THE CLASSIC COLLABORATION

CLASSIC Reference Manual

F. C. Iselin

Abstract

This report describes the classes implemented in the CLASSIC class library, as defined by the CLASSIC collaboration for the development of a Class Library for Accelerator System SImulation and Control.

Contents

1 Overview **14**
- 1.1 Module Structure . . . 15
- 1.2 Hierarchy of Field Representations . . . 16
- 1.3 Hierarchy of Geometry Representations . . . 17
- 1.4 Beam Line Elements . . . 18
- 1.5 Active Elements . . . 19
- 1.6 Multipoles . . . 20
- 1.7 Beam Lines . . . 21
- 1.8 Algebra Objects . . . 22
- 1.9 Algorithms . . . 23
- 1.10 CLASSIC Exception Objects . . . 24

2 Detailed Class Descriptions **25**
- 2.1 namespace **Physics** . . . 26
  - 2.1.1 Detailed descriptions . . . 27
- 2.2 class **AbsFileStream** . . . 29
  - 2.2.1 Detailed descriptions . . . 30
- 2.3 class **AbstractMapper** . . . 32
  - 2.3.1 Detailed descriptions . . . 35
- 2.4 class **AbstractTracker** . . . 37
  - 2.4.1 Detailed descriptions . . . 39
- 2.5 class **AlignWrapper** . . . 41
  - 2.5.1 Detailed descriptions . . . 44
- 2.6 class **ArithmeticError** . . . 46
  - 2.6.1 Detailed descriptions . . . 47
- 2.7 template class **Array1D** <class> . . . 48
  - 2.7.1 Detailed descriptions . . . 49
- 2.8 template class **Array2D** <class> . . . 52
  - 2.8.1 Detailed descriptions . . . 54
- 2.9 class **AttributeError** . . . 59
  - 2.9.1 Detailed descriptions . . . 59
- 2.10 class **AttributeSet** . . . 60
  - 2.10.1 Detailed descriptions . . . 61
- 2.11 class **BDipoleField** . . . 63
  - 2.11.1 Detailed descriptions . . . 64
- 2.12 class **BMultipoleField** . . . 66
  - 2.12.1 Detailed descriptions . . . 67

2.13 template class **BSingleMultipoleField** <int> . . . 70
2.13.1 Detailed descriptions . . . 71
2.14 class **BVector** . . . 73
2.14.1 Detailed descriptions . . . 73
2.15 class **BeamBeam** . . . 74
2.15.1 Detailed descriptions . . . 77
2.16 class **BeamBeamRep** . . . 79
2.16.1 Detailed descriptions . . . 83
2.17 class **Beamline** . . . 85
2.17.1 Detailed descriptions . . . 88
2.18 class **BeamlineGeometry** . . . 89
2.18.1 Detailed descriptions . . . 90
2.19 class **BeamlineVisitor** . . . 92
2.19.1 Detailed descriptions . . . 94
2.20 class **Channel** . . . 96
2.20.1 Detailed descriptions . . . 97
2.21 class **ClassicException** . . . 98
2.21.1 Detailed descriptions . . . 99
2.22 class **ClassicParser** . . . 100
2.22.1 Detailed descriptions . . . 100
2.23 class **Collimator** . . . 102
2.23.1 Detailed descriptions . . . 105
2.24 class **CollimatorRep** . . . 107
2.24.1 Detailed descriptions . . . 110
2.25 class **ComplexEigen** . . . 112
2.25.1 Detailed descriptions . . . 112
2.26 class **Component** . . . 114
2.26.1 Detailed descriptions . . . 118
2.27 class **ConstBField** . . . 120
2.27.1 Detailed descriptions . . . 121
2.28 class **ConstBzField** . . . 123
2.28.1 Detailed descriptions . . . 124
2.29 class **ConstChannel** . . . 126
2.29.1 Detailed descriptions . . . 126
2.30 class **ConstEField** . . . 128
2.30.1 Detailed descriptions . . . 129
2.31 class **ConstEzField** . . . 131
2.31.1 Detailed descriptions . . . 132
2.32 template class **ConstSliceIterator** <class> . . . 133
2.32.1 Detailed descriptions . . . 134
2.33 class **ConvergenceError** . . . 136
2.33.1 Detailed descriptions . . . 136
2.34 class **Corrector** . . . 137
2.34.1 Detailed descriptions . . . 141
2.35 class **CorrectorRep** . . . 142
2.35.1 Detailed descriptions . . . 146
2.36 class **CorrectorWrapper** . . . 148
2.36.1 Detailed descriptions . . . 151

2.37 class **DefaultVisitor** . . . 154
2.37.1 Detailed descriptions . . . 156
2.38 class **Diagnostic** . . . 159
2.38.1 Detailed descriptions . . . 162
2.39 class **DirectChannel** . . . 163
2.39.1 Detailed descriptions . . . 164
2.40 class **DivideError** . . . 165
2.40.1 Detailed descriptions . . . 165
2.41 class **DomainError** . . . 166
2.41.1 Detailed descriptions . . . 166
2.42 class **DoubleEigen** . . . 167
2.42.1 Detailed descriptions . . . 167
2.43 template class **DragtFinnMap** <int> . . . 169
2.43.1 Detailed descriptions . . . 170
2.44 class **Drift** . . . 173
2.44.1 Detailed descriptions . . . 176
2.45 class **DriftRep** . . . 177
2.45.1 Detailed descriptions . . . 180
2.46 class **DynamicFixedPoint** . . . 182
2.46.1 Detailed descriptions . . . 182
2.47 class **EBVectors** . . . 183
2.47.1 Detailed descriptions . . . 183
2.48 class **EDipoleField** . . . 185
2.48.1 Detailed descriptions . . . 186
2.49 class **EMField** . . . 188
2.49.1 Detailed descriptions . . . 189
2.50 class **EVector** . . . 191
2.50.1 Detailed descriptions . . . 191
2.51 class **EigenvalueError** . . . 192
2.51.1 Detailed descriptions . . . 192
2.52 class **ElementBase** . . . 193
2.52.1 Detailed descriptions . . . 197
2.53 class **ElementFactory** . . . 202
2.53.1 Detailed descriptions . . . 202
2.54 class **ElementImage** . . . 204
2.54.1 Detailed descriptions . . . 205
2.55 class **ElmPtr** . . . 207
2.55.1 Detailed descriptions . . . 207
2.56 class **Euclid3D** . . . 209
2.56.1 Detailed descriptions . . . 211
2.57 template class **FArray1D** <class,int> . . . 214
2.57.1 Detailed descriptions . . . 215
2.58 template class **FArray2D** <class,int,int> . . . 217
2.58.1 Detailed descriptions . . . 219
2.59 template class **FComplexEigen** <int> . . . 223
2.59.1 Detailed descriptions . . . 223
2.60 template class **FConstSlice** <class,int> . . . 225
2.60.1 Detailed descriptions . . . 226

2.61 template class **FDoubleEigen** <int> . . . 228
2.61.1 Detailed descriptions . . . 228
2.62 template class **FDynamicFP** <int> . . . 230
2.62.1 Detailed descriptions . . . 230
2.63 template class **FLUMatrix** <class,int> . . . 231
2.63.1 Detailed descriptions . . . 231
2.64 template class **FLieGenerator** <class,int> . . . 233
2.64.1 Detailed descriptions . . . 234
2.65 template class **FMatrix** <class,int,int> . . . 237
2.65.1 Detailed descriptions . . . 240
2.66 template class **FMonomial** <int> . . . 241
2.66.1 Detailed descriptions . . . 241
2.67 template class **FNormalForm** <int> . . . 243
2.67.1 Detailed descriptions . . . 244
2.68 template class **FSlice** <class,int> . . . 245
2.68.1 Detailed descriptions . . . 246
2.69 template class **FStaticFP** <int> . . . 248
2.69.1 Detailed descriptions . . . 248
2.70 template class **FTps** <class,int> . . . 250
2.70.1 Detailed descriptions . . . 253
2.71 template class **FTpsData** <int> . . . 258
2.71.1 Detailed descriptions . . . 258
2.72 template class **FTpsRep** <class,int> . . . 260
2.73 template class **FVector** <class,int> . . . 261
2.73.1 Detailed descriptions . . . 262
2.74 template class **FVps** <class,int> . . . 264
2.74.1 Detailed descriptions . . . 266
2.75 class **Factory** . . . 270
2.75.1 Detailed descriptions . . . 270
2.76 class **FileStream** . . . 272
2.76.1 Detailed descriptions . . . 273
2.77 class **FlaggedElmPtr** . . . 274
2.77.1 Detailed descriptions . . . 275
2.78 class **Flagger** . . . 277
2.78.1 Detailed descriptions . . . 279
2.79 class **FormatError** . . . 280
2.79.1 Detailed descriptions . . . 280
2.80 class **Geometry** . . . 281
2.80.1 Detailed descriptions . . . 282
2.81 class **IdealMapper** . . . 285
2.81.1 Detailed descriptions . . . 289
2.82 template class **IndexedChannel** <class> . . . 291
2.82.1 Detailed descriptions . . . 292
2.83 template class **IndirectChannel** <class> . . . 293
2.83.1 Detailed descriptions . . . 294
2.84 class **Integrator** . . . 295
2.84.1 Detailed descriptions . . . 298
2.85 template class **LUMatrix** <class> . . . 300

2.85.1 Detailed descriptions . . . 300
2.86 class **Lambertson** . . . 302
2.86.1 Detailed descriptions . . . 305
2.87 template class **LieMap** <class> . . . 306
2.87.1 Detailed descriptions . . . 309
2.88 template class **LinearFun** <class,int> . . . 310
2.88.1 Detailed descriptions . . . 312
2.89 template class **LinearMap** <class,int> . . . 315
2.89.1 Detailed descriptions . . . 316
2.90 class **LinearMapper** . . . 319
2.90.1 Detailed descriptions . . . 323
2.91 class **LogicalError** . . . 326
2.91.1 Detailed descriptions . . . 326
2.92 class **MapIntegrator** . . . 327
2.92.1 Detailed descriptions . . . 330
2.93 class **Mapper** . . . 331
2.93.1 Detailed descriptions . . . 335
2.94 class **Marker** . . . 337
2.94.1 Detailed descriptions . . . 340
2.95 class **MarkerRep** . . . 341
2.95.1 Detailed descriptions . . . 344
2.96 template class **Matrix** <class> . . . 346
2.96.1 Detailed descriptions . . . 349
2.97 class **Matrix3D** . . . 351
2.97.1 Detailed descriptions . . . 352
2.98 class **Monitor** . . . 354
2.98.1 Detailed descriptions . . . 357
2.99 class **MonitorRep** . . . 359
2.99.1 Detailed descriptions . . . 363
2.100class **Multipole** . . . 365
2.100.1 Detailed descriptions . . . 369
2.101class **MultipoleRep** . . . 371
2.101.1 Detailed descriptions . . . 374
2.102class **MultipoleWrapper** . . . 376
2.102.1 Detailed descriptions . . . 380
2.103class **NormalForm** . . . 382
2.103.1 Detailed descriptions . . . 383
2.104class **NormalFormError** . . . 384
2.104.1 Detailed descriptions . . . 384
2.105class **NullField** . . . 385
2.105.1 Detailed descriptions . . . 386
2.106class **NullGeometry** . . . 387
2.106.1 Detailed descriptions . . . 388
2.107class **OffsetGeometry** . . . 390
2.107.1 Detailed descriptions . . . 391
2.108class **OrbitTracker** . . . 394
2.108.1 Detailed descriptions . . . 396
2.109template class **OscillatingField** <class> . . . 398

2.109.1 Detailed descriptions . . . 399
2.110 class **OverflowError** . . . 400
2.110.1 Detailed descriptions . . . 400
2.111 template class **OwnPtr** <class> . . . 401
2.111.1 Detailed descriptions . . . 401
2.112 class **ParseError** . . . 403
2.112.1 Detailed descriptions . . . 403
2.113 class **Parser** . . . 404
2.113.1 Detailed descriptions . . . 404
2.114 class **PartBunch** . . . 405
2.114.1 Detailed descriptions . . . 405
2.115 class **PartData** . . . 407
2.115.1 Detailed descriptions . . . 408
2.116 class **Particle** . . . 410
2.116.1 Detailed descriptions . . . 411
2.117 class **Patch** . . . 413
2.117.1 Detailed descriptions . . . 416
2.118 class **PatchRep** . . . 418
2.118.1 Detailed descriptions . . . 422
2.119 class **PlanarArcGeometry** . . . 425
2.119.1 Detailed descriptions . . . 427
2.120 class **Point3D** . . . 429
2.120.1 Detailed descriptions . . . 429
2.121 template class **Pointer** <class> . . . 430
2.121.1 Detailed descriptions . . . 431
2.122 class **RBend** . . . 432
2.122.1 Detailed descriptions . . . 436
2.123 class **RBendGeometry** . . . 438
2.123.1 Detailed descriptions . . . 439
2.124 class **RBendRep** . . . 441
2.124.1 Detailed descriptions . . . 445
2.125 class **RBendWrapper** . . . 448
2.125.1 Detailed descriptions . . . 452
2.126 class **RCObject** . . . 455
2.126.1 Detailed descriptions . . . 456
2.127 class **RFCavity** . . . 457
2.127.1 Detailed descriptions . . . 460
2.128 class **RFCavityRep** . . . 462
2.128.1 Detailed descriptions . . . 466
2.129 class **RFQuadrupole** . . . 468
2.129.1 Detailed descriptions . . . 471
2.130 class **Random** . . . 472
2.130.1 Detailed descriptions . . . 472
2.131 class **RangeError** . . . 474
2.131.1 Detailed descriptions . . . 474
2.132 class **Rotation3D** . . . 475
2.132.1 Detailed descriptions . . . 476
2.133 class **SBend** . . . 478

2.133.1 Detailed descriptions . . . 482
2.134 class **SBendRep** . . . 484
2.134.1 Detailed descriptions . . . 488
2.135 class **SBendWrapper** . . . 491
2.135.1 Detailed descriptions . . . 495
2.136 class **SRotatedGeometry** . . . 498
2.136.1 Detailed descriptions . . . 500
2.137 class **Separator** . . . 503
2.137.1 Detailed descriptions . . . 506
2.138 class **SeparatorRep** . . . 508
2.138.1 Detailed descriptions . . . 511
2.139 class **Septum** . . . 514
2.139.1 Detailed descriptions . . . 517
2.140 class **SimpleStatement** . . . 518
2.140.1 Detailed descriptions . . . 519
2.141 template class **SingleMultipole** <int> . . . 521
2.141.1 Detailed descriptions . . . 525
2.142 class **SingularMatrixError** . . . 527
2.142.1 Detailed descriptions . . . 527
2.143 class **SizeError** . . . 528
2.143.1 Detailed descriptions . . . 528
2.144 template class **SliceIterator** <class> . . . 530
2.144.1 Detailed descriptions . . . 531
2.145 class **Solenoid** . . . 533
2.145.1 Detailed descriptions . . . 536
2.146 class **SolenoidRep** . . . 538
2.146.1 Detailed descriptions . . . 541
2.147 class **Statement** . . . 543
2.147.1 Detailed descriptions . . . 545
2.148 class **StaticElectricField** . . . 548
2.148.1 Detailed descriptions . . . 549
2.149 class **StaticFixedPoint** . . . 550
2.149.1 Detailed descriptions . . . 550
2.150 class **StaticMagneticField** . . . 552
2.150.1 Detailed descriptions . . . 553
2.151 class **StraightGeometry** . . . 554
2.151.1 Detailed descriptions . . . 555
2.152 class **StringStream** . . . 557
2.152.1 Detailed descriptions . . . 557
2.153 class **Surveyor** . . . 559
2.153.1 Detailed descriptions . . . 561
2.154 template class **TBeamline** <class> . . . 562
2.154.1 Detailed descriptions . . . 565
2.155 template class **Taylor** <class> . . . 568
2.155.1 Detailed descriptions . . . 569
2.156 class **TerminalStream** . . . 571
2.156.1 Detailed descriptions . . . 572
2.157 class **ThinMapper** . . . 573

2.157.1 Detailed descriptions . . . 576
2.158 class **ThinTracker** . . . 578
2.158.1 Detailed descriptions . . . 580
2.159 class **Token** . . . 582
2.159.1 Detailed descriptions . . . 583
2.160 class **TokenStream** . . . 586
2.160.1 Detailed descriptions . . . 587
2.161 template class **Tps** <class> . . . 588
2.161.1 Detailed descriptions . . . 591
2.162 class **TpsData** . . . 596
2.162.1 Detailed descriptions . . . 597
2.163 class **TpsMonomial** . . . 598
2.163.1 Detailed descriptions . . . 598
2.164 template class **TpsRep** <class> . . . 600
2.165 class **TrackIntegrator** . . . 601
2.165.1 Detailed descriptions . . . 604
2.166 class **Tracker** . . . 605
2.166.1 Detailed descriptions . . . 608
2.167 template class **TransportFun** <class,int> . . . 611
2.167.1 Detailed descriptions . . . 613
2.168 template class **TransportMap** <class,int> . . . 616
2.168.1 Detailed descriptions . . . 617
2.169 template class **Vector** <class> . . . 620
2.169.1 Detailed descriptions . . . 621
2.170 class **Vector3D** . . . 623
2.170.1 Detailed descriptions . . . 624
2.171 template class **Vps** <class> . . . 626
2.171.1 Detailed descriptions . . . 628
2.172 template class **VpsInvMap** <class> . . . 631
2.172.1 Detailed descriptions . . . 634
2.173 template class **VpsMap** <class> . . . 635
2.173.1 Detailed descriptions . . . 637
2.174 class **XCorrectorRep** . . . 640
2.174.1 Detailed descriptions . . . 644
2.175 class **XMonitorRep** . . . 645
2.175.1 Detailed descriptions . . . 648
2.176 class **YCorrectorRep** . . . 650
2.176.1 Detailed descriptions . . . 654
2.177 class **YMonitorRep** . . . 655
2.177.1 Detailed descriptions . . . 658
2.178 template class **complex** <class> . . . 660

List of Figures

1.1 CLASSIC Modules and their Dependences 15
1.2 Hierarchy for Electromagnetic Fields 16
1.3 Hierarchy for Geometry Objects 17
1.4 Hierarchy for Drift Elements 18
1.5 Hierarchy for RBend Elements 19
1.6 Hierarchy for Multipole Elements 20
1.7 Beam Line Hierarchy 21
1.8 Algebra Objects with Variable Dimensions 22
1.9 Algebra Objects with Fixed Dimensions 22
1.10 Algorithms 23
1.11 Integrator Objects 23
1.12 CLASSIC Exception Objects 24

2.1 Inheritance for class AbsFileStream 29
2.2 Inheritance for class AbstractMapper 32
2.3 Inheritance for class AbstractTracker 37
2.4 Inheritance for class AlignWrapper 41
2.5 Inheritance for class ArithmeticError 46
2.6 Inheritance for class Array1D 48
2.7 Inheritance for class Array2D 52
2.8 Inheritance for class AttributeError 59
2.9 Inheritance for class AttributeSet 60
2.10 Inheritance for class BDipoleField 63
2.11 Inheritance for class BMultipoleField 66
2.12 Inheritance for class BSingleMultipoleField 70
2.13 Inheritance for class BeamBeam 74
2.14 Inheritance for class BeamBeamRep 79
2.15 Inheritance for class Beamline 85
2.16 Inheritance for class BeamlineGeometry 89
2.17 Inheritance for class BeamlineVisitor 92
2.18 Inheritance for class Channel 96
2.19 Inheritance for class ClassicException 98
2.20 Inheritance for class ClassicParser 100
2.21 Inheritance for class Collimator 102
2.22 Inheritance for class CollimatorRep 107
2.23 Inheritance for class Component 115
2.24 Inheritance for class ConstBField 120
2.25 Inheritance for class ConstBzField 123

2.26 Inheritance for class ConstChannel . . . 126
2.27 Inheritance for class ConstEField . . . 128
2.28 Inheritance for class ConstEzField . . . 131
2.29 Inheritance for class ConvergenceError . . . 136
2.30 Inheritance for class Corrector . . . 137
2.31 Inheritance for class CorrectorRep . . . 142
2.32 Inheritance for class CorrectorWrapper . . . 148
2.33 Inheritance for class DefaultVisitor . . . 154
2.34 Inheritance for class Diagnostic . . . 159
2.35 Inheritance for class DirectChannel . . . 163
2.36 Inheritance for class DivideError . . . 165
2.37 Inheritance for class DomainError . . . 166
2.38 Inheritance for class Drift . . . 173
2.39 Inheritance for class DriftRep . . . 177
2.40 Inheritance for class EDipoleField . . . 185
2.41 Inheritance for class EMField . . . 188
2.42 Inheritance for class EigenvalueError . . . 192
2.43 Inheritance for class ElementBase . . . 194
2.44 Inheritance for class ElementFactory . . . 202
2.45 Inheritance for class ElementImage . . . 204
2.46 Inheritance for class ElmPtr . . . 207
2.47 Inheritance for class FArray1D . . . 214
2.48 Inheritance for class FArray2D . . . 217
2.49 Inheritance for class FMatrix . . . 237
2.50 Inheritance for class FVector . . . 261
2.51 Inheritance for class Factory . . . 270
2.52 Inheritance for class FileStream . . . 272
2.53 Inheritance for class FlaggedElmPtr . . . 274
2.54 Inheritance for class Flagger . . . 277
2.55 Inheritance for class FormatError . . . 280
2.56 Inheritance for class Geometry . . . 281
2.57 Inheritance for class IdealMapper . . . 285
2.58 Inheritance for class IndexedChannel . . . 291
2.59 Inheritance for class IndirectChannel . . . 293
2.60 Inheritance for class Integrator . . . 295
2.61 Inheritance for class Lambertson . . . 302
2.62 Inheritance for class LieMap . . . 306
2.63 Inheritance for class LinearMapper . . . 319
2.64 Inheritance for class LogicalError . . . 326
2.65 Inheritance for class MapIntegrator . . . 327
2.66 Inheritance for class Mapper . . . 331
2.67 Inheritance for class Marker . . . 337
2.68 Inheritance for class MarkerRep . . . 341
2.69 Inheritance for class Matrix . . . 346
2.70 Inheritance for class Monitor . . . 354
2.71 Inheritance for class MonitorRep . . . 359
2.72 Inheritance for class Multipole . . . 365
2.73 Inheritance for class MultipoleRep . . . 371

2.74 Inheritance for class MultipoleWrapper . . . 376
2.75 Inheritance for class NormalFormError . . . 384
2.76 Inheritance for class NullField . . . 385
2.77 Inheritance for class NullGeometry . . . 387
2.78 Inheritance for class OffsetGeometry . . . 390
2.79 Inheritance for class OrbitTracker . . . 394
2.80 Inheritance for class OscillatingField . . . 398
2.81 Inheritance for class OverflowError . . . 400
2.82 Inheritance for class ParseError . . . 403
2.83 Inheritance for class Parser . . . 404
2.84 Inheritance for class PartBunch . . . 405
2.85 Inheritance for class Patch . . . 413
2.86 Inheritance for class PatchRep . . . 418
2.87 Inheritance for class PlanarArcGeometry . . . 425
2.88 Inheritance for class RBend . . . 432
2.89 Inheritance for class RBendGeometry . . . 438
2.90 Inheritance for class RBendRep . . . 441
2.91 Inheritance for class RBendWrapper . . . 448
2.92 Inheritance for class RCObject . . . 455
2.93 Inheritance for class RFCavity . . . 457
2.94 Inheritance for class RFCavityRep . . . 462
2.95 Inheritance for class RFQuadrupole . . . 468
2.96 Inheritance for class RangeError . . . 474
2.97 Inheritance for class SBend . . . 478
2.98 Inheritance for class SBendRep . . . 484
2.99 Inheritance for class SBendWrapper . . . 491
2.100Inheritance for class SRotatedGeometry . . . 498
2.101Inheritance for class Separator . . . 503
2.102Inheritance for class SeparatorRep . . . 508
2.103Inheritance for class Septum . . . 514
2.104Inheritance for class SimpleStatement . . . 518
2.105Inheritance for class SingleMultipole . . . 521
2.106Inheritance for class SingularMatrixError . . . 527
2.107Inheritance for class SizeError . . . 528
2.108Inheritance for class Solenoid . . . 533
2.109Inheritance for class SolenoidRep . . . 538
2.110Inheritance for class Statement . . . 543
2.111Inheritance for class StaticElectricField . . . 548
2.112Inheritance for class StaticMagneticField . . . 552
2.113Inheritance for class StraightGeometry . . . 554
2.114Inheritance for class StringStream . . . 557
2.115Inheritance for class Surveyor . . . 559
2.116Inheritance for class TBeamline . . . 562
2.117Inheritance for class TerminalStream . . . 571
2.118Inheritance for class ThinMapper . . . 573
2.119Inheritance for class ThinTracker . . . 578
2.120Inheritance for class TokenStream . . . 586
2.121Inheritance for class TrackIntegrator . . . 601

2.122Inheritance for class Tracker 605
2.123Inheritance for class Vector 620
2.124Inheritance for class Vps 626
2.125Inheritance for class VpsInvMap 631
2.126Inheritance for class VpsMap 635
2.127Inheritance for class XCorrectorRep 640
2.128Inheritance for class XMonitorRep 645
2.129Inheritance for class YCorrectorRep 650
2.130Inheritance for class YMonitorRep 655

Chapter 1

Overview

1.1 Module Structure

The CLASSIC classes are grouped in several modules. The principal dependencies between the modules are shown in Fig. 1.1.

