_model::TissueModel, 564
- ToleranceType
 - solver, 132
- top_level_compounds
 - storage::VVEStorage_BINReader, 884
- trace
 - util::Matrix, 353
- Transform
 - util::Shapes::BSplineSurface, 201
- transpose
 - util::Matrix, 361
- travel
 - util::Contour, 261
- triangleArea
 - geometry, 120
- Type

- complex_factory::ObjReaderError, 391
- type
 - complex_factory::ObjReaderError, 391
 - storage::TypeTraits, 614
- TYPE_CHECK_ERROR
 - storage, 139
- TYPE_CONVERSION_ERROR
 - storage, 139
- TypeChecking
 - storage, 138
- TypeCheckingOption
 - storage, 139
- typeid_to_name
 - storage, 141
- typename_to_id
 - storage, 141
- types.h
 - FOR_ALL_TYPEIDS, 1045
 - FOR_ALL_TYPES, 1046
 - FOR_ALL_TYPES_NOSTRING, 1046
- u1
 - vvcomplex::DivisionData, 274
- undoInEdge
 - graph::VVBIGraph, 781
 - graph::VVGraph, 968
- UNKNOWN_ERROR
 - storage, 139
- unregisterFile
 - Model, 385
- updateAll
 - Viewer, 697
- updateCellsArea
 - bspline_tissue_model::TissueModel, 583
 - cell_system::CellSystem, 220
 - tissue_model::TissueModel, 569
- updateEdgeCache
 - graph::VVBIGraph, 782
 - graph::VVGraph, 968
- updatePositions
 - bspline_tissue_model::TissueModel, 583
- updateVertexCache
 - graph::VVBIGraph, 783
 - graph::VVGraph, 969
- useColor
 - util::Palette, 398
- useMaterial
 - util::Materials, 337
- USER_ERROR
 - storage, 139
- util, 154
 - absoluteDir, 159
 - approx, 160
 - clamp, 160
 - convertHSVtoRGB, 160
 - demangle, 161
 - gaussRan, 161, 162
 - greater, 162
 - less, 162
 - make_range, 162, 163
 - maximum, 163
 - minimum, 163
 - notequal, 164
 - operator<<, 165
 - operator>>, 165
 - operator*, 164
 - qdemangle, 166
 - ran, 166
 - random, 166, 167
 - range_c, 167
 - range_closed, 168
 - range_open, 168
 - srans, 168
 - srans_time, 168
 - tie, 169
- util::Buffer, 203
 - ~Buffer, 204
 - Buffer, 203
 - c_data, 204
- util::CircIterator, 235
 - CircIterator, 236
 - end, 237
 - isInitIterator, 237
- util::Color, 242
 - a, 244, 245
 - b, 245
 - Color, 244
 - g, 245, 246
 - operator=, 246
 - r, 247
- util::Contour, 250
 - Contour, 251
 - getMax, 252
 - getMin, 252
 - length, 252

- normal, 253
- operator(), 253
- operator=, 254
- reread, 254
- tangent, 261
- travel, 261
- util::CrossProductType< 2, T >, 265
- util::CrossProductType< 3, T >, 266
- util::FileObject, 289
 - FileObject, 290, 291
 - getFilename, 291
 - modified, 291
 - operator=, 291
 - reread, 292
 - setFilename, 292
- util::Function, 295
 - Function, 297
 - getMax, 298
 - getMin, 298
 - normalizeX, 298
 - normalizeY, 298
 - operator(), 299
 - operator=, 300
 - reread, 301
 - scaleX, 304
 - scaleY, 304, 305
 - setSamples, 305
 - shiftX, 305
 - shiftY, 306
- util::GraphInset, 307
 - _numberOfLines, 322
 - addData, 312–314
 - autoOpenClose, 322
 - backColor, 322
 - backgroundOffset, 322
 - changeSourceColor, 322
 - changeSourceOffset, 323
 - changeSourceWidth, 323
 - check, 315
 - cleanData, 316
 - clear, 316
 - closed, 323
 - closeGraph, 316
 - contourColor, 323
 - contourOffset, 323
 - contourWidth, 323