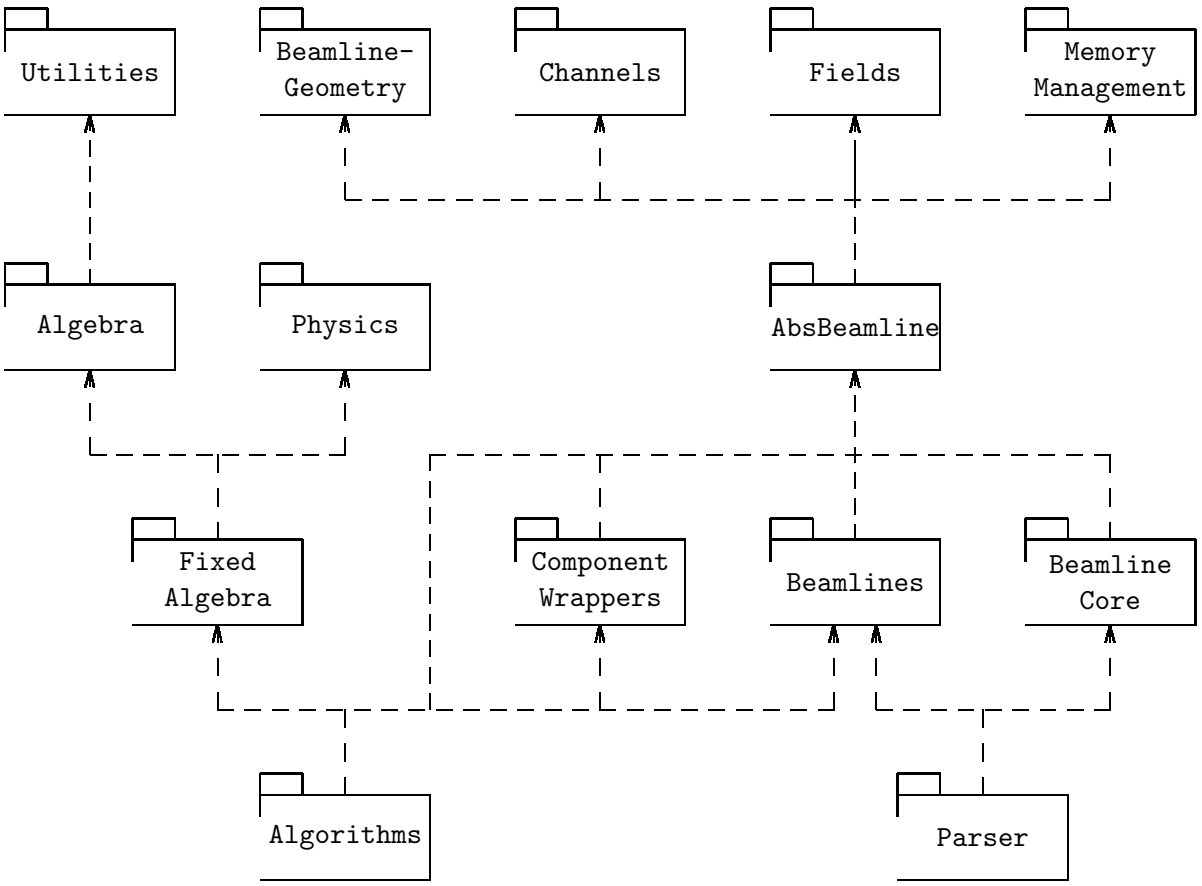

Figure 1.1: CLASSIC Modules and their Dependences

1.2 Hierarchy of Field Representations

The magnetic field hierarchy is shown in Fig. 1.2.

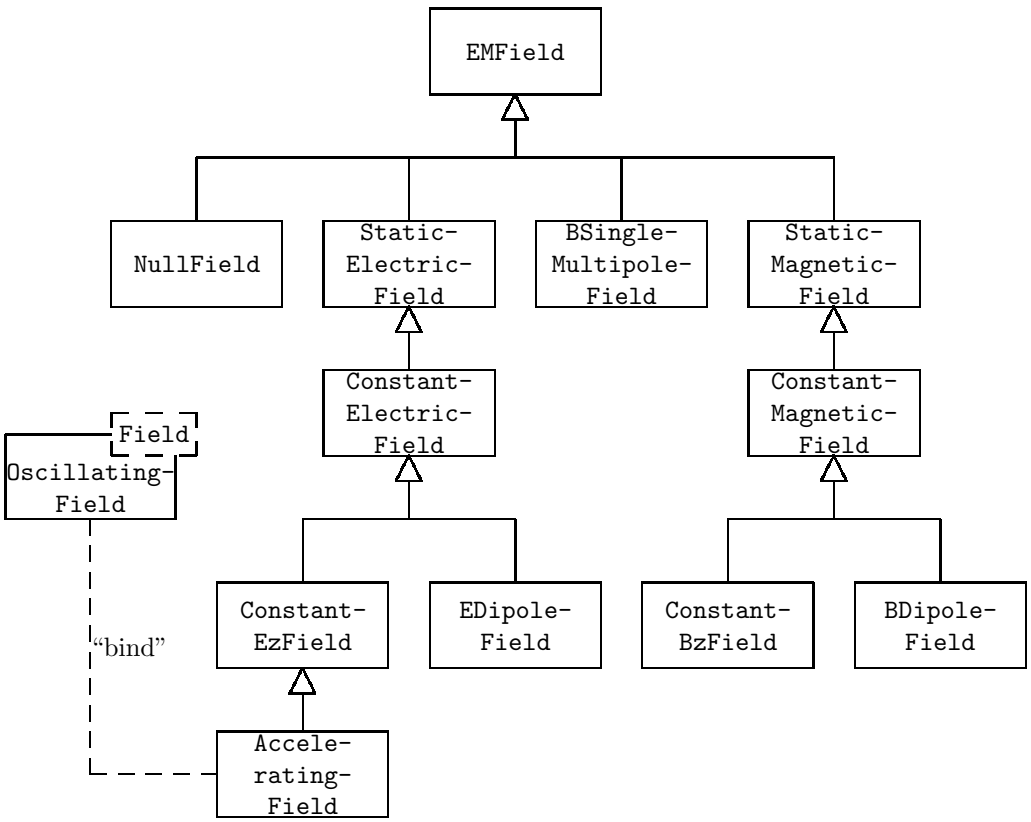

Figure 1.2: Hierarchy for Electromagnetic Fields

1.3 Hierarchy of Geometry Representations

The geometry of each element is represented by its geometry object. The geometry hierarchy is shown in Fig. 1.3.

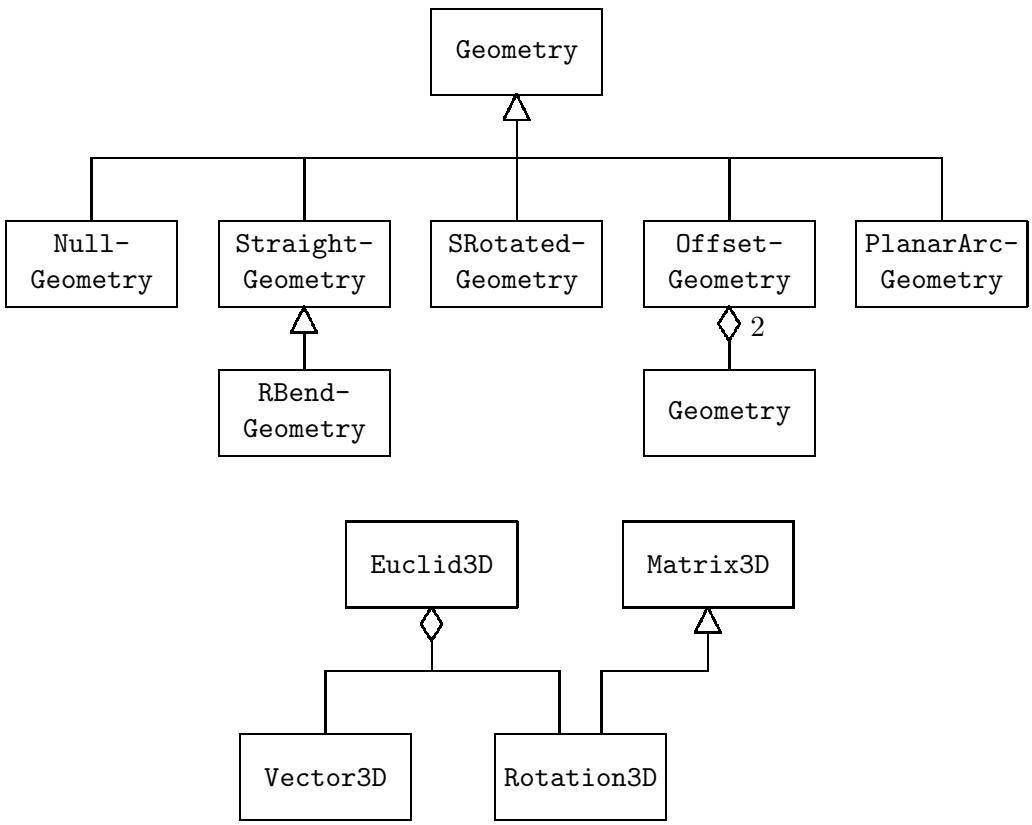

Figure 1.3: Hierarchy for Geometry Objects

1.4 Beam Line Elements

Most inactive elements follow a hierarchy as illustrated by the drift space 1.4

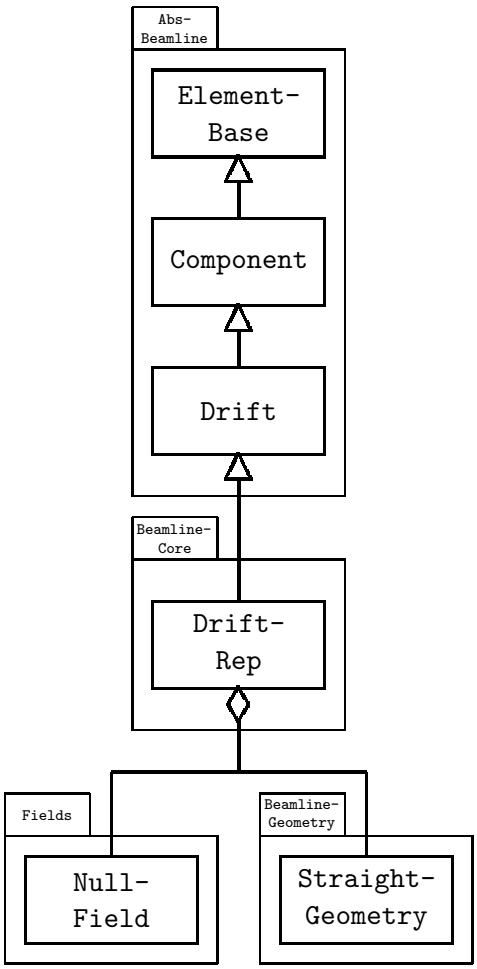

Figure 1.4: Hierarchy for Drift Elements. A similar hierarchy exists for all other elements.

1.5 Active Elements

All active elements have a hierarchy similar to the example for RBend elemens shown in Fig. 1.5.

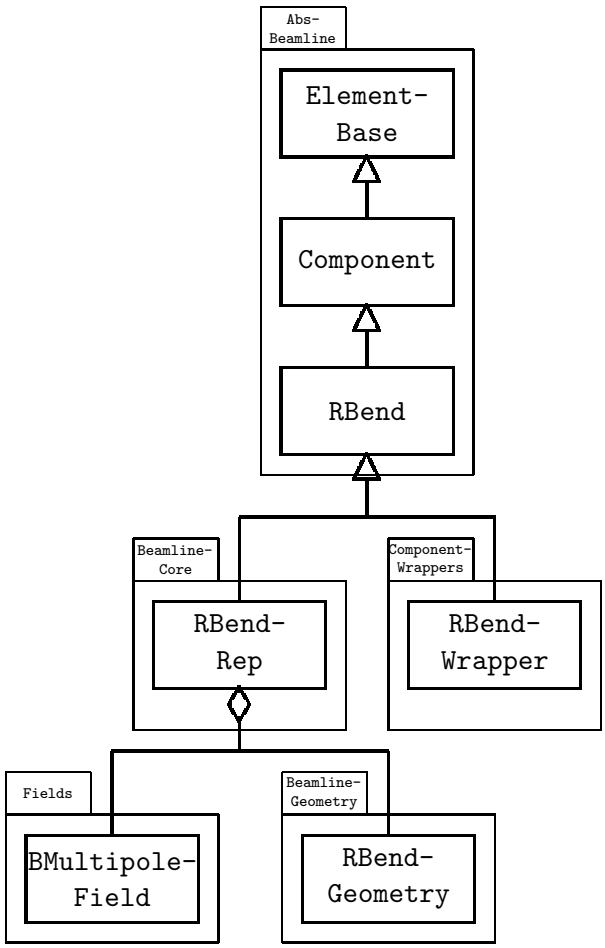

Figure 1.5: Hierarchy for RBend Elements

1.6 Multipoles

The multipoles have an extended hierarchy as shown in Fig. 1.6.

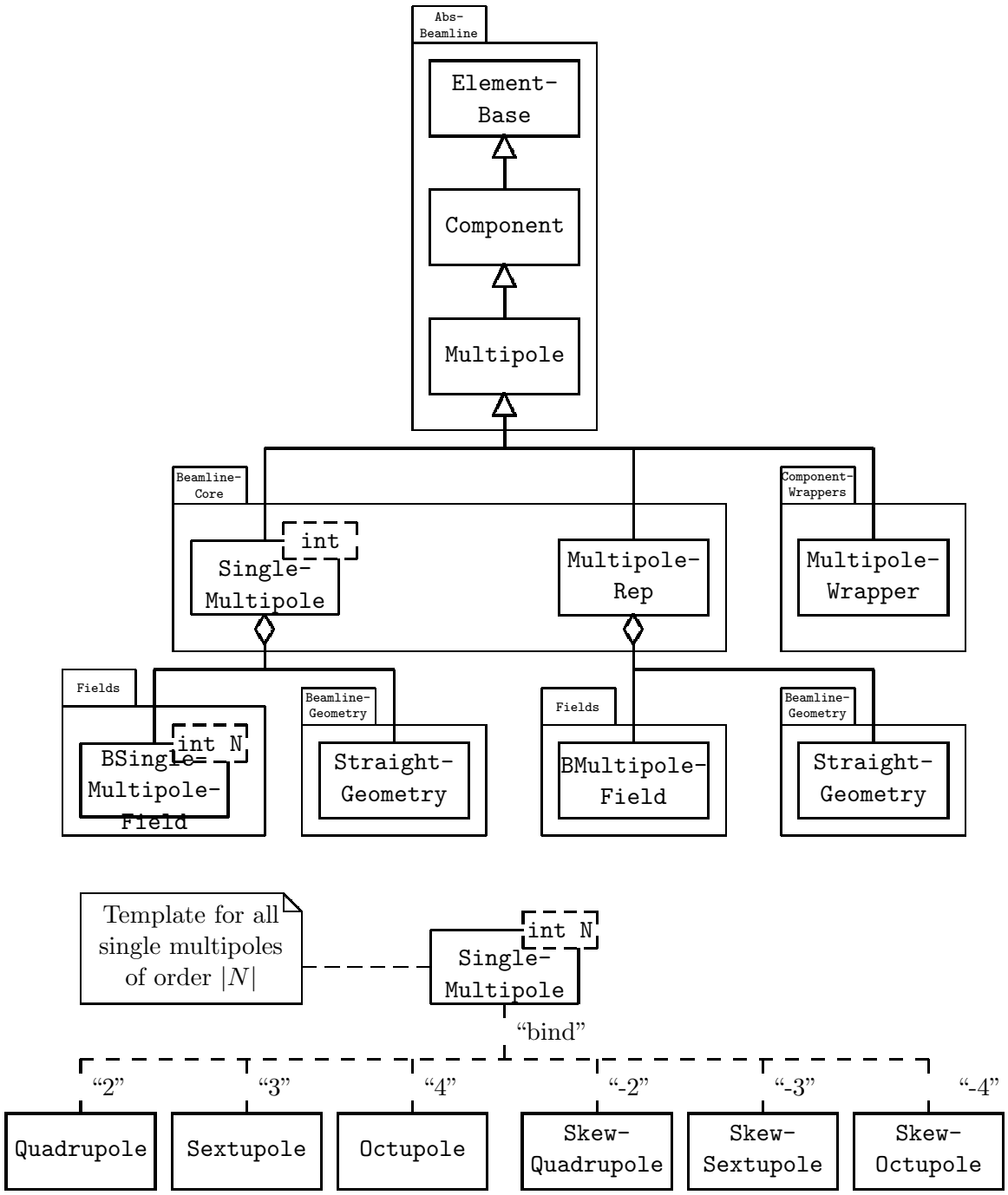

Figure 1.6: Hierarchy for Multipole Elements

1.7 Beam Lines

Beam lines are built from the hierarchy as shown in Fig. 1.7.

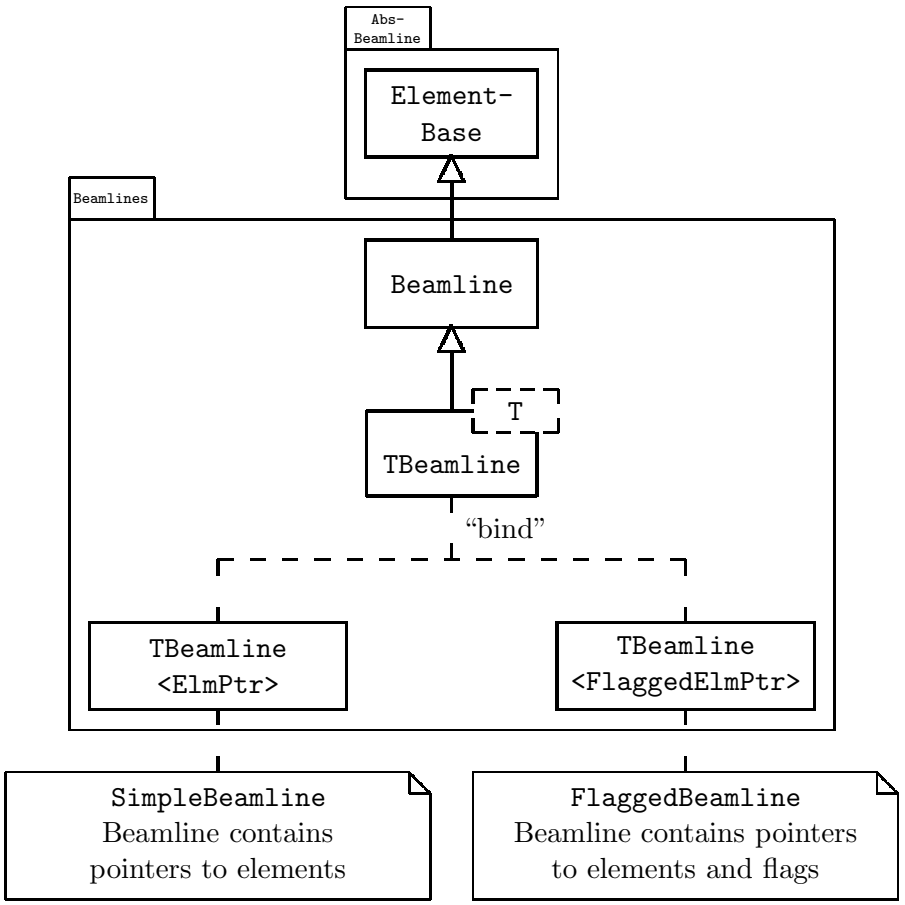

Figure 1.7: Beam Line Hierarchy

1.8 Algebra Objects

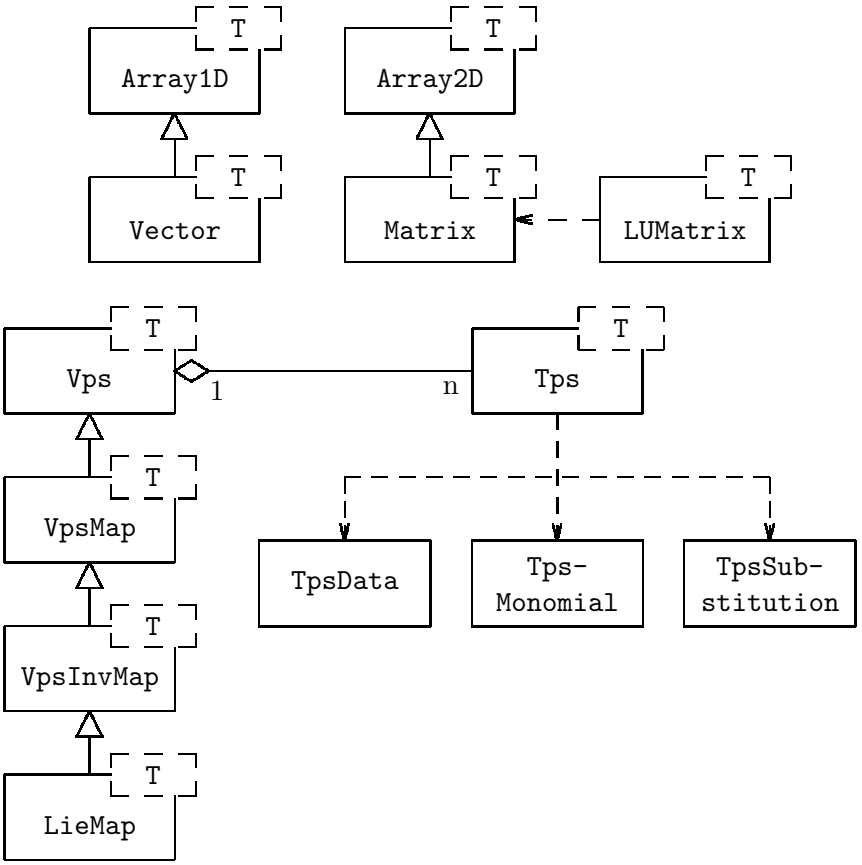

Figure 1.8: Algebra Objects with Variable Dimensions

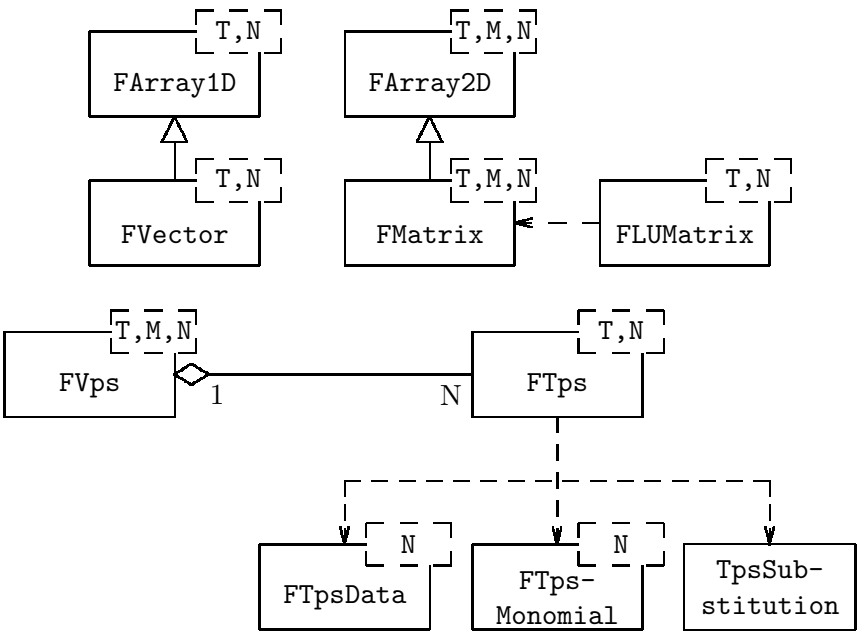

Figure 1.9: Algebra Objects with Fixed Dimensions

1.9 Algorithms

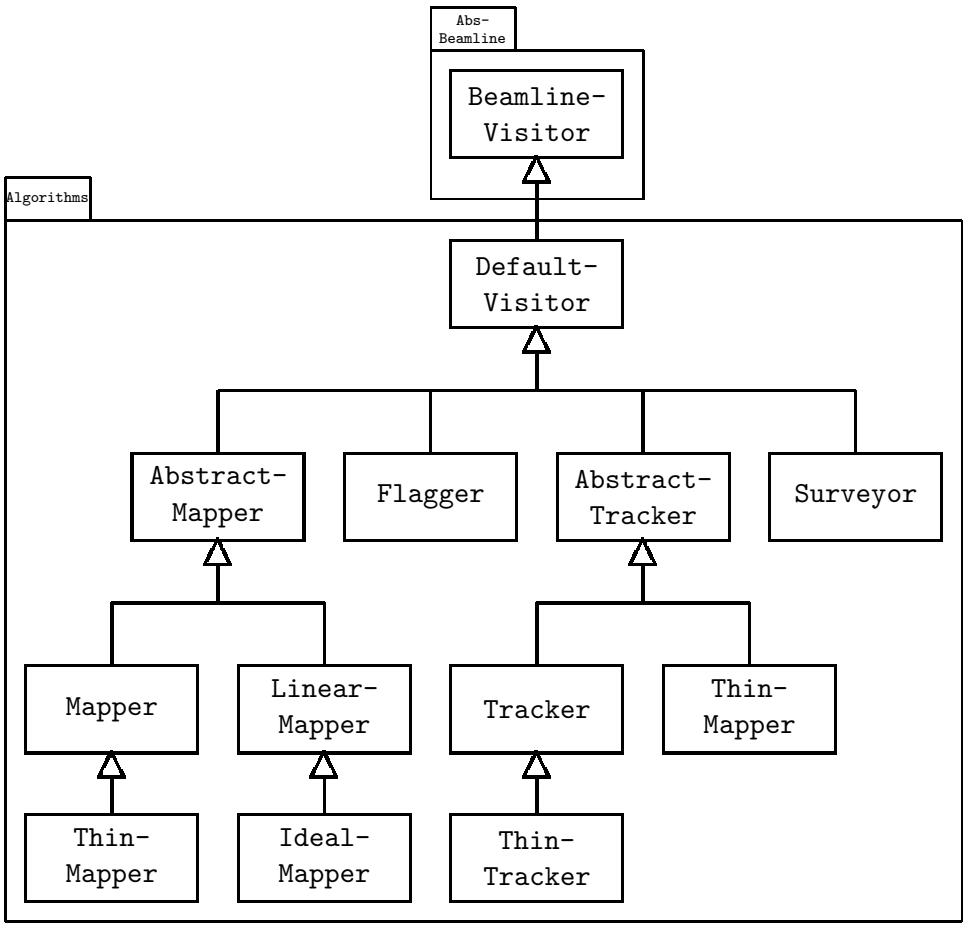

Figure 1.10: Algorithms

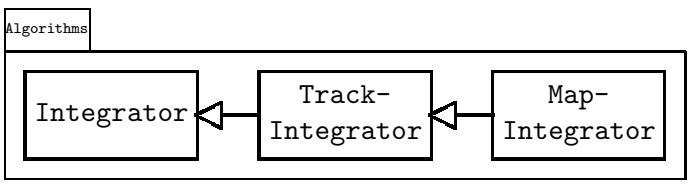

Figure 1.11: Integrator Objects

1.10 CLASSIC Exception Objects

The possible exceptions which can be thrown by CLASSIC routines have been arranged in the hierarchy shown in Fig. 1.12.

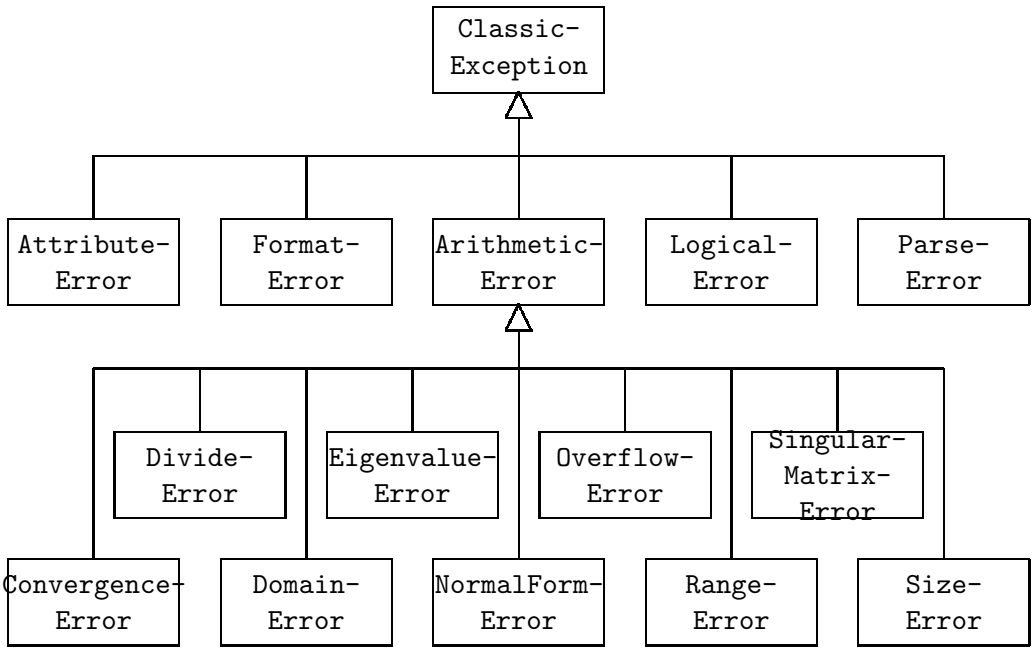

Figure 1.12: CLASSIC Exception Objects

Chapter 2

Detailed Class Descriptions

2.1 namespace Physics

A namespace defining various mathematical and physical constants.

Include file:	./Physics/Physics.hh

Synopsis

Members

- **extern const double a_e**
  The magnetic momentum anomaly for electrons, no dimension (see Section 2.1.1)
- **extern const double a_p**
  The magnetic momentum anomaly for protons, no dimension (see Section 2.1.1)
- **extern const double alpha**
  The fine structure constant, no dimension (see Section 2.1.1)
- **extern const double c**
  The velocity of light in m/s (see Section 2.1.1)
- **extern const double e**
  The value of e (see Section 2.1.1)
- **extern const double epsilon_0**
  The permittivity of vacuum in As/Vm (see Section 2.1.1)
- **extern const double h_bar**
  The reduced Planck constant in GeVs (see Section 2.1.1)
- **extern const double lamda_e**
  The reduced Compton wave length for electrons in m (see Section 2.1.1)
- **extern const double lamda_p**
  The reduced Compton wave length for protons in m (see Section 2.1.1)
- **extern const double log10e**
  The logarithm of e to the base 10 (see Section 2.1.1)
- **extern const double m_e**
  The electron rest mass in GeV (see Section 2.1.1)
- **extern const double m_p**
  The proton rest mass in GeV (see Section 2.1.1)
- **extern const double mu_0**
  The permeability of vacuum in Vs/Am (see Section 2.1.1)
- **extern const double pi**
  The value of π (see Section 2.1.1)
- **extern const double q_e**
  The elementary charge in As (see Section 2.1.1)

- **extern const double r_e**
  The classical electron radius in m (see Section 2.1.1)
- **extern const double r_p**
  The classical proton radius in m (see Section 2.1.1)
- **extern const double two_pi**
  The value of 2π (see Section 2.1.1)
- **extern const double u_two_pi**
  The value of $1/(2\pi)$ (see Section 2.1.1)

2.1.1 Detailed descriptions

Public members

- **extern const double a_e**
  The magnetic momentum anomaly for electrons, no dimension
- **extern const double a_p**
  The magnetic momentum anomaly for protons, no dimension
- **extern const double alpha**
  The fine structure constant, no dimension
- **extern const double c**
  The velocity of light in m/s
- **extern const double e**
  The value of e
- **extern const double epsilon_0**
  The permittivity of vacuum in As/Vm
- **extern const double h_bar**
  The reduced Planck constant in GeVs
- **extern const double lamda_e**
  The reduced Compton wave length for electrons in m
- **extern const double lamda_p**
  The reduced Compton wave length for protons in m
- **extern const double log10e**
  The logarithm of e to the base 10
- **extern const double m_e**
  The electron rest mass in GeV
- **extern const double m_p**
  The proton rest mass in GeV
- **extern const double mu_0**
  The permeability of vacuum in Vs/Am

- **extern const double pi**
  The value of π
- **extern const double q_e**
  The elementary charge in As
- **extern const double r_e**
  The classical electron radius in m
- **extern const double r_p**
  The classical proton radius in m
- **extern const double two_pi**
  The value of 2π
- **extern const double u_two_pi**
  The value of $1/(2\pi)$

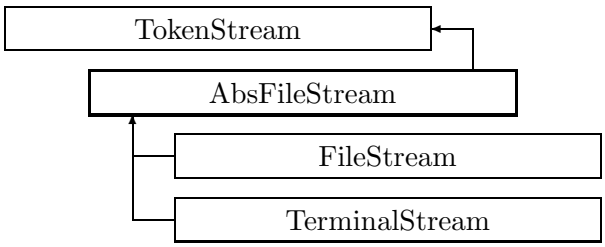

Figure 2.1: Inheritance for class AbsFileStream

2.2 class AbsFileStream

A stream of input tokens.
The source of tokens is an abstract file stream.

Type:	abstract
Superclasses:	public TokenStream
Include file:	./Parser/AbsFileStream.hh

Synopsis (including inherited members)

Public members

- **AbsFileStream (const string&)**
  Constructor. (see Section 2.2.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual bool fillLine ()**
  Read next input line. (see Section 2.2.1)
- **int getLine ()const**
  Return line number. (see Section 2.160.1)
- **const string& getName ()const**
  Return stream name. (see Section 2.160.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **void putBack (const Token&)**
  Put token back to stream. (see Section 2.160.1)
- **virtual Token readToken ()**
  Read single token from file. (see Section 2.2.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ~AbsFileStream ()**
  (see Section 2.2.1)

Protected members

- **int curr_char**
  (see Section 2.2.1)
- **int curr_line**
  (see Section 2.160.1)
- **string line**
  (see Section 2.2.1)
- **RCObject& operator= (const RCObject&)**
  (see Section 2.126.1)
- **Token put_back**
  (see Section 2.160.1)
- **bool put_back_flag**
  (see Section 2.160.1)
- **Token readNumber ()**
  (see Section 2.2.1)
- **Token readString ()**
  (see Section 2.2.1)
- **Token readWord ()**
  (see Section 2.2.1)
- **bool skipComments ()**
  (see Section 2.2.1)
- **string stream_name**
  (see Section 2.160.1)

2.2.1 Detailed descriptions

Public members

- **AbsFileStream (const string&)**
  Constructor.

  Store stream name.
- **virtual bool fillLine ()**
  Read next input line.
- **virtual Token readToken ()**
  Read single token from file.

  Returns false when end of file is hit.
- **virtual ~AbsFileStream ()**

Protected members

- **int curr_char**
- **string line**
- **Token readNumber ()**
- **Token readString ()**
- **Token readWord ()**
- **bool skipComments ()**

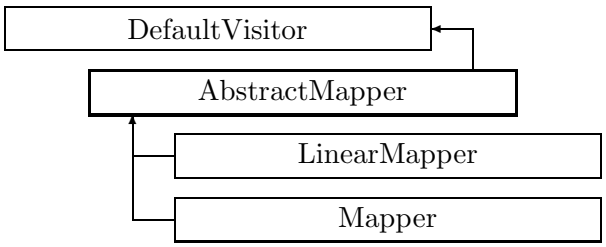

Figure 2.2: Inheritance for class AbstractMapper

2.3 class AbstractMapper

Build transfer map.
An abstract visitor class implementing the default behaviour for all visitors capable of tracking a transfer map through a beam line. It implements access to the accumulated map, and keeps track of the beam reference data. This class redefines all visitXXX() methods for elements as pure to force their implementation in derived classes.

Type:	abstract
Superclasses:	public DefaultVisitor
Include file:	./Algorithms/AbstractMapper.hh

Synopsis (including inherited members)

Public members

- **AbstractMapper (const Beamline&,const PartData&,bool,bool)**
  Constructor. (see Section 2.3.1)
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **virtual void getMap (LinearMap<double,6>&)const**
  Return the linear part of the accumulated map. (see Section 2.3.1)
- **virtual void getMap (FVps<double,6>&)const**
  Return the full map accumulated so far. (see Section 2.3.1)
- **virtual void setMap (const LinearMap<double,6>&)**
  Reset the linear part of the accumulated map for restart. (see Section 2.3.1)
- **virtual void setMap (const FVps<double,6>&)**
  Reset the full map for restart. (see Section 2.3.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper.. (see Section 2.37.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam. (see Section 2.3.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)

- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.3.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.3.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector. (see Section 2.3.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.37.1)
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic. (see Section 2.3.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift. (see Section 2.3.1)
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.37.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson. (see Section 2.3.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.37.1)
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker. (see Section 2.3.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor. (see Section 2.3.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole. (see Section 2.3.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.37.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.3.1)
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend. (see Section 2.3.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.37.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity. (see Section 2.3.1)

- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole. (see Section 2.3.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend. (see Section 2.3.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.37.1)
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator. (see Section 2.3.1)
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum. (see Section 2.3.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid. (see Section 2.3.1)
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.37.1)
- **virtual ~AbstractMapper ()**
  (see Section 2.3.1)

Protected members

- **bool back_beam**
  (see Section 2.37.1)
- **bool back_track**
  (see Section 2.37.1)
- **FTps<double,6> buildMultipoleVectorPotential (const BMultipoleField&)**
  Construct the vector potential for a Multipole. (see Section 2.3.1)
- **FTps<double,6> buildSBendVectorPotential (const BMultipoleField&,double)**
  Construct the vector potential for a SBend. (see Section 2.3.1)
- **double flip_B**
  (see Section 2.37.1)
- **double flip_s**
  (see Section 2.37.1)
- **const Beamline& itsLine**
  (see Section 2.37.1)
- **const PartData itsReference**
  The reference information. (see Section 2.3.1)
- **bool local_flip**
  (see Section 2.37.1)

2.3.1 Detailed descriptions

Public members

- **AbstractMapper (const Beamline&,const PartData&,bool,bool)**
  Constructor.

  The beam line to be tracked is **bl**. The particle reference data are taken from **data**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.

- **virtual void getMap (LinearMap<double,6>&)const**
  Return the linear part of the accumulated map.

- **virtual void getMap (FVps<double,6>&)const**
  Return the full map accumulated so far.

- **virtual void setMap (const LinearMap<double,6>&)**
  Reset the linear part of the accumulated map for restart.

- **virtual void setMap (const FVps<double,6>&)**
  Reset the full map for restart.

- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam.

- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator.

- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component.

- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector.

- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic.

- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift.

- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson.

- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker.

- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor.

- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole.

- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch.

- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend.
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity.
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole.
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend.
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator.
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum.
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid.
- **virtual ~AbstractMapper ()**

Protected members

- **FTps<double,6> buildMultipoleVectorPotential (const BMultipoleField&)**
  Construct the vector potential for a Multipole.
- **FTps<double,6> buildSBendVectorPotential (const BMultipoleField&,double)**
  Construct the vector potential for a SBend.
- **const PartData itsReference**
  The reference information.

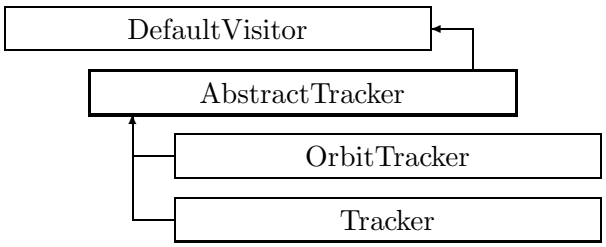

Figure 2.3: Inheritance for class AbstractTracker

2.4 class AbstractTracker

Track particles or bunches.
An abstract base class for all visitors capable of tracking particles through a beam element. This class redefines all visitXXX() methods for elements as pure to force their implementation in derived classes.

Type:	abstract
Superclasses:	public DefaultVisitor
Include file:	./Algorithms/AbstractTracker.hh

Synopsis (including inherited members)

Public members

- **AbstractTracker (const Beamline&,const PartData&,bool,bool)**
  Constructor. (see Section 2.4.1)
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper.. (see Section 2.4.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam. (see Section 2.4.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.4.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.4.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector. (see Section 2.4.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.37.1)

- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic. (see Section 2.4.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift. (see Section 2.4.1)
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.37.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson. (see Section 2.4.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.37.1)
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker. (see Section 2.4.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor. (see Section 2.4.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole. (see Section 2.4.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.37.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.4.1)
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend. (see Section 2.4.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.37.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity. (see Section 2.4.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole. (see Section 2.4.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend. (see Section 2.4.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.37.1)
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator. (see Section 2.4.1)

- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum. (see Section 2.4.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid. (see Section 2.4.1)
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.37.1)
- **virtual ~AbstractTracker ()**
  (see Section 2.4.1)

Protected members

- **bool back_beam**
  (see Section 2.37.1)
- **bool back_track**
  (see Section 2.37.1)
- **double flip_B**
  (see Section 2.37.1)
- **double flip_s**
  (see Section 2.37.1)
- **const Beamline& itsLine**
  (see Section 2.37.1)
- **const PartData itsReference**
  The reference information. (see Section 2.4.1)
- **bool local_flip**
  (see Section 2.37.1)

2.4.1 Detailed descriptions

Public members

- **AbstractTracker (const Beamline&,const PartData&,bool,bool)**
  Constructor.

  The beam line to be tracked is **bl**. The particle reference data are taken from **data**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper..
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam.
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator.

- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component.
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector.
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic.
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift.
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson.
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker.
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor.
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole.
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch.
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend.
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity.
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole.
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend.
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator.
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum.
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid.
- **virtual ~AbstractTracker ()**

Protected members

- **const PartData itsReference**
  The reference information.