 - data, 323
 - draw, 317
 - drawFrame, 318
 - drawWithName, 318
 - graphClosed, 319
 - GraphInset, 312
 - initialized, 324
 - iterator, 311
 - keepAllData, 324
 - linesColor, 324
 - linesMinMaxView, 324
 - linesOffset, 324
 - linesWidth, 324
 - numberOfLines, 319
 - openGraph, 319
 - pointer, 311
 - readParms, 319
 - reference, 311
 - reverse_iterator, 311
 - setNumberOfLines, 320
 - setTime, 321
 - showAll, 324
 - showIndices, 325
 - size, 321
 - time, 325
 - timeSpan, 325
 - value_type, 311
 - windowPosition, 325
 - windowSize, 325
 - writeCSV, 321
- util::GraphInset::Data, 267
 - changed, 267
 - time, 267
 - values, 267
- util::KeyFramer, 330
 - reread, 331
- util::Materials, 333
 - blend, 334
 - getMaterial, 335
 - Materials, 334
 - reread, 336
 - useMaterial, 337
- util::Materials::Material, 332
- util::Matrix, 339
 - c_data, 347
 - cofactor, 354
 - data, 348
 - det, 354
 - identity, 348
 - inverse, 355
 - map, 355–360
 - Matrix, 345–347
 - nbColumns, 348
 - nbRows, 348

- norm, 360
- normsq, 360
- operator*, 349, 361
- operator~, 352
- operator(), 348, 349
- operator+, 350
- operator-, 350
- operator/, 350
- operator=, 351
- rotation, 352, 353
- size, 353
- trace, 353
- transpose, 361
- zero, 354
- util::Palette, 392
 - blend, 394
 - getColor, 395
 - Palette, 393
 - reread, 396
 - select, 396
 - selectColor, 397
 - useColor, 398
- util::Parms, 400
 - all, 404–409
 - isLoading, 410
 - operator(), 410–413
 - Parms, 403
 - verboseLevel, 414
- util::Point, 415
 - ~Point, 416
 - c_data, 417
 - operator*=, 417
 - operator-, 417
 - operator/, 417
 - operator/=: 417
 - Point, 416
 - proj, 418
 - proj_length, 418
- util::range, 430
 - begin, 432
 - cbegin, 433
 - cend, 433
 - end, 433
 - operator=, 433, 434
 - range, 431, 432
 - setBegin, 434
 - setEnd, 434
 - size, 434
- util::refpair, 435
 - first, 436
 - operator=, 436
 - refpair, 435, 436
 - second, 436
- util::SelectMemberIterator, 447
 - base, 452
 - base_iterator, 450
 - difference_type, 450
 - it, 455
 - iterator_category, 450
 - operator*, 452
 - operator++, 452, 453
 - operator+=, 453
 - operator-, 455
 - operator->, 454
 - operator--, 453, 454
 - operator=, 454
 - operator=, 455
 - pointer, 450
 - reference, 450
 - SelectMemberIterator, 451
 - value_type, 450
- util::set_vector, 457
- util::Shapes, 170
- util::Shapes::BSplineSurface, 194
 - BoundingBox, 196
 - Draw, 196
 - IsTextured, 200
 - Load, 200
 - Recompute, 201
 - Reread, 201
 - TextureId, 201
 - Transform, 201
- util::Tensor, 505
 - ~Tensor, 506
 - angle_theta, 506
 - grow, 506
 - scale_s, 506
 - scale_t, 507
 - set, 507
 - Tensor, 505
- util::TextureID, 508
 - ~TextureID, 510
 - bind, 511
 - blend, 511
 - clamp, 511
 - decals, 511
 - filter, 511
 - modulate, 512
 - operator=, 512
 - replace, 513

- Texture1D, 509
- util::Texture2D, 514
- ~Texture2D, 516
- bind, 517
- blend, 517
- clamp, 517
- decal, 518
- filter, 518
- modulate, 518
- operator=, 518
- replace, 519
- Texture2D, 515, 516
- util::Vector, 616
- angle, 645, 646