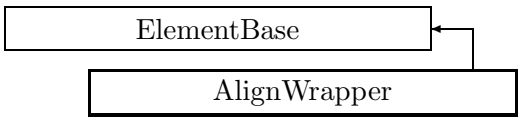

Figure 2.4: Inheritance for class AlignWrapper

2.5 class AlignWrapper

Define the position of a misaligned element.
An AlignWrapper is used to store misalignment errors or deliberate misalignments. It acts as a wrapper around a component or a complete beam line. Rotations and translations are defined about the design local frame, which in turn is specified by the position of the element on the design geometry. An AlignWrapper is non-sharable by default.

Type:	Instantiable
Superclasses:	public ElementBase
Include file:	./AbsBeamline/AlignWrapper.hh

Synopsis (including inherited members)

Public members

- **virtual void accept (BeamlineVisitor&)const**
  Apply BeamlineVisitor. (see Section 2.5.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual AlignWrapper* clone ()const**
  Return clone. (see Section 2.5.1)
- **virtual ElementBase* copyStructure ()**
  Make structural copy. (see Section 2.5.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual ElementBase* getElement ()const**
  Return the contained element. (see Section 2.5.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)

- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual Euclid3D getEntranceTransform ()const**
  Get entrance patch. (see Section 2.5.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual Euclid3D getExitTransform ()const**
  Get exit patch. (see Section 2.5.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.5.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.5.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.5.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)

- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Disallow misalignment of an already misaligned object. (see Section 2.5.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.5.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.5.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **Euclid3D& offset ()const**
  Return the offset. (see Section 2.5.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove AlignWrapper. (see Section 2.5.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove AlignWrapper. (see Section 2.5.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.5.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.5.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.5.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.5.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **void setElement (ElementBase*)**
  Replace the contained element. (see Section 2.5.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)

- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)

2.5.1 Detailed descriptions

Public members

- **virtual void accept (BeamlineVisitor&)const**
  Apply BeamlineVisitor.
- **virtual AlignWrapper* clone ()const**
  Return clone.

  Return an identical deep copy of the wrapper and its contents.
- **virtual ElementBase* copyStructure ()**
  Make structural copy.
- **virtual ElementBase* getElement ()const**
  Return the contained element.
- **virtual Euclid3D getEntranceTransform ()const**
  Get entrance patch.

  Returns the entrance patch (transformation) which is used to transform the global geometry to the local geometry at entrance of the misaligned element.
- **virtual Euclid3D getExitTransform ()const**
  Get exit patch.

  Returns the exit patch (transformation) which is used to transform the local geometry to the global geometry at exit of the misaligned element.
- **virtual Geometry& getGeometry ()**
  Get geometry.

  Return the element geometry. Version for non-constant object.
- **virtual const Geometry& getGeometry ()const**
  Get geometry.

  Return the element geometry. Version for constant object.
- **virtual const string& getType ()const**
  Get element type string.

  Returns the type string for the enclosed item.
- **virtual ElementBase* makeAlignWrapper ()**
  Disallow misalignment of an already misaligned object.

  This method returns **this**, since "this" is already an AlignWrapper.

- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors.

  Wrap the contained element in a field wrapper, unless such a wrapper already exists.

- **virtual void makeSharable ()**
  Set sharable flag.

  The whole structure depending on **this** is marked as sharable. After this call a **copyStructure()** call reuses the element.

- **Euclid3D& offset ()const**
  Return the offset.

  This method can be used to get or set the offset. The offset is declared as mutable, so as to allow changing it in a constant structure.

- **virtual ElementBase* removeAlignWrapper ()**
  Remove AlignWrapper.

  Return the element or field wrapper contained in "this". Version for non-const object.

- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove AlignWrapper.

  Return the element or field wrapper contained in "this". Version for const object.

- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper.

  Remove any field wrapper on the contained object.

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper.

  Remove the field wrapper for constant object.

- **virtual ElementBase* removeWrappers ()**
  Return the design element.

  This method removes all wrappers on the contained object. Version for non-const object.

- **virtual const ElementBase* removeWrappers ()const**
  Return the design element.

  Version for const object.

- **void setElement (ElementBase*)**
  Replace the contained element.

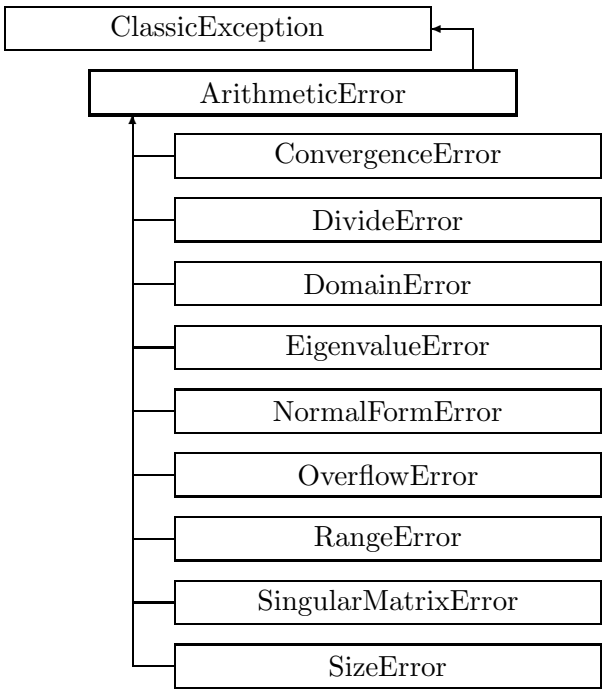

Figure 2.5: Inheritance for class ArithmeticError

2.6 class ArithmeticError

The abstract base class for all CLASSIC arithmetic exceptions.
An object derived from this class is thrown when an arithmetic error occurs.

Type:	Instantiable
Superclasses:	public ClassicException
Include file:	./Utilities/ArithmeticError.hh

Synopsis (including inherited members)

Protected members

- **ArithmeticError (const string&,const string&)**
  The usual constructor. (see Section 2.6.1)
- **ArithmeticError (const ArithmeticError&)**
  (see Section 2.6.1)
- **const string message**
  (see Section 2.21.1)
- **const string method**
  (see Section 2.21.1)
- **virtual ~ArithmeticError ()**
  (see Section 2.6.1)

2.6.1 Detailed descriptions

Protected members

- **ArithmeticError (const string&,const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  msg the message string identifying the exception

- **ArithmeticError (const ArithmeticError&)**

- **virtual ~ArithmeticError ()**

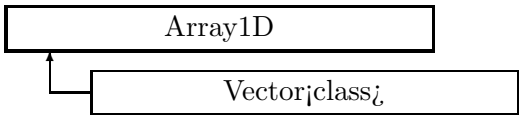

Figure 2.6: Inheritance for class Array1D

2.7 template class Array1D <class>

One-dimensional array.
A templated representation for one-dimensional arrays. This class implements storage management and component access, but contains no arithmetic operations.

Type:	Instantiable
Include file:	./Algebra/Array1D.hh

Synopsis (including inherited members)

Public members

- **Array1D ()**
  Default constructor. (see Section 2.7.1)
- **Array1D (int)**
  Constructor. (see Section 2.7.1)
- **Array1D (int,const T&)**
  Constructor. (see Section 2.7.1)
- **Array1D (const Array1D<T>&)**
  (see Section 2.7.1)
- **iterator begin ()**
  Get beginning of data. (see Section 2.7.1)
- **const_iterator begin ()const**
  Get beginning of data. (see Section 2.7.1)
- **typedef const T* const_iterator**
  The iterator type for constant array. (see Section 2.7.1)
- **iterator end ()**
  Get end of data. (see Section 2.7.1)
- **const_iterator end ()const**
  Get end of data. (see Section 2.7.1)
- **typedef T* iterator**
  The iterator type for the array. (see Section 2.7.1)
- **T& operator() (int)**
  Get reference to element. (see Section 2.7.1)

- **const T& operator() (int)const**
  Get value of element. (see Section 2.7.1)
- **Array1D<T>& operator= (const Array1D<T>&)**
  (see Section 2.7.1)
- **T& operator[] (int)**
  Get reference to element. (see Section 2.7.1)
- **const T& operator[] (int)const**
  Get value of element. (see Section 2.7.1)
- **void resize (int)**
  Change array size. (see Section 2.7.1)
- **int size ()const**
  Get array size. (see Section 2.7.1)
- **typedef T value_type**
  The value type of this array. (see Section 2.7.1)
- **~Array1D ()**
  (see Section 2.7.1)

Protected members

- **T* data**
  (see Section 2.7.1)
- **int len**
  (see Section 2.7.1)

2.7.1 Detailed descriptions

Public members

- **Array1D ()**
  Default constructor.

  Constructs an array of zero length.
- **Array1D (int)**
  Constructor.

  Reserves space for **n** elements and leaves them undefined.
- **Array1D (int,const T&)**
  Constructor.

  Reserves space for **n** elements and store **n** copies of **t**.
- **Array1D (const Array1D<T>&)**

- **iterator begin ()**
  Get beginning of data.

  Return pointer to beginning of array. Version for non-constant objects.

- **const_iterator begin ()const**
  Get beginning of data.

  Return pointer to beginning of array. Version for constant objects.

- **typedef const T* const_iterator**
  The iterator type for constant array.

- **iterator end ()**
  Get end of data.

  Return pointer past end of array. Version for non-constant objects.

- **const_iterator end ()const**
  Get end of data.

  Return pointer past end of array. Version for constant objects.

- **typedef T* iterator**
  The iterator type for the array.

- **T& operator() (int)**
  Get reference to element.

  Return a reference to the element in position **n**. Throw RangeError, if **n** is out of range.

- **const T& operator() (int)const**
  Get value of element.

  Return the value of the element in position **n**. Throw RangeError, if **n** is out of range.

- **Array1D<T>& operator= (const Array1D<T>&)**

- **T& operator[] (int)**
  Get reference to element.

  Return a reference to the element in position **n**. Result is undefined, if **n** is out of range.

- **const T& operator[] (int)const**
  Get value of element.

  Return the value of the element in position **n**. Result is undefined, if **n** is out of range.

- **void resize (int)**
  Change array size.

  Elements added are left undefined.

- **int size ()const**
  Get array size.

- **typedef T value_type**
  The value type of this array.

- **~Array1D ()**

Protected members

- **T* data**

- **int len**

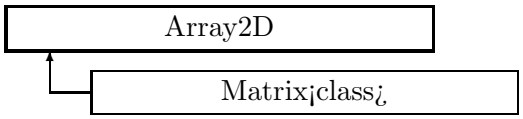

Figure 2.7: Inheritance for class Array2D

2.8 template class Array2D <class>

Two-dimensional array.
A templated representation for 2-dimensional arrays. This class implements storage management and component access, as well as access to rows and columns, but contains no arithmetic operations.

Type:	Instantiable
Include file:	./Algebra/Array2D.hh

Synopsis (including inherited members)

Public members

- **Array2D ()**
  Default constructor. (see Section 2.8.1)
- **Array2D (int,int)**
  Constructor. (see Section 2.8.1)
- **Array2D (int,int,const T&)**
  Constructor. (see Section 2.8.1)
- **Array2D (const Array2D&)**
  (see Section 2.8.1)
- **iterator begin ()**
  Get pointer to beginning of data. (see Section 2.8.1)
- **const_iterator begin ()const**
  Get pointer to beginning of data. (see Section 2.8.1)
- **col_iterator col_begin (int)**
  Get column iterator. (see Section 2.8.1)
- **const_col_iterator col_begin (int)const**
  Get column iterator. (see Section 2.8.1)
- **col_iterator col_end (int)**
  Get column iterator. (see Section 2.8.1)
- **const_col_iterator col_end (int)const**
  Get column iterator. (see Section 2.8.1)
- **typedef SliceIterator<T> col_iterator**
  The iterator type for access by columns. (see Section 2.8.1)

- **typedef ConstSliceIterator<T> const_col_iterator**
  The iterator type for access by columns for a constant array. (see Section 2.8.1)
- **typedef const T* const_iterator**
  The iterator type for sequential access to all elements (see Section 2.8.1)
- **typedef const T* const_row_iterator**
  The iterator type for access by rows for a constant array. (see Section 2.8.1)
- **iterator end ()**
  Get pointer past end of data. (see Section 2.8.1)
- **const_iterator end ()const**
  Get pointer past end of data. (see Section 2.8.1)
- **void getColumn (Array1D<T>&,int)const**
  Fetch column. (see Section 2.8.1)
- **void getRow (Array1D<T>&,int)const**
  Fetch row. (see Section 2.8.1)
- **typedef T* iterator**
  The iterator type for sequential access to all elements. (see Section 2.8.1)
- **int ncols ()const**
  Get number of columns. (see Section 2.8.1)
- **int nrows ()const**
  Get number of rows. (see Section 2.8.1)
- **T& operator() (int,int)**
  Get element reference. (see Section 2.8.1)
- **const T& operator() (int,int)const**
  Get element value. (see Section 2.8.1)
- **Array2D<T>& operator= (const Array2D<T>&)**
  (see Section 2.8.1)
- **row_iterator operator[] (int)**
  Get row iterator. (see Section 2.8.1)
- **const_row_iterator operator[] (int)const**
  Get row iterator. (see Section 2.8.1)
- **void putColumn (const Array1D<T>&,int)**
  Store column. (see Section 2.8.1)
- **void putRow (const Array1D<T>&,int)**
  Store row. (see Section 2.8.1)
- **row_iterator row_begin (int)**
  Get row iterator. (see Section 2.8.1)

- **const_row_iterator row_begin (int)const**
  Get row iterator. (see Section 2.8.1)
- **row_iterator row_end (int)**
  Get row iterator. (see Section 2.8.1)
- **const_row_iterator row_end (int)const**
  Get row iterator. (see Section 2.8.1)
- **typedef T* row_iterator**
  The iterator type for access by rows. (see Section 2.8.1)
- **int size ()const**
  Get total size (rows times columns). (see Section 2.8.1)
- **void swapColumns (int,int)**
  Exchange columns. (see Section 2.8.1)
- **void swapRows (int,int)**
  Exchange rows. (see Section 2.8.1)
- **typedef T value_type**
  The value type of the array. (see Section 2.8.1)
- **~Array2D ()**
  (see Section 2.8.1)

Protected members

- **int cols**
  (see Section 2.8.1)
- **T* data**
  (see Section 2.8.1)
- **int len**
  (see Section 2.8.1)
- **int rows**
  (see Section 2.8.1)

2.8.1 Detailed descriptions

Public members

- **Array2D ()**
  Default constructor.

  Constructs array of zero rows and zero columns.
- **Array2D (int,int)**
  Constructor.

  Reserves **rows** times **cols** elements and leaves them undefined.

- **Array2D (int,int,const T&)**
  Constructor.

  Reserves **rows** x **cols** elements and sets them to **t**.

- **Array2D (const Array2D&)**

- **iterator begin ()**
  Get pointer to beginning of data.

  Treat the array as one-dimensional. Version for non-constant objects.

- **const_iterator begin ()const**
  Get pointer to beginning of data.

  Treat the array as one-dimensional. Version for constant objects.

- **col_iterator col_begin (int)**
  Get column iterator.

  Return pointer to beginning of column c. Throw RangeError if c is out of range. Version for non-constant objects.

- **const_col_iterator col_begin (int)const**
  Get column iterator.

  Return pointer to beginning of column c. Throw RangeError if c is out of range. Version for constant objects.

- **col_iterator col_end (int)**
  Get column iterator.

  Return pointer past end of column c. Throw RangeError if c is out of range. Version for non-constant objects.

- **const_col_iterator col_end (int)const**
  Get column iterator.

  Return pointer past end of column c. Throw RangeError if c is out of range. Version for constant objects.

- **typedef SliceIterator<T> col_iterator**
  The iterator type for access by columns.

- **typedef ConstSliceIterator<T> const_col_iterator**
  The iterator type for access by columns for a constant array.

- **typedef const T* const_iterator**
  The iterator type for sequential access to all elements

  of a constant array.

- **typedef const T* const_row_iterator**
  The iterator type for access by rows for a constant array.

- **iterator end ()**
  Get pointer past end of data.

  Treat the array as one-dimensional. Version for non-constant objects.

- **const_iterator end ()const**
  Get pointer past end of data.

  Treat the array as one-dimensional. Version for constant objects.

- **void getColumn (Array1D<T>&,int)const**
  Fetch column.

  Store the column **c** into the array **toArray**. Throw RangeError if c is out of range.

- **void getRow (Array1D<T>&,int)const**
  Fetch row.

  Store the row **r** into the array **toArray**. Throw RangeError if r is out of range.

- **typedef T* iterator**
  The iterator type for sequential access to all elements.

- **int ncols ()const**
  Get number of columns.

- **int nrows ()const**
  Get number of rows.

- **T& operator() (int,int)**
  Get element reference.

  Return a reference to element in row **r** and column **c**.

- **const T& operator() (int,int)const**
  Get element value.

  Return the value of element in row **r** and column **c**.

- **Array2D<T>& operator= (const Array2D<T>&)**

- **row_iterator operator[] (int)**
  Get row iterator.

  Return pointer to beginning of row r. Result is undefined, if r is out of range. Version for non-constant objects.

- **const_row_iterator operator[] (int)const**
  Get row iterator.

  Return pointer to beginning of row r. Result is undefined, if r is out of range. Version for constant objects.

- **void putColumn (const Array1D<T>&,int)**
  Store column.

  Fill the column **c** from the array **fromArray**. Throw RangeError if c is out of range.

- **void putRow (const Array1D<T>&,int)**
  Store row.

  Fill the row **r** from the array **fromArray**. Throw RangeError if r is out of range.

- **row_iterator row_begin (int)**
  Get row iterator.

  Return pointer to beginning of row r. Throw RangeError, if row is out of range. Version for non-constant objects.

- **const_row_iterator row_begin (int)const**
  Get row iterator.

  Return pointer to beginning of constant row r. Throws RangeError, if row is out of range. Version for constant objects.

- **row_iterator row_end (int)**
  Get row iterator.

  Return pointer past end of row r. Throw RangeError, if row is out of range. Version for non-constant objects.

- **const_row_iterator row_end (int)const**
  Get row iterator.

  Return pointer past end of constant row r. Throws RangeError, if row is out of range. Version for constant objects.

- **typedef T* row_iterator**
  The iterator type for access by rows.

- **int size ()const**
  Get total size (rows times columns).

- **void swapColumns (int,int)**
  Exchange columns.

  Exchange the columns **c1** and **c2**. Throw RangeError, if either index is out of range.

- **void swapRows (int,int)**
  Exchange rows.

  Exchange the rows **r1** and **r2**. Throw RangeError, if either index is out of range.

- **typedef T value_type**
  The value type of the array.

- **~Array2D ()**

Protected members

- **int cols**

- **T* data**
- **int len**
- **int rows**

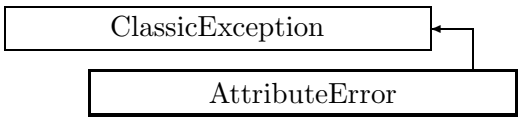

Figure 2.8: Inheritance for class AttributeError

2.9 class AttributeError

Exception for bad object attribute.
This exception is thrown when bad object attributes are detected.

Type:	Instantiable
Superclasses:	public ClassicException
Include file:	./Utilities/AttributeError.hh

Synopsis (including inherited members)

Public members

- **AttributeError (const string&,const string&)**
  The usual constructor. (see Section 2.9.1)
- **AttributeError (const AttributeError&)**
  (see Section 2.9.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~AttributeError ()**
  (see Section 2.9.1)

2.9.1 Detailed descriptions

Public members

- **AttributeError (const string&,const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  msg the message string identifying the exception
- **AttributeError (const AttributeError&)**
- **virtual ~AttributeError ()**

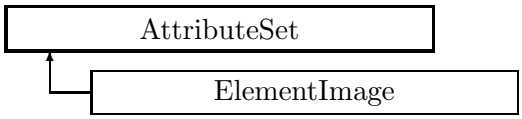

Figure 2.9: Inheritance for class AttributeSet

2.10 class AttributeSet

Map of string versus double value.
Class AttributeSet implements a map of name (string) versus value (double) for user-defined attributes. This map is intended for algorithms that require specific, but not predefined data in the accelerator model for their working.

Type:	Instantiable
Include file:	./AbsBeamline/AttributeSet.hh

Synopsis (including inherited members)

Public members

- **AttributeSet ()**
  Default constructor. (see Section 2.10.1)
- **AttributeSet (const AttributeSet&)**
  (see Section 2.10.1)
- **typedef std::map<string,double,std::less<string> > NameMap**
  A map of name versus value. (see Section 2.10.1)
- **const_iterator begin ()const**
  Iterator accessing first member. (see Section 2.10.1)
- **typedef NameMap::const_iterator const_iterator**
  An iterator for a map of name versus value. (see Section 2.10.1)
- **const_iterator end ()const**
  Iterator marking the end of the list. (see Section 2.10.1)
- **double getAttribute (const string&)const**
  Get attribute value. (see Section 2.10.1)
- **Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.10.1)
- **const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.10.1)
- **bool hasAttribute (const string&)const**
  Test for presence of an attribute. (see Section 2.10.1)
- **const AttributeSet& operator= (const AttributeSet&)**
  (see Section 2.10.1)

- **void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.10.1)
- **void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.10.1)
- **virtual ~AttributeSet ()**
  (see Section 2.10.1)

Protected members

- **NameMap itsMap**
  The attribute map. (see Section 2.10.1)

2.10.1 Detailed descriptions

Public members

- **AttributeSet ()**
  Default constructor.

  Constructs an empty map.
- **AttributeSet (const AttributeSet&)**

- **typedef std::map<string,double,std::less<string> > NameMap**
  A map of name versus value.
- **const_iterator begin ()const**
  Iterator accessing first member.
- **typedef NameMap::const_iterator const_iterator**
  An iterator for a map of name versus value.
- **const_iterator end ()const**
  Iterator marking the end of the list.
- **double getAttribute (const string&)const**
  Get attribute value.

  If the attribute does not exist, return zero.
- **Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.
- **const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel.

  This method constructs a Channel permitting read-only access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.

- **bool hasAttribute (const string&)const**
  Test for presence of an attribute.

  If the attribute exists, return true, otherwise false.

- **const AttributeSet& operator= (const AttributeSet&)**

- **void removeAttribute (const string&)**
  Remove an existing attribute.

  If the key **aKey** exists, this method removes it.

- **void setAttribute (const string&,double)**
  Set value of an attribute.

- **virtual ~AttributeSet ()**

Protected members

- **NameMap itsMap**
  The attribute map.

Figure 2.10: Inheritance for class BDipoleField

2.11 class BDipoleField

The field of a magnetic dipole.
A static magnetic dipole field in the (x,y)-plane.

Type:	Instantiable
Superclasses:	public ConstBField
Include file:	./Fields/BDipoleField.hh

Synopsis (including inherited members)

Public members

- **BDipoleField ()**
  Default constructor. (see Section 2.11.1)
- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.11.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.11.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)
- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)
- **BDipoleField& addField (const BDipoleField&)**
  Add to field. (see Section 2.11.1)

- **virtual double getBx ()const**
  Get horizontal component. (see Section 2.11.1)
- **virtual double getBy ()const**
  Get vertical component. (see Section 2.11.1)
- **virtual double getBz ()const**
  Get component. (see Section 2.27.1)
- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)
- **virtual void scale (double)**
  Scale the field. (see Section 2.11.1)
- **virtual void setBx (double)**
  Set horizontal component. (see Section 2.11.1)
- **virtual void setBy (double)**
  Set vertical component. (see Section 2.11.1)
- **virtual void setBz (double)**
  Set component. (see Section 2.27.1)
- **BDipoleField& subtractField (const BDipoleField&)**
  Subtract from field. (see Section 2.11.1)
- **virtual ˜BDipoleField ()**
  (see Section 2.11.1)

2.11.1 Detailed descriptions

Public members

- **BDipoleField ()**
  Default constructor.

  Constructs a null field.
- **virtual BVector Bfield (const Point3D&)const**
  Get field.

  Return the time-independent part of the magnetic field in point **P**. This override forces implementation in derived classes.
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field.

  Return the magnetic field at time **t** in point **P**. This override forces implementation in derived classes.
- **BDipoleField& addField (const BDipoleField&)**
  Add to field.

  Add **field** to the old value; return new value.

- **virtual double getBx ()const**
  Get horizontal component.

  Return the horizontal component of the field in Teslas.

- **virtual double getBy ()const**
  Get vertical component.

  Return the vertical component of the field in Teslas.

- **virtual void scale (double)**
  Scale the field.

  Multiply the field by **scalar**.

- **virtual void setBx (double)**
  Set horizontal component.

  Assign the horizontal component of the field in Teslas.

- **virtual void setBy (double)**
  Set vertical component.

  Assign the vertical component of the field in Teslas.

- **BDipoleField& subtractField (const BDipoleField&)**
  Subtract from field.

  Subtract **field** from the old value; return new value.

- **virtual ˜BDipoleField ()**

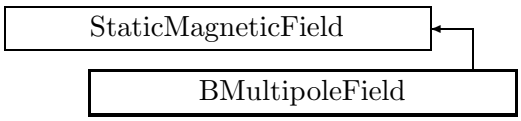

Figure 2.11: Inheritance for class BMultipoleField

2.12 class BMultipoleField

The magnetic field of a multipole.
The static components are defined by the equation center By + i * Bx = sum (Bn + i * An) (x + i * y) ** (n - 1) /center where the index n runs from 1 (dipole) to N (2N-pole). The field component n is stored in the pairs[n-1], The array pairs has the dimension N.

Type:	Instantiable
Superclasses:	public StaticMagneticField
Include file:	./Fields/BMultipoleField.hh

Synopsis (including inherited members)

Public members

- **BMultipoleField ()**
  Default constructor. (see Section 2.12.1)
- **BMultipoleField (const BMultipoleField&)**
  (see Section 2.12.1)
- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.12.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.12.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)
- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)

- **BMultipoleField& addField (const BMultipoleField&)**
  Add to field. (see Section 2.12.1)
- **double getNormalComponent (int)const**
  Get component. (see Section 2.12.1)
- **double getSkewComponent (int)const**
  Get component. (see Section 2.12.1)
- **double normal (int)const**
  Get component. (see Section 2.12.1)
- **double& normal (int)**
  Get component. (see Section 2.12.1)
- **BMultipoleField& operator= (const BMultipoleField&)**
  (see Section 2.12.1)
- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)
- **int order ()const**
  Return order. (see Section 2.12.1)
- **void scale (double)**
  Scale the field. (see Section 2.12.1)
- **void setNormalComponent (int,double)**
  Set component. (see Section 2.12.1)
- **void setSkewComponent (int,double)**
  Set component. (see Section 2.12.1)
- **double skew (int)const**
  Get component. (see Section 2.12.1)
- **double& skew (int)**
  Get component. (see Section 2.12.1)
- **BMultipoleField& subtractField (const BMultipoleField&)**
  Subtract from field. (see Section 2.12.1)
- **virtual ˜BMultipoleField ()**
  (see Section 2.12.1)

2.12.1 Detailed descriptions

Public members

- **BMultipoleField ()**
  Default constructor.

  Constructs a null field.

- **BMultipoleField (const BMultipoleField&)**

- **virtual BVector Bfield (const Point3D&)const**
  Get field.

  Return the time-independent part of the magnetic field in point **P**. This override forces implementation in derived classes.

- **virtual BVector Bfield (const Point3D&,double)const**
  Get field.

  Return the magnetic field at time **t** in point **P**. This override forces implementation in derived classes.

- **BMultipoleField& addField (const BMultipoleField&)**
  Add to field.

  Add **field** to the old field; return new field.

- **double getNormalComponent (int)const**
  Get component.

  Return the value of the n'th normal multipole component in T/m**(n-1). If **n** is larger than N, the return value is zero.

- **double getSkewComponent (int)const**
  Get component.

  Return the value of the n'th skew multipole component in T/m**(n-1). If **n** is larger than N, the return value is zero.

- **double normal (int)const**
  Get component.

  Return the value of the n'th normal multipole component in T/m**(n-1). Fast version: **n** is not checked. the result is undefined if it is larger than N.

- **double& normal (int)**
  Get component.

  Return a reference to the n'th normal multipole component in T/m**(n-1). Fast version: **n** is not checked. the result is undefined if it is larger than N.

- **BMultipoleField& operator= (const BMultipoleField&)**

- **int order ()const**
  Return order.

- **void scale (double)**
  Scale the field.

  Multiply the field by **scalar**.

- **void setNormalComponent (int,double)**
  Set component.

  Assign the value of the n'th normal multipole component in T/m**(n-1). If **n** is larger than N, the new field component is inserted.

- **void setSkewComponent (int,double)**
  Set component.

  Assign the value of the n'th skew multipole component in T/m**(n-1). If **n** is larger than N, the new field component is inserted.

- **double skew (int)const**
  Get component.

  Return the value of the n'th skew multipole component in T/m**(n-1). Fast version: **n** is not checked. the result is undefined if it is larger than N.

- **double& skew (int)**
  Get component.

  Return a reference to the n'th skew multipole component in T/m**(n-1). Fast version: **n** is not checked. the result is undefined if it is larger than N.

- **BMultipoleField& subtractField (const BMultipoleField&)**
  Subtract from field.

  Subtract **field** from the old field; return new field.

- **virtual ~BMultipoleField ()**

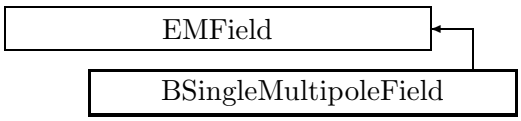

Figure 2.12: Inheritance for class BSingleMultipoleField

2.13 template class BSingleMultipoleField <int>

Representation for a single magnetic multipole field.
The order and the skew flag are encoded in a template parameter. center order > 0: normal multipole, order < 0: skew multipole. /center Thus the compiler may optimize by unrolling loops on the order. It should also omit some tests and eliminate unreachable code.

Type:	Instantiable
Superclasses:	public EMField
Include file:	./Fields/BSingleMultipoleField.hh

Synopsis (including inherited members)

Public members

- **BSingleMultipoleField ()**
  Default constructor. (see Section 2.13.1)
- **BSingleMultipoleField (const BSingleMultipoleField&)**
  (see Section 2.13.1)
- **virtual BVector Bfield (const Point3D&)const**
  Field at a given point. (see Section 2.13.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.13.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)
- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)

- **virtual double getComponent ()const**
  Return field coefficient. (see Section 2.13.1)
- **operator ()const**
  Conversion operator. (see Section 2.13.1)
- **BSingleMultipoleField& operator= (const BSingleMultipoleField&)**
  (see Section 2.13.1)
- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)
- **void scale (double)**
  Scale the field. (see Section 2.13.1)
- **virtual void setComponent (double)**
  Set field coefficient. (see Section 2.13.1)
- **int size ()const**
  Return order. (see Section 2.13.1)
- **virtual ~BSingleMultipoleField ()**
  (see Section 2.13.1)

2.13.1 Detailed descriptions

Public members

- **BSingleMultipoleField ()**
  Default constructor.

  Constructs a null field.
- **BSingleMultipoleField (const BSingleMultipoleField&)**
- **virtual BVector Bfield (const Point3D&)const**
  Field at a given point.

  Return the magnetic field at point **P**.
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field.

  Return the magnetic field at time **t** in point **P**. This default action returns the static part BField(P).
- **virtual double getComponent ()const**
  Return field coefficient.

  Return the single multipole coefficient in T/m**n.
- **operator ()const**
  Conversion operator.

  Return the field as a BMultipoleField.

- **BSingleMultipoleField& operator= (const BSingleMultipoleField&)**

- **void scale (double)**
  Scale the field.

  Multiply the field by **scalar**.

- **virtual void setComponent (double)**
  Set field coefficient.

  Assign the single multipole coefficient in T/m**n.

- **int size ()const**
  Return order.

- **virtual ~BSingleMultipoleField ()**

2.14 class BVector

A magnetic field vector.

Type:	Instantiable
Include file:	./Fields/EMField.hh

Synopsis (including inherited members)

Public members

- **BVector (double,double,double)**
  Constructor. (see Section 2.14.1)
- **double getBx ()const**
  Get component. (see Section 2.14.1)
- **double getBy ()const**
  Get component. (see Section 2.14.1)
- **double getBz ()const**
  Get component. (see Section 2.14.1)
- **BVector operator* (double)const**
  Scale. (see Section 2.14.1)

2.14.1 Detailed descriptions

Public members

- **BVector (double,double,double)**
  Constructor.

  Construct the field vector from its components (Bx,By,Bz) in T.
- **double getBx ()const**
  Get component.

  Return the x-component of the field in T.
- **double getBy ()const**
  Get component.

  Return the y-component of the field in T.
- **double getBz ()const**
  Get component.

  Return the z-component of the field in T.
- **BVector operator* (double)const**
  Scale.

  Multiply the field vector by **scalar**.

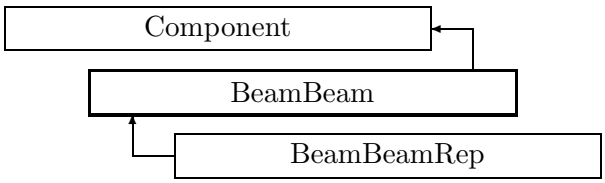

Figure 2.13: Inheritance for class BeamBeam

2.15 class BeamBeam

Abstract beam-beam interaction.
Class BeamBeam defines the abstract interface for beam-beam interactions.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/BeamBeam.hh

Synopsis (including inherited members)

Public members

- **BeamBeam (const string&)**
  Constructor with given name. (see Section 2.15.1)
- **BeamBeam ()**
  (see Section 2.15.1)
- **BeamBeam (const BeamBeam&)**
  (see Section 2.15.1)
- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to BeamBeam. (see Section 2.15.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getBunchCharge ()const**
  Get bunch charge. (see Section 2.15.1)
- **virtual const Vector3D& getBunchDisplacement ()const**
  Get displacement. (see Section 2.15.1)
- **virtual const Matrix3D& getBunchMoment ()const**
  Get moments. (see Section 2.15.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)

- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~BeamBeam ()**
  (see Section 2.15.1)

2.15.1 Detailed descriptions

Public members

- **BeamBeam (const string&)**
  Constructor with given name.
- **BeamBeam ()**
- **BeamBeam (const BeamBeam&)**

- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to BeamBeam.

- **virtual double getBunchCharge ()const**
  Get bunch charge.

  Return the number of particles times the particle charge in the opposite bunch. Units are proton charges.

- **virtual const Vector3D& getBunchDisplacement ()const**
  Get displacement.

  Return the displacement vector for position of opposite bunch. Units are metres.

- **virtual const Matrix3D& getBunchMoment ()const**
  Get moments.

  Return the moment matrix for the opposite bunch (the matrix of second momenta). Units are square metres.

- **virtual ~BeamBeam ()**

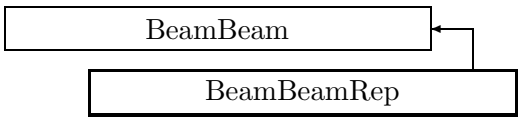

Figure 2.14: Inheritance for class BeamBeamRep

2.16 class BeamBeamRep

Representation of beam-beam interaction.
A concrete representation for a beam-beam interaction. The opposite bunch has a Gaussian distribution of the form
N(x,y,s) = c * exp(- transpose(z - delta) sigma**(-1) (z - delta)),
with the definitions

z position vector in space,

delta centroid of opposite bunch in global reference system,

sigma beam matrix in the TRANSPORT sense,

c a normalising factor, such that the total charge is Q,

sigma(i,i) standard deviation sigma(i) in direction i,

r(i,j) = sigma(i,j)/sqrt(sigma(i,i)*sigma(i,j)), correlations between phase space coordinates i and j.

Type:	Instantiable
Superclasses:	public BeamBeam
Include file:	./BeamlineCore/BeamBeamRep.hh

Synopsis (including inherited members)

Public members

- **BeamBeamRep (const string&)**
  Constructor with given name. (see Section 2.16.1)
- **BeamBeamRep ()**
  (see Section 2.16.1)
- **BeamBeamRep (const BeamBeamRep&)**
  (see Section 2.16.1)
- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)

- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to BeamBeam. (see Section 2.15.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.16.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getBunchCharge ()const**
  Get bunch charge. (see Section 2.16.1)
- **virtual const Vector3D& getBunchDisplacement ()const**
  Get displacement. (see Section 2.16.1)
- **virtual const Matrix3D& getBunchMoment ()const**
  Get moments. (see Section 2.16.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.16.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)

- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual NullField& getField ()**
  Get field. (see Section 2.16.1)
- **virtual const NullField& getField ()const**
  Get field. (see Section 2.16.1)
- **virtual NullGeometry& getGeometry ()**
  Get geometry. (see Section 2.16.1)
- **virtual const NullGeometry& getGeometry ()const**
  Get geometry. (see Section 2.16.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.16.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Return type name string (see Section 2.16.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)

- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **void setBunchCharge (double)**
  Set the bunch charge. (see Section 2.16.1)
- **void setBunchDisplacement (const Vector3D&)**
  Set displacement. (see Section 2.16.1)
- **void setBunchMoment (const Matrix3D&)**
  Set moments. (see Section 2.16.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)

- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ˜BeamBeamRep ()**
  (see Section 2.16.1)

2.16.1 Detailed descriptions

Public members

- **BeamBeamRep (const string&)**
  Constructor with given name.
- **BeamBeamRep ()**
- **BeamBeamRep (const BeamBeamRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual double getBunchCharge ()const**
  Get bunch charge.

  Return the number of particles times the particle charge in the opposite bunch. Units are proton charges.
- **virtual const Vector3D& getBunchDisplacement ()const**
  Get displacement.

  Return the displacement vector for position of opposite bunch. Units are metres.
- **virtual const Matrix3D& getBunchMoment ()const**
  Get moments.

  Return the moment matrix for the opposite bunch (the matrix of second momenta). Units are square metres.
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute**aKey** and returns it. If the attribute does not exist, it returns NULL.
- **virtual NullField& getField ()**
  Get field.

  Version for non-constant object.

- **virtual const NullField& getField ()const**
  Get field.

  Version for constant object.

- **virtual NullGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const NullGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual const string& getType ()const**
  Return type name string

- **void setBunchCharge (double)**
  Set the bunch charge.

  Units are proton charges.

- **void setBunchDisplacement (const Vector3D&)**
  Set displacement.

  Assign the displacement vector for the opposite bunch. Units are metres.

- **void setBunchMoment (const Matrix3D&)**
  Set moments.

  Assign the moment matrix for the opposite bunch (the matrix of second momenta). Units are square metres.

- **virtual ~BeamBeamRep ()**

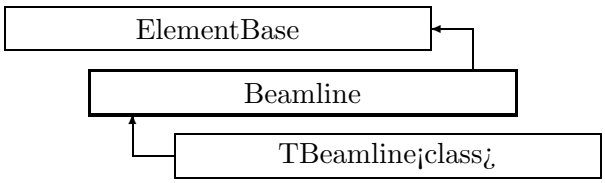

Figure 2.15: Inheritance for class Beamline

2.17 class Beamline

An abstract sequence of beam line components.
A beam line is built as a list of objects derived from ElmPtr. Each ElmPtr ("element pointer") points to a ElementBase, and may contain additional data describing the position, like lattice functions etc.

Type:	abstract
Superclasses:	public ElementBase
Include file:	./Beamlines/Beamline.hh

Synopsis (including inherited members)

Public members

- **Beamline (const string&)**
  Constructor with given name. (see Section 2.17.1)
- **Beamline ()**
  (see Section 2.17.1)
- **Beamline (const Beamline&)**
  (see Section 2.17.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor. (see Section 2.52.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)

- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)

- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual void iterate (BeamlineVisitor&,bool)const**
  Apply visitor to all elements of the line. (see Section 2.17.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)

- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Beamline ()**
  (see Section 2.17.1)

2.17.1 Detailed descriptions

Public members

- **Beamline (const string&)**
  Constructor with given name.
- **Beamline ()**
- **Beamline (const Beamline&)**
- **virtual void iterate (BeamlineVisitor&,bool)const**
  Apply visitor to all elements of the line.

  If the parameter **reverse** is true, theline is traversed in reverse direction. If any error occurs, this method may throw an exception.
- **virtual ~Beamline ()**

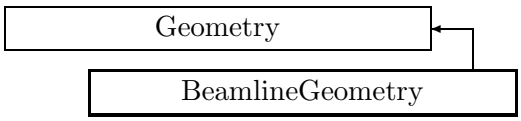

Figure 2.16: Inheritance for class BeamlineGeometry

2.18 class BeamlineGeometry

Implements the composite geometry of a beam line.

Type:	Instantiable
Superclasses:	public Geometry
Include file:	./Beamlines/BeamlineGeometry.hh

Synopsis (including inherited members)

Public members

- **BeamlineGeometry (const Beamline&)**
  Constructor. (see Section 2.18.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.18.1)
- **virtual double getElementLength ()const**
  Get element length. (see Section 2.18.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.18.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.18.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.80.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.18.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.18.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.80.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.18.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.18.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.18.1)

- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.18.1)
- **const Geometry& operator= (const Geometry&)**
  (see Section 2.80.1)
- **virtual void setElementLength (double)**
  Set geometry length. (see Section 2.80.1)
- **virtual ˜BeamlineGeometry ()**
  (see Section 2.18.1)

2.18.1 Detailed descriptions

Public members

- **BeamlineGeometry (const Beamline&)**
  Constructor.

  The geometry is linked to th beamline **line**.
- **virtual double getArcLength ()const**
  Get arc length.

  Return the length of the geometry, measured along the design orbit.
- **virtual double getElementLength ()const**
  Get element length.

  Return the length of the geometry, measured along the design polygone.
- **virtual double getEntrance ()const**
  Get entrance position.

  Return the arc length from the origin to the entrance of the geometry (non-positive0)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getEntrance()). Return the transform of the local coordinate system from the origin to the entrance of the element.
- **virtual double getExit ()const**
  Get exit position.

  Return the arc length from the origin to the exit of the geometry (non-negative).
- **virtual Euclid3D getExitFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getExit()). Return the transform of the local coordinate system from the origin to the exit of the element.
- **virtual double getOrigin ()const**
  Get origin position.

  Return the arc length from the entrance to the origin of the geometry (non-negative).

- **virtual Euclid3D getTotalTransform ()const**
  Get transform.

  Equivalent to getTransform(getEntrance(), getExit()). Return the transform of the local coordinate system from the entrance to the exit of the element.

- **virtual Euclid3D getTransform (double,double)const**
  Get transform.

  Return the transform of the local coordinate system from the position **fromS** to the position **toS**.

- **virtual Euclid3D getTransform (double)const**
  Get transform.

  Equivalent to getTransform(0.0, s). Return the transform of the local coordinate system from the origin and **s**.

- **virtual ~BeamlineGeometry ()**

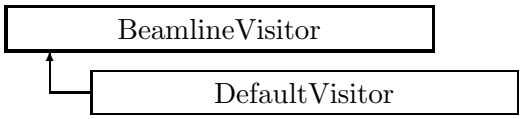

Figure 2.17: Inheritance for class BeamlineVisitor

2.19 class BeamlineVisitor

Abstract algorithm.
The abstract class BeamlineVisitor is the base class for all visitors (algorithms) that can iterator over a beam line representation. A BeamlineVisitor applies itself to the representation via the "Visitor" pattern, see
E. Gamma, R. Helm, R. Johnson, and J. Vlissides,
Design Patterns, Elements of Reusable Object-Oriented Software.
By using only pure abstract classes as an interface between the BeamlineVisitor and the beam line representation, we decouple the former from the implementation details of the latter.
The interface is defined in such a way that a visitor cannot modify the structure of a beam line, but it can assign special data like misalignments or integrators without problems.

Type:	abstract
Include file:	./AbsBeamline/BeamlineVisitor.hh

Synopsis (including inherited members)

Public members

- **BeamlineVisitor ()**
  (see Section 2.19.1)
- **virtual void execute ()**
  Execute the algorithm on the attached beam line. (see Section 2.19.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper. (see Section 2.19.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam interaction. (see Section 2.19.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a Beamline. (see Section 2.19.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.19.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component (catch all). (see Section 2.19.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a closed orbit corrector. (see Section 2.19.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper. (see Section 2.19.1)

- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic. (see Section 2.19.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift space. (see Section 2.19.1)
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.19.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.19.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson septum magnet. (see Section 2.19.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.19.1)
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker. (see Section 2.19.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a beam position monitor. (see Section 2.19.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole. (see Section 2.19.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper. (see Section 2.19.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.19.1)
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend. (see Section 2.19.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper. (see Section 2.19.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity. (see Section 2.19.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole. (see Section 2.19.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend. (see Section 2.19.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper. (see Section 2.19.1)
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to an electrostatic separator. (see Section 2.19.1)

- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum magnet. (see Section 2.19.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid. (see Section 2.19.1)
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.19.1)
- **virtual ~BeamlineVisitor ()**
  (see Section 2.19.1)

2.19.1 Detailed descriptions

Public members

- **BeamlineVisitor ()**

- **virtual void execute ()**
  Execute the algorithm on the attached beam line.
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper.
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam interaction.
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a Beamline.
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator.
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component (catch all).
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a closed orbit corrector.
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic.
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift space.
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr.
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator.

- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson septum magnet.
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping.
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker.
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a beam position monitor.
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole.
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch.
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend.
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity.
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole.
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend.
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to an electrostatic separator.
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum magnet.
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid.
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking.
- **virtual ~BeamlineVisitor ()**

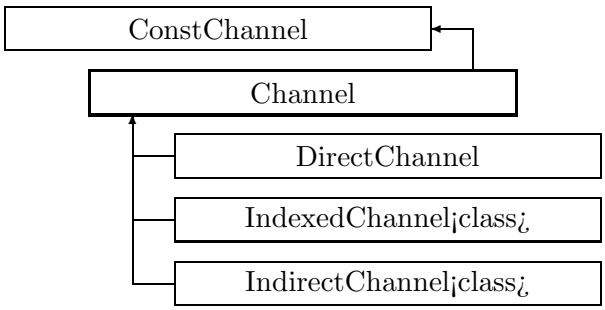

Figure 2.18: Inheritance for class Channel

2.20 class Channel

Abstract interface for read/write access to variable.
Class Channel allows access to an arbitrary **double** value.

Type:	abstract
Superclasses:	public ConstChannel
Include file:	./Channels/Channel.hh

Synopsis (including inherited members)

Public members

- **Channel ()**
  (see Section 2.20.1)
- **virtual Channel* clone ()const**
  Duplicate the channel. (see Section 2.20.1)
- **virtual bool get (double&)const**
  Read channel. (see Section 2.29.1)
- **virtual bool isSettable ()const**
  Test if settable. (see Section 2.20.1)
- **operator double ()const**
  Read channel. Check if settable. (see Section 2.29.1)
- **double operator+= (double)**
  Subtract and assign **value** to channel. (see Section 2.20.1)
- **double operator-= (double)**
  Store **value** into channel. (see Section 2.20.1)
- **double operator= (double)**
  Assign **value** to channel. Add and assign **value** to channel. (see Section 2.20.1)
- **virtual bool set (double)**
  (see Section 2.20.1)
- **virtual ~Channel ()**
  (see Section 2.20.1)

2.20.1 Detailed descriptions

Public members

- **Channel ()**

- **virtual Channel* clone ()const**
  Duplicate the channel.

- **virtual bool isSettable ()const**
  Test if settable.

  Return true, if the channel can receive values.

- **double operator+= (double)**
  Subtract and assign value to channel.

- **double operator-= (double)**
  Store value into channel.

  If the channel can be written, store **value** into it and return true, otherwise return false.

- **double operator= (double)**
  Assign value to channel. Add and assign value to channel.

- **virtual bool set (double)**

- **virtual ~Channel ()**

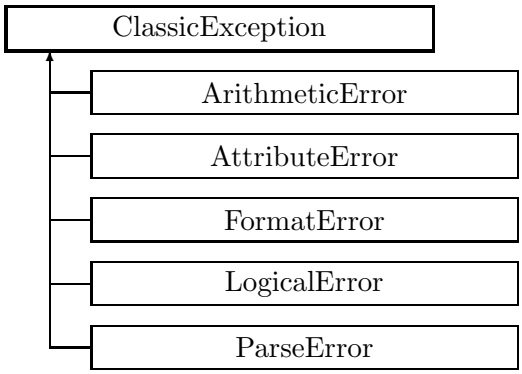

Figure 2.19: Inheritance for class ClassicException

2.21 class ClassicException

The abstract base class for all exceptions in CLASSIC.

Type:	Instantiable
Include file:	./Utilities/ClassicException.hh

Synopsis (including inherited members)

Public members

- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)

Protected members

- **ClassicException (const string&,const string&)**
  The usual constructor. (see Section 2.21.1)
- **ClassicException (const ClassicException&)**
  (see Section 2.21.1)
- **ClassicException ()**
  (see Section 2.21.1)
- **const string message**
  (see Section 2.21.1)
- **const string method**
  (see Section 2.21.1)
- **virtual ~ClassicException ()**
  (see Section 2.21.1)

2.21.1 Detailed descriptions

Public members

- **virtual const string& what ()const**
  Return the message string for the exception.
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception.

Protected members

- **ClassicException (const string&,const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  msg the message string identifying the exception
- **ClassicException (const ClassicException&)**
- **ClassicException ()**
- **const string message**
- **const string method**
- **virtual ~ClassicException ()**

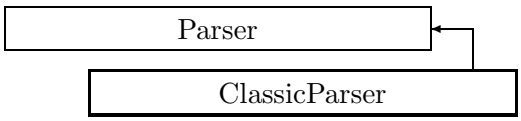

Figure 2.20: Inheritance for class ClassicParser

2.22 class ClassicParser

A parser for the standard input format (SIF) described in
center D. C. Carey and F. Ch. Iselin:br A Standard Input Language for Particle Beam and Acceleratorbr Computer Programs.br 1984 Snowmass Summer Study. /center This is the default parser for the CLASSIC standard input language. The program should declare only one object of class ClassicParser.

Type:	Instantiable
Superclasses:	public Parser
Include file:	./Parser/ClassicParser.hh

Synopsis (including inherited members)

Public members

- **ClassicParser ()**
  (see Section 2.22.1)
- **virtual void parse (Statement&)const**
  Parse and execute the statement. (see Section 2.22.1)
- **virtual Statement* readStatement (TokenStream*)const**
  Read complete statement from token stream. (see Section 2.22.1)
- **virtual void run (TokenStream*)const**
  Read statements and parse. (see Section 2.22.1)
- **virtual ˜ClassicParser ()**
  (see Section 2.22.1)

2.22.1 Detailed descriptions

Public members

- **ClassicParser ()**

- **virtual void parse (Statement&)const**
  Parse and execute the statement.
- **virtual Statement* readStatement (TokenStream*)const**
  Read complete statement from token stream.
- **virtual void run (TokenStream*)const**
  Read statements and parse.

  Read one statement at a time on token stream, parse it, and execute it.

- **virtual ~ClassicParser ()**

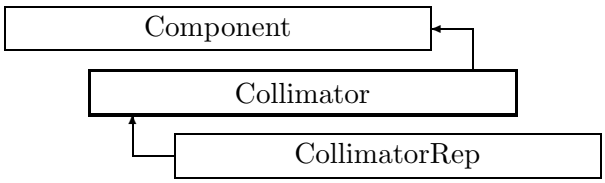

Figure 2.21: Inheritance for class Collimator

2.23 class Collimator

Abstract collimator.
Class Collimator defines the abstract interface for a collimator.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Collimator.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Collimator (const string&)**
  Constructor with given name. (see Section 2.23.1)
- **Collimator ()**
  (see Section 2.23.1)
- **Collimator (const Collimator&)**
  (see Section 2.23.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Collimator. (see Section 2.23.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)

- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual double getXsize ()const**
  Return the horizontal half-aperture. (see Section 2.23.1)
- **virtual double getYsize ()const**
  Return the vertical half-aperture. (see Section 2.23.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)

- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Collimator ()**
  (see Section 2.23.1)

2.23.1 Detailed descriptions

Public members

- **Collimator (const string&)**
  Constructor with given name.
- **Collimator ()**
- **Collimator (const Collimator&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Collimator.

- **virtual double getXsize ()const**
  Return the horizontal half-aperture.
- **virtual double getYsize ()const**
  Return the vertical half-aperture.
- **virtual ~Collimator ()**

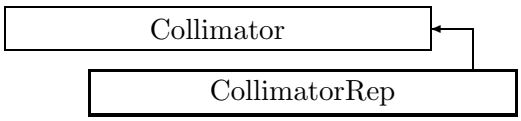

Figure 2.22: Inheritance for class CollimatorRep

2.24 class CollimatorRep

Representation for a collimator.

Type:	Instantiable
Superclasses:	public Collimator
Include file:	./BeamlineCore/CollimatorRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **CollimatorRep (const string&)**
  Constructor with given name. (see Section 2.24.1)
- **CollimatorRep ()**
  (see Section 2.24.1)
- **CollimatorRep (const CollimatorRep&)**
  (see Section 2.24.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Collimator. (see Section 2.23.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.24.1)

- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.24.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual NullField& getField ()**
  Get field. (see Section 2.24.1)
- **virtual const NullField& getField ()const**
  Get field. (see Section 2.24.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.24.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.24.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.24.1)

- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.24.1)
- **virtual double getXsize ()const**
  Return the horizontal half-aperture. (see Section 2.24.1)
- **virtual double getYsize ()const**
  Return the vertical half-aperture. (see Section 2.24.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)

- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void setXsize (double)**
  Return the horizontal half-aperture. (see Section 2.24.1)
- **virtual void setYsize (double)**
  Return the vertical half-aperture. (see Section 2.24.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~CollimatorRep ()**
  (see Section 2.24.1)

2.24.1 Detailed descriptions

Public members

- **CollimatorRep (const string&)**
  Constructor with given name.
- **CollimatorRep ()**
- **CollimatorRep (const CollimatorRep&)**

- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.

- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.

- **virtual NullField& getField ()**
  Get field.

  Version for non-constant object.

- **virtual const NullField& getField ()const**
  Get field.

  Version for constant object.

- **virtual StraightGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual double getXsize ()const**
  Return the horizontal half-aperture.

- **virtual double getYsize ()const**
  Return the vertical half-aperture.

- **virtual void setXsize (double)**
  Return the horizontal half-aperture.

- **virtual void setYsize (double)**
  Return the vertical half-aperture.

- **virtual ~CollimatorRep ()**

2.25 class ComplexEigen

Complex eigenvector routines.
Representation of eigenvalues and eigenvectors for a matrix of type Matrix<complex<double> >.

Type:	Instantiable
Include file:	./Algebra/ComplexEigen.hh

Synopsis (including inherited members)

Public members

- **ComplexEigen (const Matrix<complex<double> >&,bool)**
  Constructor. (see Section 2.25.1)
- **ComplexEigen ()**
  (see Section 2.25.1)
- **ComplexEigen (const ComplexEigen&)**
  (see Section 2.25.1)
- **const Vector<complex<double> >& eigenValues ()const**
  Get eigenvalues. (see Section 2.25.1)
- **const Matrix<complex<double> >& eigenVectors ()const**
  Get eigenvectors. (see Section 2.25.1)
- **~ComplexEigen ()**
  (see Section 2.25.1)

2.25.1 Detailed descriptions

Public members

- **ComplexEigen (const Matrix<complex<double> >&,bool)**
  Constructor.

  Construct the vector of eigenvalues for the matrix **M**. If **vec** is true, the matrix of eigenvectors is also built.
- **ComplexEigen ()**
- **ComplexEigen (const ComplexEigen&)**
- **const Vector<complex<double> >& eigenValues ()const**
  Get eigenvalues.

  Return the eigenvalues as a complex vector.
- **const Matrix<complex<double> >& eigenVectors ()const**
  Get eigenvectors.

  Return the eigenvectors as the column vectors of a complex matrix.

- **~ComplexEigen ()**

2.26 class Component

Interface for a single beam element.
Class Component defines the abstract interface for an arbitrary single component in a beam line. A component is the basic element in the accelerator model, like a dipole, a quadrupole, etc. It is normally associated with an electro-magnetic field, which may be null.

Type:	abstract
Superclasses:	public ElementBase
Include file:	./AbsBeamline/Component.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Component (const string&)**
  Constructor with given name. (see Section 2.26.1)
- **Component ()**
  (see Section 2.26.1)
- **Component (const Component&)**
  (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor. (see Section 2.52.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)

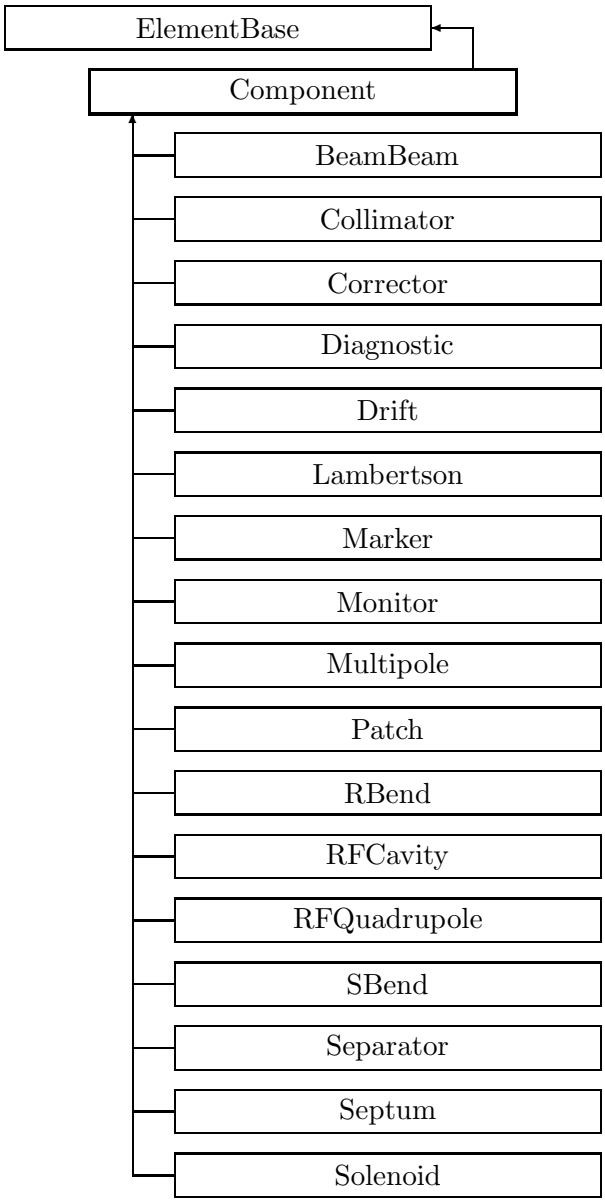

Figure 2.23: Inheritance for class Component

- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)

- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)

- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Component ()**
  (see Section 2.26.1)

2.26.1 Detailed descriptions

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point.

  Return the value of the time-independent part of the magnetic field at point **P**.
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point.

  Return the value of the time-dependent part of the magnetic field at point **P** for time **t**.
- **Component (const string&)**
  Constructor with given name.
- **Component ()**
- **Component (const Component&)**
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point.

  Return the value of the time-independent part of both electric and magnetic fields at point **P**.

- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point.

  Return the value of the time-dependent part of both electric and magnetic fields at point **P** for time **t**.

- **EVector Efield (const Point3D&)const**
  Return the field in a point.

  Return the value of the time-independent part of the electric field at point **P**.

- **EVector Efield (const Point3D&,double)const**
  Return the field in a point.

  Return the value of the time-dependent part of the electric field at point **P** for time **t**.

- **virtual const ElementBase& getDesign ()const**
  Return design element.

  If a component is a wrapper, this method returns a pointer to its underlying design element, otherwise a pointer to this component. The default version returns "this".

- **virtual EMField& getField ()**
  Return field.

  The representation of the electro-magnetic field of the component (version for non-constant object).

- **virtual const EMField& getField ()const**
  Return field.

  The representation of the electro-magnetic field of the component (version for constant object).

- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch.

  This catch-all method implements a hook for tracking a particle bunch through a non-standard component. The default version throws a LogicalError.

- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map.

  This catch-all method implements a hook for tracking a transfer map through a non-standard component. The default version throws a LogicalError.

- **virtual ~Component ()**

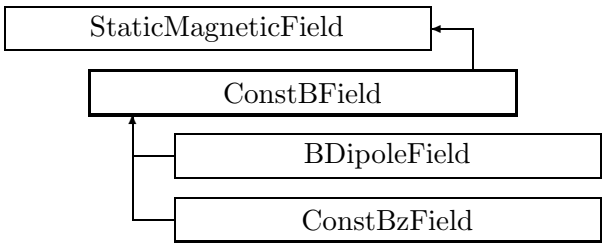

Figure 2.24: Inheritance for class ConstBField

2.27 class ConstBField

A homogenous magnetostatic field.
A static magnetic field independent of (x,y,z).

Type:	abstract
Superclasses:	public StaticMagneticField
Include file:	./Fields/ConstBField.hh

Synopsis (including inherited members)

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **ConstBField ()**
  Default constructor. (see Section 2.27.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)
- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)

- **virtual double getBx ()const**
  Get component. (see Section 2.27.1)
- **virtual double getBy ()const**
  Get component. (see Section 2.27.1)
- **virtual double getBz ()const**
  Get component. (see Section 2.27.1)
- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)
- **virtual void scale (double)**
  Scale the field. (see Section 2.49.1)
- **virtual void setBx (double)**
  Set component. (see Section 2.27.1)
- **virtual void setBy (double)**
  Set component. (see Section 2.27.1)
- **virtual void setBz (double)**
  Set component. (see Section 2.27.1)
- **virtual ~ConstBField ()**
  (see Section 2.27.1)

2.27.1 Detailed descriptions

Public members

- **ConstBField ()**
  Default constructor.

  Constructs null field.
- **virtual double getBx ()const**
  Get component.

  Return the x-component of the magnetic field in T.
- **virtual double getBy ()const**
  Get component.

  Return the y-component of the magnetic field in T.
- **virtual double getBz ()const**
  Get component.

  Return the z-component of the magnetic field in T.
- **virtual void setBx (double)**
  Set component.

  Assign the x-component of the magnetic field in T.

- **virtual void setBy (double)**
  Set component.

  Assign the y-component of the magnetic field in T.

- **virtual void setBz (double)**
  Set component.

  Assign the z-component of the magnetic field in T.

- **virtual ~ConstBField ()**

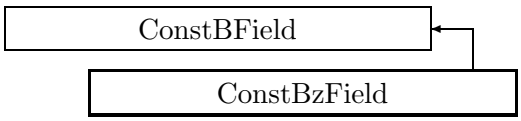

Figure 2.25: Inheritance for class ConstBzField

2.28 class ConstBzField

A homogeneous magnetostatic field in z-direction.

Type:	Instantiable
Superclasses:	public ConstBField
Include file:	./Fields/ConstBzField.hh

Synopsis (including inherited members)

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.28.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.28.1)
- **ConstBzField ()**
  Default constructor. (see Section 2.28.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)
- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)
- **virtual double getBx ()const**
  Get component. (see Section 2.27.1)
- **virtual double getBy ()const**
  Get component. (see Section 2.27.1)

- **virtual double getBz ()const**
  Get component. (see Section 2.28.1)
- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)
- **virtual void scale (double)**
  Scale the field. (see Section 2.28.1)
- **virtual void setBx (double)**
  Set component. (see Section 2.27.1)
- **virtual void setBy (double)**
  Set component. (see Section 2.27.1)
- **virtual void setBz (double)**
  Set component. (see Section 2.28.1)
- **virtual ~ConstBzField ()**
  (see Section 2.28.1)

2.28.1 Detailed descriptions

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field.

  Return the time-independent part of the magnetic field in point **P**. This override forces implementation in derived classes.
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field.

  Return the magnetic field at time **t** in point **P**. This override forces implementation in derived classes.
- **ConstBzField ()**
  Default constructor.

  Constructs a null field.
- **virtual double getBz ()const**
  Get component.

  Return the z-component of the magnetic field in T.
- **virtual void scale (double)**
  Scale the field.

  Multiply the field by **scalar**.
- **virtual void setBz (double)**
  Set component.

  Assign the z-component of the magnetic field in T.

- **virtual ~ConstBzField ()**

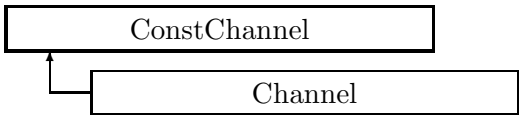

Figure 2.26: Inheritance for class ConstChannel

2.29 class ConstChannel

Abstract interface for read-only access to variable.
Class ConstChannel allows access to an arbitrary **double** value.

Type:	abstract
Include file:	./Channels/ConstChannel.hh

Synopsis (including inherited members)

Public members

- **ConstChannel ()**
  (see Section 2.29.1)
- **virtual ConstChannel* clone ()const**
  Duplicate the channel. (see Section 2.29.1)
- **virtual bool get (double&)const**
  Read channel. (see Section 2.29.1)
- **virtual bool isSettable ()const**
  (see Section 2.29.1)
- **operator double ()const**
  Read channel. Check if settable. (see Section 2.29.1)
- **virtual ~ConstChannel ()**
  (see Section 2.29.1)

2.29.1 Detailed descriptions

Public members

- **ConstChannel ()**

- **virtual ConstChannel* clone ()const**
  Duplicate the channel.
- **virtual bool get (double&)const**
  Read channel.

  If the channel can be read, set **value** and return true, otherwise return false.
- **virtual bool isSettable ()const**

- **operator double ()const**
  Read channel. Check if settable.

  Return the value read from the channel. Return false for this class (channel cannot be set).

- **virtual ~ConstChannel ()**

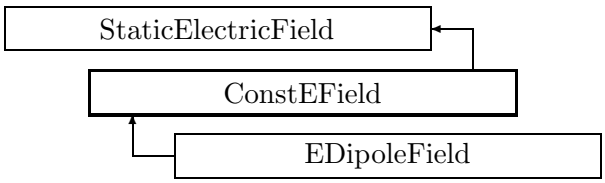

Figure 2.27: Inheritance for class ConstEField

2.30 class ConstEField

A homogeneous electricstatic field.

Type:	abstract
Superclasses:	public StaticElectricField
Include file:	./Fields/ConstEField.hh

Synopsis (including inherited members)

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **ConstEField ()**
  Default constructor. (see Section 2.30.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)
- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)
- **virtual double getEx ()const**
  Get component. (see Section 2.30.1)

- **virtual double getEy ()const**
  Get component. (see Section 2.30.1)
- **virtual double getEz ()const**
  Get component. (see Section 2.30.1)
- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)
- **virtual void scale (double)**
  Scale the field. (see Section 2.49.1)
- **virtual void setEx (double)**
  Set component. (see Section 2.30.1)
- **virtual void setEy (double)**
  Set component. (see Section 2.30.1)
- **virtual void setEz (double)**
  Set component. (see Section 2.30.1)
- **virtual ~ConstEField ()**
  (see Section 2.30.1)

2.30.1 Detailed descriptions

Public members

- **ConstEField ()**
  Default constructor.

  Constructs null field.
- **virtual double getEx ()const**
  Get component.

  Return the x-component of the electric field in A/m.
- **virtual double getEy ()const**
  Get component.