- begin, 627, 628
- c_data, 628
- cross, 628
- data, 628
- divide, 646
- end, 629
- i, 629, 630
- j, 630
- k, 631
- l, 631, 632
- map, 646–648
- max, 649
- min, 649
- mult, 632
- multiply, 649
- norm, 632, 650
- normalize, 632
- normalized, 633, 650, 651
- normsq, 633, 651
- operator<, 637
- operator<=, 638
- operator>, 639
- operator>=, 639
- operator*, 634, 652
- operator*=, 634
- operator^, 653, 654
- operator+, 635, 652
- operator+=, 635
- operator-, 635, 653
- operator=, 636
- operator/, 636
- operator/=, 637
- operator=, 638
- operator==, 639
- operator%, 651, 652
- orthogonal, 654
- projectXY, 640
- set, 640, 641
- size, 642
- t, 642
- Vector, 625–627
- x, 642, 643
- y, 643, 644
- z, 644
- util::WatchDog, 973
- addObject, 975
- changeFilename, 975
- fileObjects, 977
- guarded, 976
- model, 977
- names, 978
- removeObject, 976
- watch, 977
- WatchDog, 974
- vl
- vvcomplex::DivisionData, 274
- valence
- graph::VVBGraph, 783, 784
- graph::VVGraph, 969
- valid
- algorithms::TriangleGrowth, 603
- value_type
- graph::VVGraph, 921
- util::GraphInset, 311
- util::SelectMemberIterator, 450
- valueCenterColor
- tissue::Tissue, 549
- valueColor
- tissue::Tissue, 550
- values
- util::GraphInset::Data, 267
- Vec
- solver::Solver, 473
- Vector
- util::Vector, 625–627
- vector
- algorithms::TriangleSurface, 611, 612
- vector3d
- algorithms::TriangleSurface, 612
- velocity
- cell_system::CellSystemJunction, 234
- verboseLevel
- util::Parms, 414

- version
 - cell_system::CellSystem, 221
 - Model, 385
 - storage::VVEStorage, 865
- versionNumber
 - cell_system::CellSystem, 221
 - Model, 386
 - storage, 141
- Vertex
 - graph::Vertex, 662–664
- vertex
 - tissue, 145
- vertex1_t
 - graph::VVBIGraph, 728
- vertex2_t
 - graph::VVBIGraph, 728
- vertex_counter
 - graph, 123
- vertex_identity_t
 - graph, 122
- vertex_t
 - graph::VVGraph, 921
- VertexInternals
 - solver::Solver, 474
- vertices
 - algorithms::TriangleSurface, 613
- vertices1
 - graph::VVBIGraph, 784, 785
- vertices2
 - graph::VVBIGraph, 785
- Viewer, 676
 - ~Viewer, 681
 - addToolBar, 681
 - addViewer, 682
 - cameraRecordingStart, 682
 - cameraRecordingStop, 682
 - configOpenGL, 682
 - createViewer, 683
 - domElement, 683
 - draw, 684
 - drawLight, 684
 - drawWithNames, 685
 - helpMenu, 685
 - helpString, 685
 - index, 686
 - init, 686
 - initFromDOMEElement, 686
 - keyPressEvent, 688
 - mainWindow, 688
 - menuBar, 689
 - model, 689
 - modelRecordingStart, 689
 - modelRecordingStop, 689
 - newViewer, 690
 - openOpenGLConfig, 690
 - openScreenshotFormatDialog, 690
 - postDraw, 690
 - postSelection, 691
 - preDraw, 691
 - saveScreenshot, 691
 - screenshotCounter, 692
 - screenshotFileName, 692
 - screenshotFormat, 692
 - screenshotQuality, 693
 - setIndex, 693
 - setModel, 693
 - setScreenshotCounter, 693
 - setScreenshotFileName, 694
 - setScreenshotFormat, 694
 - setScreenshotQuality, 694
 - setStatusMessage, 695
 - showEvent, 695
 - startRecordingCameraAnimation, 695
 - startRecordingModelAnimation, 696