  Return the y-component of the electric field in A/m.
- **virtual double getEz ()const**
  Get component.

  Return the z-component of the electric field in A/m.
- **virtual void setEx (double)**
  Set component.

  Assign the x-component of the electric field in A/m.
- **virtual void setEy (double)**
  Set component.

  Assign the y-component of the electric field in A/m.

- **virtual void setEz (double)**
  Set component.

  Assign the z-component of the electric field in A/m.

- **virtual ~ConstEField ()**

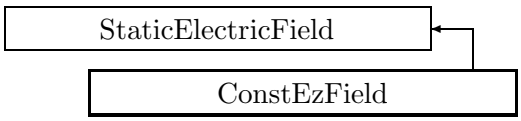

Figure 2.28: Inheritance for class ConstEzField

2.31 class ConstEzField

A homogeneous electrostatic field in z-direction.

Type:	Instantiable
Superclasses:	public StaticElectricField
Include file:	./Fields/ConstEzField.hh

Synopsis (including inherited members)

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **ConstEzField ()**
  Default constructor. (see Section 2.31.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.31.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.31.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)
- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)
- **virtual double getEz ()const**
  Get component. (see Section 2.31.1)
- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)

- **virtual void scale (double)**
  Scale the field. (see Section 2.31.1)
- **virtual void setEz (double)**
  Set component. (see Section 2.31.1)
- **virtual ~ConstEzField ()**
  (see Section 2.31.1)

2.31.1 Detailed descriptions

Public members

- **ConstEzField ()**
  Default constructor.

  Constructs a null field.
- **virtual EVector Efield (const Point3D&)const**
  Get field.

  Return the time-independent part of the electric field in point **P**.
- **virtual EVector Efield (const Point3D&,double)const**
  Get field.

  Return the electric field at time **t** in point **P**.
- **virtual double getEz ()const**
  Get component.

  Return the x-component of the electric field in A/m.
- **virtual void scale (double)**
  Scale the field.

  Multiply the field by **scalar**.
- **virtual void setEz (double)**
  Set component.

  Assign the z-component of the electric field in A/m.
- **virtual ~ConstEzField ()**

2.32 template class ConstSliceIterator <class>

Iterator for array slice.
An iterator permitting to iterate on a constant array with a stride different from 1.

Type:	Instantiable
Include file:	./Algebra/SliceIterator.hh

Synopsis (including inherited members)

Public members

- **ConstSliceIterator (const T*,ptrdiff_t)**
  Constructor. (see Section 2.32.1)
- **ConstSliceIterator (const ConstSliceIterator<T>&)**
  (see Section 2.32.1)
- **typedef ptrdiff_t difference_type**
  The pointer difference type. (see Section 2.32.1)
- **typedef std::random_access_iterator_tag iterator_category**
  The iterator tag, taken from the standard template library. (see Section 2.32.1)
- **bool operator!= (const ConstSliceIterator<T>&)const**
  (see Section 2.32.1)
- **const T& operator* ()const**
  Dereference. (see Section 2.32.1)
- **ConstSliceIterator<T> operator+ (ptrdiff_t)const**
  Add multiple stride. (see Section 2.32.1)
- **ConstSliceIterator<T>& operator++ ()**
  Increment (iterate forward). (see Section 2.32.1)
- **ConstSliceIterator<T> operator++ (int)**
  Increment (iterate forward). (see Section 2.32.1)
- **ConstSliceIterator<T>& operator+= (ptrdiff_t)**
  Add multiple stride and assign. (see Section 2.32.1)
- **ConstSliceIterator<T> operator- (ptrdiff_t)const**
  Subtract multiple stride. (see Section 2.32.1)
- **difference_type operator- (const ConstSliceIterator<T>&)const**
  Get pointer difference. (see Section 2.32.1)
- **ConstSliceIterator<T>& operator– ()**
  Decrement (iterate backward). (see Section 2.32.1)
- **ConstSliceIterator<T> operator– (int)**
  Decrement (iterate backward). (see Section 2.32.1)

- **ConstSliceIterator<T>& operator-= (ptrdiff_t)**
  Subtract multiple stride and assign. (see Section 2.32.1)
- **ConstSliceIterator<T>& operator= (const ConstSliceIterator<T>&)**
  (see Section 2.32.1)
- **bool operator== (const ConstSliceIterator<T>&)const**
  (see Section 2.32.1)
- **const T& operator[] (int)const**
  Delegate. (see Section 2.32.1)
- **typedef const T* pointer**
  The pointer type. (see Section 2.32.1)
- **typedef const T& reference**
  The reference type. (see Section 2.32.1)
- **typedef const T value_type**
  The value type. (see Section 2.32.1)

2.32.1 Detailed descriptions

Public members

- **ConstSliceIterator (const T*,ptrdiff_t)**
  Constructor.

  Construct iterator for **array**, given a **stride**.
- **ConstSliceIterator (const ConstSliceIterator<T>&)**

- **typedef ptrdiff_t difference_type**
  The pointer difference type.
- **typedef std::random_access_iterator_tag iterator_category**
  The iterator tag, taken from the standard template library.
- **bool operator!= (const ConstSliceIterator<T>&)const**

- **const T& operator* ()const**
  Dereference.
- **ConstSliceIterator<T> operator+ (ptrdiff_t)const**
  Add multiple stride.
- **ConstSliceIterator<T>& operator++ ()**
  Increment (iterate forward).
- **ConstSliceIterator<T> operator++ (int)**
  Increment (iterate forward).

- **ConstSliceIterator<T>& operator+= (ptrdiff_t)**
  Add multiple stride and assign.
- **ConstSliceIterator<T> operator- (ptrdiff_t)const**
  Subtract multiple stride.
- **difference_type operator- (const ConstSliceIterator<T>&)const**
  Get pointer difference.
- **ConstSliceIterator<T>& operator– ()**
  Decrement (iterate backward).
- **ConstSliceIterator<T> operator– (int)**
  Decrement (iterate backward).
- **ConstSliceIterator<T>& operator-= (ptrdiff_t)**
  Subtract multiple stride and assign.
- **ConstSliceIterator<T>& operator= (const ConstSliceIterator<T>&)**
- **bool operator== (const ConstSliceIterator<T>&)const**
- **const T& operator[] (int)const**
  Delegate.
- **typedef const T* pointer**
  The pointer type.
- **typedef const T& reference**
  The reference type.
- **typedef const T value_type**
  The value type.

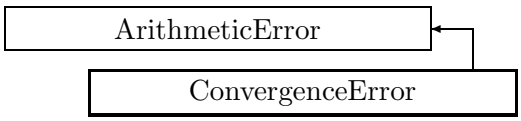

Figure 2.29: Inheritance for class ConvergenceError

2.33 class ConvergenceError

Convergence error exception.
This exception is thrown, when an iterative algorithm fails to converge.

Type:	Instantiable
Superclasses:	public ArithmeticError
Include file:	./Utilities/ConvergenceError.hh

Synopsis (including inherited members)

Public members

- **ConvergenceError (const string&,const string&)**
  The usual constructor. (see Section 2.33.1)
- **ConvergenceError (const ConvergenceError&)**
  (see Section 2.33.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~ConvergenceError ()**
  (see Section 2.33.1)

2.33.1 Detailed descriptions

Public members

- **ConvergenceError (const string&,const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  msg the message string identifying the exception
- **ConvergenceError (const ConvergenceError&)**
- **virtual ~ConvergenceError ()**

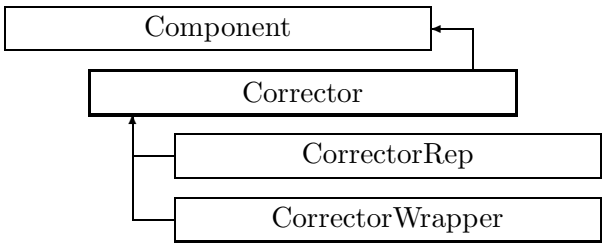

Figure 2.30: Inheritance for class Corrector

2.34 class Corrector

Interface for general corrector.
Class Corrector defines the abstract interface for closed orbit correctors.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Corrector.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Corrector (const string&)**
  Corrector is off (inactive). Corrector acts on x-plane. Corrector acts on y-plane. Corrector acts on both planes. Constructor with given name. (see Section 2.34.1)
- **Corrector ()**
  (see Section 2.34.1)
- **Corrector (const Corrector&)**
  (see Section 2.34.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **enum Plane**
  Plane selection. (see Section 2.34.1)

- **virtual void accept (BeamlineVisitor&)const**
  Apply a visitor to Corrector. (see Section 2.34.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BDipoleField& getField ()**
  Return the corrector field. (see Section 2.34.1)
- **virtual const BDipoleField& getField ()const**
  Return the corrector field. (see Section 2.34.1)

- **virtual StraightGeometry& getGeometry ()**
  Return the corrector geometry. (see Section 2.34.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Return the corrector geometry. Version for const object. (see Section 2.34.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Plane getPlane ()const**
  Return the plane on which the corrector acts. (see Section 2.34.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Corrector ()**
  (see Section 2.34.1)

Protected members

- **void operator= (const Corrector&)**
  (see Section 2.34.1)
- **RCObject& operator= (const RCObject&)**
  (see Section 2.126.1)
- **mutable bool shareFlag**
  (see Section 2.52.1)

2.34.1 Detailed descriptions

Public members

- **Corrector (const string&)**
  Corrector is off (inactive). Corrector acts on x-plane. Corrector acts on y-plane. Corrector acts on both planes. Constructor with given name.
- **Corrector ()**
- **Corrector (const Corrector&)**
- **enum Plane**
  Plane selection.
- **virtual void accept (BeamlineVisitor&)const**
  Apply a visitor to Corrector.
- **virtual BDipoleField& getField ()**
  Return the corrector field.

  Version for non-constant object.
- **virtual const BDipoleField& getField ()const**
  Return the corrector field.

  Version for constant object.
- **virtual StraightGeometry& getGeometry ()**
  Return the corrector geometry.
- **virtual const StraightGeometry& getGeometry ()const**
  Return the corrector geometry. Version for const object.
- **virtual Plane getPlane ()const**
  Return the plane on which the corrector acts.
- **virtual ~Corrector ()**

Protected members

- **void operator= (const Corrector&)**

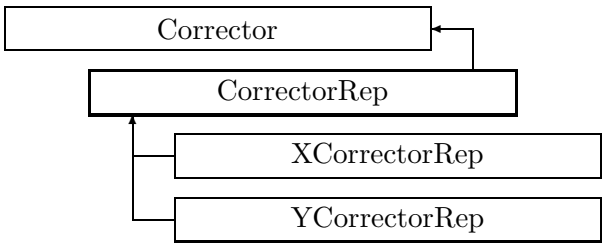

Figure 2.31: Inheritance for class CorrectorRep

2.35 class CorrectorRep

Representation of a closed orbit corrector.
The base class acts on both planes.

Type:	Instantiable
Superclasses:	public Corrector
Include file:	./BeamlineCore/CorrectorRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **CorrectorRep (const string&)**
  Constructor with given name. (see Section 2.35.1)
- **CorrectorRep ()**
  (see Section 2.35.1)
- **CorrectorRep (const CorrectorRep&)**
  (see Section 2.35.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **enum Plane**
  Plane selection. (see Section 2.34.1)

- **virtual void accept (BeamlineVisitor&)const**
  Apply a visitor to Corrector. (see Section 2.34.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.35.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getBx ()const**
  Get horizontal field component in Teslas. (see Section 2.35.1)
- **virtual double getBy ()const**
  Get vertical field component in Teslas. (see Section 2.35.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.35.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)

- **virtual BDipoleField& getField ()**
  Get corrector field. (see Section 2.35.1)
- **virtual const BDipoleField& getField ()const**
  Get corrector field. Version for const corrector. (see Section 2.35.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.35.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.35.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.35.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Plane getPlane ()const**
  Get plane(s) of action. (see Section 2.35.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.35.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.35.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)

- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setActive (bool)**
  Set active flag. (see Section 2.35.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setBx (double)**
  Set horizontal field component in Teslas. (see Section 2.35.1)
- **virtual void setBy (double)**
  Set vertical field component in Teslas. (see Section 2.35.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~CorrectorRep ()**
  (see Section 2.35.1)

Protected members

- **bool active**
  The active/inactive flag. (see Section 2.35.1)
- **BDipoleField field**
  The corrector strengths. (see Section 2.35.1)
- **StraightGeometry geometry**
  The corrector geometry. (see Section 2.35.1)
- **void operator= (const Corrector&)**
  (see Section 2.34.1)
- **RCObject& operator= (const RCObject&)**
  (see Section 2.126.1)
- **mutable bool shareFlag**
  (see Section 2.52.1)

2.35.1 Detailed descriptions

Public members

- **CorrectorRep (const string&)**
  Constructor with given name.
- **CorrectorRep ()**
- **CorrectorRep (const CorrectorRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual double getBx ()const**
  Get horizontal field component in Teslas.
- **virtual double getBy ()const**
  Get vertical field component in Teslas.
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.
- **virtual BDipoleField& getField ()**
  Get corrector field.
- **virtual const BDipoleField& getField ()const**
  Get corrector field. Version for const corrector.

- **virtual StraightGeometry& getGeometry ()**
  Get geometry.

  Return the element geometry. Version for non-constant object.

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.

  Return the element geometry Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual Plane getPlane ()const**
  Get plane(s) of action.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors.

  Build a FieldWrapper pointing to the corrector and return a pointer to that wrapper.

- **virtual void setActive (bool)**
  Set active flag.

  If **flag** is true, the corrector is activated, otherwise deactivated.

- **virtual void setBx (double)**
  Set horizontal field component in Teslas.

- **virtual void setBy (double)**
  Set vertical field component in Teslas.

- **virtual ~CorrectorRep ()**

Protected members

- **bool active**
  The active/inactive flag.

- **BDipoleField field**
  The corrector strengths.

- **StraightGeometry geometry**
  The corrector geometry.

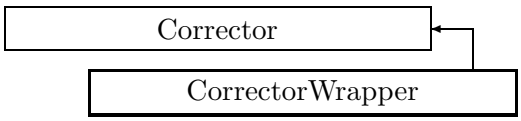

Figure 2.32: Inheritance for class CorrectorWrapper

2.36 class CorrectorWrapper

Representation for a perturbed closed orbit corrector.
A CorrectorWrapper represents a unique instance of a corrector magnet in the accelerator model. It defines imperfections of the field, related to an "ideal" magnet contained in the wrapper.

Type:	Instantiable
Superclasses:	public Corrector
Include file:	./ComponentWrappers/CorrectorWrapper.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **CorrectorWrapper (Corrector*)**
  Constructor. (see Section 2.36.1)
- **CorrectorWrapper (const CorrectorWrapper&)**
  (see Section 2.36.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **enum Plane**
  Plane selection. (see Section 2.34.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to modified corrector. (see Section 2.36.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Make clone. (see Section 2.36.1)
- **virtual ElementBase* copyStructure ()**
  Make structural copy. (see Section 2.36.1)
- **virtual BDipoleField& errorField ()const**
  Get corrector field error. (see Section 2.36.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const Corrector& getDesign ()const**
  Get design corrector. (see Section 2.36.1)
- **virtual Corrector& getDesign ()**
  Get design corrector. (see Section 2.36.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BDipoleField& getField ()**
  Get corrector field. (see Section 2.36.1)
- **virtual const BDipoleField& getField ()const**
  Get corrector field. (see Section 2.36.1)

- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.36.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.36.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Plane getPlane ()const**
  Get plane of action. (see Section 2.36.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.36.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Make wrapper for this corrector. (see Section 2.36.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.36.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.36.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.36.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Remove all wrappers. (see Section 2.36.1)
- **virtual const ElementBase* removeWrappers ()const**
  Remove all wrappers. (see Section 2.36.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~CorrectorWrapper ()**
  (see Section 2.36.1)

2.36.1 Detailed descriptions

Public members

- **CorrectorWrapper (Corrector*)**
  Constructor.

  Construct a wrapper for the argument. The wrapper is not sharable by default.
- **CorrectorWrapper (const CorrectorWrapper&)**

- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to modified corrector.
- **virtual ElementBase* clone ()const**
  Make clone.
- **virtual ElementBase* copyStructure ()**
  Make structural copy.
- **virtual BDipoleField& errorField ()const**
  Get corrector field error.

  This method can be used to get or set the error field. The error field offset is mutable, so as to allow changing it in a constant structure.
- **virtual const Corrector& getDesign ()const**
  Get design corrector.

  Version for constant object.
- **virtual Corrector& getDesign ()**
  Get design corrector.

  Version for non-constant object.
- **virtual BDipoleField& getField ()**
  Get corrector field.

  Version for non-constant object.
- **virtual const BDipoleField& getField ()const**
  Get corrector field.

  Version for constant object.
- **virtual StraightGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.
- **virtual Plane getPlane ()const**
  Get plane of action.
- **virtual const string& getType ()const**
  Get element type string.
- **virtual ElementBase* makeFieldWrapper ()**
  Make wrapper for this corrector.

  Return **this**, since this is already a field wrapper.

- **virtual void makeSharable ()**
  Set sharable flag.

  The whole structure depending on **this** is marked as sharable. After this call a **copyStructure()** call reuses the element.

- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper.

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper.

- **virtual ElementBase* removeWrappers ()**
  Remove all wrappers.

- **virtual const ElementBase* removeWrappers ()const**
  Remove all wrappers.

- **virtual ~CorrectorWrapper ()**

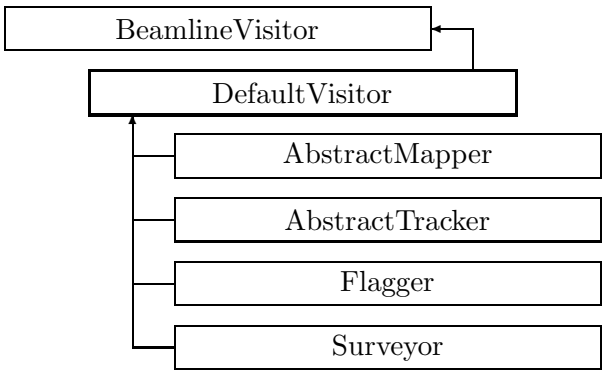

Figure 2.33: Inheritance for class DefaultVisitor

2.37 class DefaultVisitor

Default algorithms.
A default implementation for all visitors that can iterate over a beam line representation. This abstract base class implements the default behaviour for the structural classes Beamline and FlaggedElmPtr, and for all wrappers. It also holds the data required for all visitors in a protected area.

Type:	abstract
Superclasses:	public BeamlineVisitor
Include file:	./Algorithms/DefaultVisitor.hh

Synopsis (including inherited members)

Public members

- **DefaultVisitor (const Beamline&,bool,bool)**
  Constructor. (see Section 2.37.1)
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper.. (see Section 2.37.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam. (see Section 2.37.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.37.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.37.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector. (see Section 2.37.1)

- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.37.1)
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic. (see Section 2.37.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift. (see Section 2.37.1)
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.37.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson. (see Section 2.37.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.37.1)
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker. (see Section 2.37.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor. (see Section 2.37.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole. (see Section 2.37.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.37.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.37.1)
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend. (see Section 2.37.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.37.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity. (see Section 2.37.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole. (see Section 2.37.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend. (see Section 2.37.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.37.1)

- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator. (see Section 2.37.1)
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum. (see Section 2.37.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid. (see Section 2.37.1)
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.37.1)
- **virtual ~DefaultVisitor ()**
  (see Section 2.37.1)

Protected members

- **bool back_beam**
  (see Section 2.37.1)
- **bool back_track**
  (see Section 2.37.1)
- **double flip_B**
  (see Section 2.37.1)
- **double flip_s**
  (see Section 2.37.1)
- **const Beamline& itsLine**
  (see Section 2.37.1)
- **bool local_flip**
  (see Section 2.37.1)

2.37.1 Detailed descriptions

Public members

- **DefaultVisitor (const Beamline&,bool,bool)**
  Constructor.

  Arguments:

  The beamline to be used.

  If true, the beam runs backwards through the line.

  If true, we track against the beam.
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline.
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper..

- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam.
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line.
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator.
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component.
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector.
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper..
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic.
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift.
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr.
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator.
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson.
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping.
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker.
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor.
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole.
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper..
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch.
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend.

- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper..
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity.
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole.
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend.
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper..
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator.
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum.
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid.
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking.
- **virtual ~DefaultVisitor ()**

Protected members

- **bool back_beam**
- **bool back_track**
- **double flip_B**
- **double flip_s**
- **const Beamline& itsLine**
- **bool local_flip**

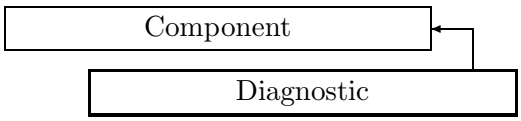

Figure 2.34: Inheritance for class Diagnostic

2.38 class Diagnostic

Interface for beam diagnostics.
Class Diagnostic defines the abstract interface for a beam diagnostic.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Diagnostic.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Diagnostic (const string&)**
  Constructor with given name. (see Section 2.38.1)
- **Diagnostic ()**
  (see Section 2.38.1)
- **Diagnostic (const Diagnostic&)**
  (see Section 2.38.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Diagnostic. (see Section 2.38.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)

- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Diagnostic ()**
  (see Section 2.38.1)

2.38.1 Detailed descriptions

Public members

- **Diagnostic (const string&)**
  Constructor with given name.
- **Diagnostic ()**
- **Diagnostic (const Diagnostic&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Diagnostic.
- **virtual ~Diagnostic ()**

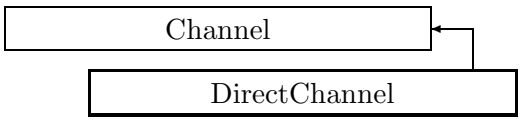

Figure 2.35: Inheritance for class DirectChannel

2.39 class DirectChannel

Direct access to a double variable.
Class DirectChannel allows direct access to a **double** variable.