 - statusMessage, 696
 - stepDrawFinished, 696
 - stopRecordingCameraAnimation, 696
 - stopRecordingModelAnimation, 696
 - updateAll, 697
 - Viewer, 680
 - wasInitialized, 697
- viewRulesApplication
 - cell_system::CellSystem, 226
- viewRulesDefinitions
 - cell_system::CellSystem, 226
- volumeLeftCell
 - cell_system, 84
- vvassert
 - assert.h, 1060
- vvassert_msg
 - assert.h, 1061
- VVBIGraph
 - graph::VVBIGraph, 728
- VVComplex
 - vvcomplex::VVComplex, 801
- vvcomplex, 171
 - cell, 180
 - cell_edge, 180

- cell_graph, 180
- cell_junction_edge, 180
- complex_graph, 180
- const_cell_edge, 180
- const_cell_junction_edge, 181
- const_junction_cell_edge, 181
- const_wall, 181
- FindCenter, 182
- findCenter, 182
- findDivisionPoints, 183
- FindOppositeWall, 183
- FindWallMin, 185
- junction, 181
- junction_cell_edge, 181
- testDivisionOnVertices, 185
- wall, 181
- wall_graph, 181
- vvcomplex::DivisionData, 270
 - divide_at_u1, 273
 - divide_at_v1, 273
 - DivisionData, 271, 272
 - junction, 271
 - operator bool, 272
 - pu, 273
 - pv, 274
 - u1, 274
 - v1, 274
- vvcomplex::InModelDivisionParam, 328
- vvcomplex::VVComplex, 790
 - ~VVComplex, 802
 - addCell, 803–807
 - addCellExtra, 808
 - addCellUnsorted, 808
 - adjacentCell, 809
 - adjacentCells, 810
 - border, 810–812
 - C, 827
 - cell, 796
 - cell_arc, 796
 - cell_edge, 796
 - cell_graph, 796
 - cell_junction_edge, 797
 - clear, 812
 - clearOldGraphs, 813
 - complex_graph, 797
 - connectFromJunctions, 813
 - const_cell_edge, 797
 - const_cell_junction_edge, 798
 - const_junction_cell_edge, 798
 - const_wall, 798
 - deleteCell, 815
 - deleteCellInGraphs, 815
 - deleteJunction, 816
 - deleteJunctionExtra, 816
 - divideCell, 817–820
 - division_data, 799
 - interfaceWall, 822
 - interfaceWallSpan, 822
 - junction, 799
 - junction_cell_edge, 800
 - mergeCells, 822
 - model, 828
 - OldC, 828
 - OldS, 829
 - OldW, 829
 - reconnectGraphs, 823
 - S, 830
 - save_parameters, 831
 - saveSubgraphs, 824
 - serialize, 825
 - splitWall, 826
 - splitWallExtra, 827
 - VVComplex, 801
 - W, 831
 - wall, 800
 - wall_arc, 800
 - wall_graph, 800
- vvcomplex::VVComplexGraph, 833
 - addCell, 838–841
 - adjacentCell, 841
 - adjacentCells, 842
 - any_cell, 843
 - any_junction, 843
 - border, 843, 844
 - const_iterator1, 837
 - const_iterator2, 837
 - deleteCell, 845
 - divideCell, 845
 - division_data, 837
 - get_cell, 847
 - get_junction, 848
 - interfaceWall, 848
 - interfaceWallSpan, 848
 - iterator1, 837
 - iterator2, 838
 - mergeCells, 849
 - nb_cells, 850
 - nb_junctions, 850
 - no_cell, 851
 - no_junction, 851

- size_type, 838
- splitWall, 851, 852
- vvcomplex::VVComplexGraph::division_ - result_t, 269
- vvelib/ Directory Reference, 71
- vvelib/algorithms/ Directory Reference, 57
- vvelib/algorithms/cellsystem.h, 989
- vvelib/algorithms/complex.h, 991
- vvelib/algorithms/draw_graphs.h, 1001
- vvelib/algorithms/graph.h, 1002
- vvelib/algorithms/insert.h, 1005
- vvelib/algorithms/parallel.h, 1006
- vvelib/algorithms/solver.h, 1008
- vvelib/algorithms/split.h, 1010
- vvelib/algorithms/tissue.h, 1011
- vvelib/algorithms/triangle_growth.h, 1013
- vvelib/bspline_tissue_model.h, 1015
- vvelib/factory/ Directory Reference, 62
- vvelib/factory/complex_grid.h, 1020
- vvelib/factory/grid.h, 1022
- vvelib/factory/objreader.h, 1023
- vvelib/geometry/ Directory Reference, 63
- vvelib/geometry/area.h, 1025
- vvelib/geometry/coordinates.h, 1026
- vvelib/geometry/geometry.h, 1027
- vvelib/geometry/intersection.h, 1028
- vvelib/geometry/projection.h, 1030
- vvelib/geometry/quaternion.h, 1031
- vvelib/graph/ Directory Reference, 64
- vvelib/graph/edge.h, 1032
- vvelib/graph/vertex.h, 1033
- vvelib/graph/vvbigraph.h, 1035
- vvelib/graph/vvgraph.h, 1037
- vvelib/model.h, 1039
- vvelib/shape/ Directory Reference, 65
- vvelib/shape/quadric.h, 1040
- vvelib/storage/ Directory Reference, 66
- vvelib/storage/complex.h, 1000
- vvelib/storage/config/ Directory Reference, 59
- vvelib/storage/graph.h, 1004
- vvelib/storage/storage.h, 1041
- vvelib/storage/storage_xml.h, 1043
- vvelib/storage/types.h, 1044
- vvelib/storage/vector.h, 1048
- vvelib/test/ Directory Reference, 67
- vvelib/test/graph.vvh, 1051
- vvelib/test/test.h, 1053
- vvelib/tissue_model.h, 1056
- vvelib/util/ Directory Reference, 68
- vvelib/util/assert.h, 1060
- vvelib/util/buffer.h, 1062
- vvelib/util/clamp.h, 1063
- vvelib/util/color.h, 1064
- vvelib/util/contour.h, 1066
- vvelib/util/dir.h, 1067
- vvelib/util/forall.h, 1068
- vvelib/util/function.h, 1073
- vvelib/util/gllerrorcheck.h, 1074
- vvelib/util/graph_inset.h, 1075
- vvelib/util/leaf_class.h, 1076
- vvelib/util/materials.h, 1077
- vvelib/util/member_iterator.h, 1078
- vvelib/util/palette.h, 1079
- vvelib/util/parms.h, 1080
- vvelib/util/point.h, 1081
- vvelib/util/random.h, 1082
- vvelib/util/range.h, 1084
- vvelib/util/static_assert.h, 1086
- vvelib/util/tensor.h, 1088
- vvelib/util/texture.h, 1089
- vvelib/util/tie.h, 1090
- vvelib/util/vector.h, 1049
- vvelib/util/watchdog.h, 1091
- VVGraph
 - graph::VVGraph, 922
- W
 - vvcomplex::VVComplex, 831
- w
 - geometry::Quaternion, 429
- wall
 - vvcomplex, 181
 - vvcomplex::VVComplex, 800
- wall_arc
 - vvcomplex::VVComplex, 800
- wall_graph
 - vvcomplex, 181
 - vvcomplex::VVComplex, 800
- wasInitialized
 - Viewer, 697
- watch
 - util::WatchDog, 977
- WatchDog
 - util::WatchDog, 974
- weakRef
 - graph::Vertex, 672
- WeakVertex

- graph::WeakVertex, [981](#), [982](#)
- windowPosition
 - util::GraphInset, [325](#)
- windowSize
 - util::GraphInset, [325](#)
- writeCSV
 - util::GraphInset, [321](#)
- writer
 - storage::old::VVEStorage_
BINReader, [874](#)
 - storage::VVEStorage, [866](#)
 - storage::VVEStorage_BINReader,
[883](#)
 - storage::VVEStorage_BINWriter,
[891](#)
 - storage::VVEStorage_XMLReader,
[899](#)
 - storage::VVEStorage_XMLWriter,
[906](#)
- x
 - util::Vector, [642](#), [643](#)
- y
 - util::Vector, [643](#), [644](#)
- z
 - util::Vector, [644](#)
- zero
 - util::Matrix, [354](#)