Type:	Instantiable
Superclasses:	public Channel
Include file:	./Channels/DirectChannel.hh

Synopsis (including inherited members)

Public members

- **DirectChannel (double&)**
  Constructor. (see Section 2.39.1)
- **DirectChannel (const DirectChannel&)**
  (see Section 2.39.1)
- **virtual DirectChannel* clone ()const**
  Duplicate the channel. (see Section 2.39.1)
- **virtual bool get (double&)const**
  Fetch from channel. (see Section 2.39.1)
- **virtual bool isSettable ()const**
  Test if settable. (see Section 2.20.1)
- **operator double ()const**
  Read channel. Check if settable. (see Section 2.29.1)
- **double operator+= (double)**
  Subtract and assign **value** to channel. (see Section 2.20.1)
- **double operator-= (double)**
  Store **value** into channel. (see Section 2.20.1)
- **double operator= (double)**
  Assign **value** to channel. Add and assign **value** to channel. (see Section 2.20.1)
- **virtual bool set (double)**
  Store into channel. (see Section 2.39.1)
- **virtual ~DirectChannel ()**
  (see Section 2.39.1)

2.39.1 Detailed descriptions

Public members

- **DirectChannel (double&)**
  Constructor.

  The constructed channel gives read/write access to the variable **value**. The variable **value** must not be destroyed as long as this channel is active.

- **DirectChannel (const DirectChannel&)**

- **virtual DirectChannel* clone ()const**
  Duplicate the channel.

- **virtual bool get (double&)const**
  Fetch from channel.

  If the channel can be read, set **value** and return true, otherwise return false.

- **virtual bool set (double)**
  Store into channel.

  If the channel can be written, store **value** into it and return true, otherwise return false.

- **virtual ~DirectChannel ()**

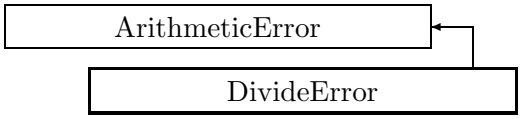

Figure 2.36: Inheritance for class DivideError

2.40 class DivideError

Zero divide error.
This exception is thrown, when a division by zero occurs.

Type:	Instantiable
Superclasses:	public ArithmeticError
Include file:	./Utilities/DivideError.hh

Synopsis (including inherited members)

Public members

- **DivideError (const string&)**
  The usual constructor. (see Section 2.40.1)
- **DivideError (const DivideError&)**
  (see Section 2.40.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~DivideError ()**
  (see Section 2.40.1)

2.40.1 Detailed descriptions

Public members

- **DivideError (const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception
- **DivideError (const DivideError&)**
- **virtual ~DivideError ()**

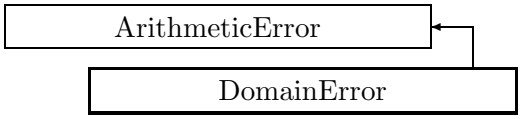

Figure 2.37: Inheritance for class DomainError

2.41 class DomainError

Domain error exception.
This exception is thrown, when a domain error is detected in a function.

Type:	Instantiable
Superclasses:	public ArithmeticError
Include file:	./Utilities/DomainError.hh

Synopsis (including inherited members)

Public members

- **DomainError (const string&)**
  The usual constructor. (see Section 2.41.1)
- **DomainError (const DomainError&)**
  (see Section 2.41.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~DomainError ()**
  (see Section 2.41.1)

2.41.1 Detailed descriptions

Public members

- **DomainError (const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception
- **DomainError (const DomainError&)**
- **virtual ~DomainError ()**

2.42 class DoubleEigen

Double eigenvector routines.
Representation of eigenvalues and eigenvectors for a matrix of type Matrix<double>.

Type:	Instantiable
Include file:	./Algebra/DoubleEigen.hh

Synopsis (including inherited members)

Public members

- **DoubleEigen (const Matrix<double>&,bool)**
  Constructor. (see Section 2.42.1)
- **DoubleEigen ()**
  (see Section 2.42.1)
- **DoubleEigen (const DoubleEigen&)**
  (see Section 2.42.1)
- **Vector<complex<double> > eigenValues ()const**
  Get eigenvalues. (see Section 2.42.1)
- **Matrix<complex<double> > eigenVectors ()const**
  Get eigenvectors. (see Section 2.42.1)
- **Matrix<double> packedEigenVectors ()const**
  Get eigenvectors. (see Section 2.42.1)
- **~DoubleEigen ()**
  (see Section 2.42.1)

2.42.1 Detailed descriptions

Public members

- **DoubleEigen (const Matrix<double>&,bool)**
  Constructor.

  Construct the vector of complex eigenvalues for the matrix **M**. If **vec** is true, the matrix of eigenvectors is also built.
- **DoubleEigen ()**
- **DoubleEigen (const DoubleEigen&)**
- **Vector<complex<double> > eigenValues ()const**
  Get eigenvalues.

  Return the eigenvalues as a complex vector.

- **Matrix<complex<double> > eigenVectors ()const**
  Get eigenvectors.

  Return the eigenvectors as the column vectors of a complex matrix.

- **Matrix<double> packedEigenVectors ()const**
  Get eigenvectors.

  Return the eigenvectors packed in a real matrix. Real eigenvectors appear a single column in the same position as the corresponding real eigenvalue. Complex eigenvalues occur in conjugate pairs, and the corresponding eigenvectors appear as two real columns containing the real and imaginary parts respectively.

- **~DoubleEigen ()**

2.43 template class DragtFinnMap <int>

A Lie algebraic map, factored according to Dragt and Finn.
The type of coefficients is double, and the number of variables is N. It should be noted that some algorithms, notably concatenation of maps, are limited to maps with generators up to order 6.

Type:	Instantiable
Include file:	./FixedAlgebra/DragtFinnMap.hh

Synopsis (including inherited members)

Public members

- **DragtFinnMap (const DragtFinnMap&)**
  (see Section 2.43.1)
- **DragtFinnMap ()**
  Construct identity map. (see Section 2.43.1)
- **DragtFinnMap (const FVps<double,2*N>&)**
  Build factored representation from a Taylor series map. (see Section 2.43.1)
- **DragtFinnMap (const FTps<double,2*N>&)**
  Build factored representation from a pseudo-Hamiltonian. (see Section 2.43.1)
- **DragtFinnMap (const LinearMap<double,2*N>&)**
  Build factored representation from a linear map. (see Section 2.43.1)
- **void assign (const FMatrix<double,2*N,2*N>&)**
  Assign the matrix representing the linear transform. (see Section 2.43.1)
- **void assign (const FTps<double,2*N>&)**
  Assign the complete set of generators. (see Section 2.43.1)
- **void assign (const FLieGenerator<double,N>&)**
  Assign the generator for a selected order. (see Section 2.43.1)
- **DragtFinnMap catenate (const DragtFinnMap&)**
  Substitute (concatenate) with another map in beam order. (see Section 2.43.1)
- **static DragtFinnMap factorBerzForestIrwin (const FTps<double,2*N>&)**
  (see Section 2.43.1)
- **static DragtFinnMap factorDouglas (const FTps<double,2*N>&)**
  (see Section 2.43.1)
- **static DragtFinnMap factorSimple (const FTps<double,2*N>&)**
  (see Section 2.43.1)
- **const FLieGenerator<double,N> getGenerator (int)const**
  Return the generator for a selected order. (see Section 2.43.1)
- **const FTps<double,2*N>& getGenerators ()const**
  Return the complete set of generators. (see Section 2.43.1)

- **const FMatrix<double,2*N,2*N>& getMatrix ()const**
  Return the matrix representing the linear transform. (see Section 2.43.1)
- **FMatrix<double,2*N,2*N>& getMatrix ()**
  Return the matrix representing the linear transform. (see Section 2.43.1)
- **int getOrder ()const**
  Return the order of the generators. (see Section 2.43.1)
- **DragtFinnMap inverse ()**
  Compute inverse map. (see Section 2.43.1)
- **operator ()const**
  Convert to Taylor series map. Convert to Taylor series map. (see Section 2.43.1)
- **void operator+= (const FTps<double,2*N>&)**
  Add to set of generators. (see Section 2.43.1)
- **void operator+= (const FLieGenerator<double,N>&)**
  Add to generator of given order. (see Section 2.43.1)
- **void operator-= (const FTps<double,2*N>&)**
  Subtract from set of generators. (see Section 2.43.1)
- **void operator-= (const FLieGenerator<double,N>&)**
  Subtract from generator of given order. (see Section 2.43.1)
- **const DragtFinnMap& operator= (const DragtFinnMap&)**
  (see Section 2.43.1)
- **DragtFinnMap reverse ()**
  Compute inverse factorisation of map. (see Section 2.43.1)
- **~DragtFinnMap ()**
  (see Section 2.43.1)

2.43.1 Detailed descriptions

Public members

- **DragtFinnMap (const DragtFinnMap&)**

- **DragtFinnMap ()**
  Construct identity map.
- **DragtFinnMap (const FVps<double,2*N>&)**
  Build factored representation from a Taylor series map.
- **DragtFinnMap (const FTps<double,2*N>&)**
  Build factored representation from a pseudo-Hamiltonian.
- **DragtFinnMap (const LinearMap<double,2*N>&)**
  Build factored representation from a linear map.

- **void assign (const FMatrix<double,2*N,2*N>&)**
  Assign the matrix representing the linear transform.
- **void assign (const FTps<double,2*N>&)**
  Assign the complete set of generators.
- **void assign (const FLieGenerator<double,N>&)**
  Assign the generator for a selected order.
- **DragtFinnMap catenate (const DragtFinnMap&)**
  Substitute (concatenate) with another map in beam order.
- **static DragtFinnMap factorBerzForestIrwin (const FTps<double,2*N>&)**
- **static DragtFinnMap factorDouglas (const FTps<double,2*N>&)**
- **static DragtFinnMap factorSimple (const FTps<double,2*N>&)**
- **const FLieGenerator<double,N> getGenerator (int)const**
  Return the generator for a selected order.
- **const FTps<double,2*N>& getGenerators ()const**
  Return the complete set of generators.
- **const FMatrix<double,2*N,2*N>& getMatrix ()const**
  Return the matrix representing the linear transform.
- **FMatrix<double,2*N,2*N>& getMatrix ()**
  Return the matrix representing the linear transform.
- **int getOrder ()const**
  Return the order of the generators.
- **DragtFinnMap inverse ()**
  Compute inverse map.

  The first-order generator must be zero.
- **operator ()const**
  Convert to Taylor series map. Convert to Taylor series map.
- **void operator+= (const FTps<double,2*N>&)**
  Add to set of generators.
- **void operator+= (const FLieGenerator<double,N>&)**
  Add to generator of given order.
- **void operator-= (const FTps<double,2*N>&)**
  Subtract from set of generators.
- **void operator-= (const FLieGenerator<double,N>&)**
  Subtract from generator of given order.

- **const DragtFinnMap& operator= (const DragtFinnMap&)**

- **DragtFinnMap reverse ()**
  Compute inverse factorisation of map.

  The first-order generator must be zero.

- **~DragtFinnMap ()**

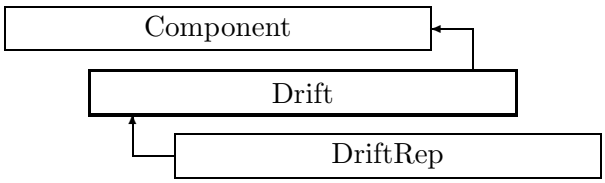

Figure 2.38: Inheritance for class Drift

2.44 class Drift

Interface for drift space.
Class Drift defines the abstract interface for a drift space.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Drift.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Drift (const string&)**
  Constructor with given name. (see Section 2.44.1)
- **Drift ()**
  (see Section 2.44.1)
- **Drift (const Drift&)**
  (see Section 2.44.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Drift. (see Section 2.44.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)

- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Drift ()**
  (see Section 2.44.1)

2.44.1 Detailed descriptions

Public members

- **Drift (const string&)**
  Constructor with given name.
- **Drift ()**
- **Drift (const Drift&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Drift.
- **virtual ~Drift ()**

Figure 2.39: Inheritance for class DriftRep

2.45 class DriftRep

Representation for a drift space.

Type:	Instantiable
Superclasses:	public Drift
Include file:	./BeamlineCore/DriftRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **DriftRep (const string&)**
  Constructor with given name. (see Section 2.45.1)
- **DriftRep ()**
  (see Section 2.45.1)
- **DriftRep (const DriftRep&)**
  (see Section 2.45.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Drift. (see Section 2.44.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.45.1)

- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.45.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual NullField& getField ()**
  Get field. (see Section 2.45.1)
- **virtual const NullField& getField ()const**
  Get field. (see Section 2.45.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.45.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.45.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.45.1)

- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.45.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)

- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ˜DriftRep ()**
  (see Section 2.45.1)

2.45.1 Detailed descriptions

Public members

- **DriftRep (const string&)**
  Constructor with given name.
- **DriftRep ()**
- **DriftRep (const DriftRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.

- **virtual NullField& getField ()**
  Get field.

  Version for non-constant object.

- **virtual const NullField& getField ()const**
  Get field.

  Version for constant object.

- **virtual StraightGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual ~DriftRep ()**

2.46 class DynamicFixedPoint

Dynamic fix point of a map.
Class DynamicFixedPoint represents the dynamic fixed point of a truncated power series map.

Type:	Instantiable
Include file:	./Algebra/DynamicFixedPoint.hh

Synopsis (including inherited members)

Public members

- **DynamicFixedPoint (const VpsInvMap<double>&)**
  Constructor. (see Section 2.46.1)
- **DynamicFixedPoint ()**
  (see Section 2.46.1)
- **DynamicFixedPoint (const DynamicFixedPoint&)**
  (see Section 2.46.1)
- **const Vector<double>& getFixedPoint ()const**
  Get fixed point transform. (see Section 2.46.1)
- **const VpsInvMap<double>& getFixedPointMap ()const**
  Get map around fixed point. (see Section 2.46.1)
- **~DynamicFixedPoint ()**
  (see Section 2.46.1)

2.46.1 Detailed descriptions

Public members

- **DynamicFixedPoint (const VpsInvMap<double>&)**
  Constructor.

  Find the fixed point of "map".
- **DynamicFixedPoint ()**
- **DynamicFixedPoint (const DynamicFixedPoint&)**
- **const Vector<double>& getFixedPoint ()const**
  Get fixed point transform.

  Return the fixed point as a vector of phase space coordinates.
- **const VpsInvMap<double>& getFixedPointMap ()const**
  Get map around fixed point.

  Return the map around the computed fixed point.
- **~DynamicFixedPoint ()**

2.47 class EBVectors

A representation of an electromagnetic field.
Contains both an electric and a magnetic field vector. These represent the instantaneous electromagnetic field in a given point.

Type:	Instantiable
Include file:	./Fields/EMField.hh

Synopsis (including inherited members)

Public members

- **EBVectors (const EVector&,const BVector&)**
  Constructor. (see Section 2.47.1)
- **BVector getB ()const**
  Get field. (see Section 2.47.1)
- **double getBx ()const**
  Get component. (see Section 2.47.1)
- **double getBy ()const**
  Get component. (see Section 2.47.1)
- **double getBz ()const**
  Get component. (see Section 2.47.1)
- **EVector getE ()const**
  Get component. (see Section 2.47.1)
- **double getEx ()const**
  Get component. (see Section 2.47.1)
- **double getEy ()const**
  Get component. (see Section 2.47.1)
- **double getEz ()const**
  Get component. (see Section 2.47.1)

2.47.1 Detailed descriptions

Public members

- **EBVectors (const EVector&,const BVector&)**
  Constructor.

  Construct the field from the two field vectors **E** (in A/m) and **B** (in T).
- **BVector getB ()const**
  Get field.

  Return the magnetic field vector in T.

- **double getBx ()const**
  Get component.

  Return the x-component of the magnetic field in T.

- **double getBy ()const**
  Get component.

  Return the y-component of the magnetic field in T.

- **double getBz ()const**
  Get component.

  Return the z-component of the magnetic field in T.

- **EVector getE ()const**
  Get component.

  Return the electric field vector in A/m.

- **double getEx ()const**
  Get component.

  Return the x-component of the electric field in A/m.

- **double getEy ()const**
  Get component.

  Return the y-component of the electric field in A/m.

- **double getEz ()const**
  Get component.

  Return the z-component of the electric field in A/m.

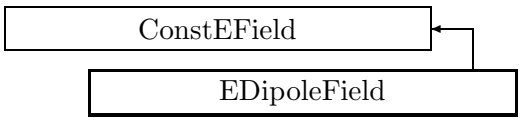

Figure 2.40: Inheritance for class EDipoleField

2.48 class EDipoleField

A static homogeneous electrostatic field in the (x,y)-plane.

Type:	Instantiable
Superclasses:	public ConstEField
Include file:	./Fields/EDipoleField.hh

Synopsis (including inherited members)

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **EDipoleField ()**
  Default constructor. (see Section 2.48.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.48.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.48.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)
- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)
- **virtual double getEx ()const**
  Get component. (see Section 2.48.1)
- **virtual double getEy ()const**
  Get component. (see Section 2.48.1)

- **virtual double getEz ()const**
  Get component. (see Section 2.30.1)
- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)
- **virtual void scale (double)**
  Scale the field. (see Section 2.48.1)
- **virtual void setEx (double)**
  Set component. (see Section 2.48.1)
- **virtual void setEy (double)**
  Set component. (see Section 2.48.1)
- **virtual void setEz (double)**
  Set component. (see Section 2.30.1)
- **virtual ~EDipoleField ()**
  (see Section 2.48.1)

2.48.1 Detailed descriptions

Public members

- **EDipoleField ()**
  Default constructor.

  Constructs a null field.
- **virtual EVector Efield (const Point3D&)const**
  Get field.

  Return the time-independent part of the electric field in point **P**.
- **virtual EVector Efield (const Point3D&,double)const**
  Get field.

  Return the electric field at time **t** in point **P**.
- **virtual double getEx ()const**
  Get component.

  Return the x-component of the electric field in A/m.
- **virtual double getEy ()const**
  Get component.

  Return the y-component of the electric field in A/m.
- **virtual void scale (double)**
  Scale the field.

  Multiply the field by **scalar**.

- **virtual void setEx (double)**
  Set component.

  Assign the x-component of the electric field in A/m.

- **virtual void setEy (double)**
  Set component.

  Assign the y-component of the electric field in A/m.

- **virtual ~EDipoleField ()**

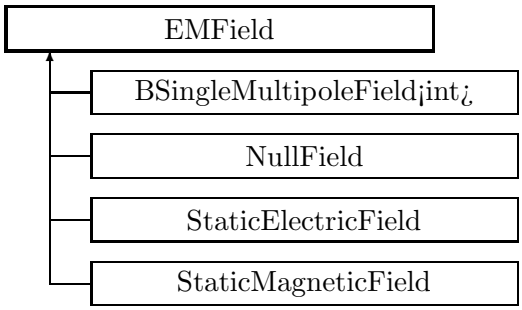

Figure 2.41: Inheritance for class EMField

2.49 class EMField

Abstract base class for electromagnetic fields.
This class represent a time-varying position dependent electromagnetic field. Derived classes include time-constant fields as well as spatially homogeneous fields.

Type:	abstract
Include file:	./Fields/EMField.hh

Synopsis (including inherited members)

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **EMField ()**
  Default constructor. (see Section 2.49.1)
- **EMField (const EMField&)**
  (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)

- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)
- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)
- **virtual void scale (double)**
  Scale the field. (see Section 2.49.1)
- **virtual ~EMField ()**
  (see Section 2.49.1)

2.49.1 Detailed descriptions

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field.

  Return the time-independent part of the magnetic field in point **P**. This default action returns a zero field.
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field.

  Return the magnetic field at time **t** in point **P**. This default action returns the static part BField(P).
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field.

  Return the static part of the field pair (E,B) in point **P**. This default action returns EBVectors(EField(P), BField(P)).
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field.

  Return the field pair (E,B) at time **t** in point **P**. This default action returns the static part EBfield(P).
- **EMField ()**
  Default constructor.

  Construct zero field.
- **EMField (const EMField&)**
- **virtual EVector Efield (const Point3D&)const**
  Get field.

  Return the time-independent part of the electric field in point **P**. This default action returns a zero field.

- **virtual EVector Efield (const Point3D&,double)const**
  Get field.

  Return the electric field at time **t** in point **P**. This default action returns the static part EField(P).

- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field.

- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field.

- **static const EVector ZeroEfield**
  The constant representing a zero electric field.

- **const EMField& operator= (const EMField&)**

- **virtual void scale (double)**
  Scale the field.

  Multiply the field by **scalar**. This method must be defined for each derived class.

- **virtual ~EMField ()**

2.50 class EVector

An electric field vector.

Type:	Instantiable
Include file:	./Fields/EMField.hh

Synopsis (including inherited members)

Public members

- **EVector (double,double,double)**
  Constructor. (see Section 2.50.1)
- **double getEx ()const**
  Get component. (see Section 2.50.1)
- **double getEy ()const**
  Get component. (see Section 2.50.1)
- **double getEz ()const**
  Get component. (see Section 2.50.1)
- **EVector operator* (double)const**
  Scale. (see Section 2.50.1)

2.50.1 Detailed descriptions

Public members

- **EVector (double,double,double)**
  Constructor.

  Construct the field vector from is components (Ex,Ey,Ez) in A/m.
- **double getEx ()const**
  Get component.

  Return the x-component of the field in A/m.
- **double getEy ()const**
  Get component.

  Return the y-component of the field in A/m.
- **double getEz ()const**
  Get component.

  Return the z-component of the field in A/m.
- **EVector operator* (double)const**
  Scale.

  Multiply the field vector by **scalar**.

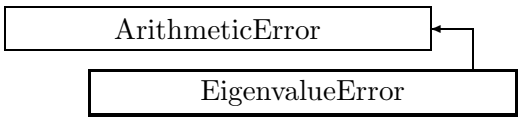

Figure 2.42: Inheritance for class EigenvalueError

2.51 class EigenvalueError

Eigenvalue error exception.
This exception is thrown, when an eigenvalue routine fails to compute all eigenvalues.

Type:	Instantiable
Superclasses:	public ArithmeticError
Include file:	./Utilities/EigenvalueError.hh

Synopsis (including inherited members)

Public members

- **EigenvalueError (const string&,const string&)**
  The usual constructor. (see Section 2.51.1)
- **EigenvalueError (const EigenvalueError&)**
  (see Section 2.51.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~EigenvalueError ()**
  (see Section 2.51.1)

2.51.1 Detailed descriptions

Public members

- **EigenvalueError (const string&,const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  msg the message string identifying the exception
- **EigenvalueError (const EigenvalueError&)**
- **virtual ~EigenvalueError ()**

2.52 class ElementBase

Interface for basic beam line object.
This class defines the abstract interface for all objects which can be contained in a beam line. ElementBase forms the base class for two distinct but related hierarchies of objects:

A set of concrete accelerator element classes, which compose the standard accelerator component library (SACL).

A second hierarchy which parallels the SACL, acting as container or wrapper-like objects. These latter classes are used to construct beam-lines composed of referenced "design" components, together with beam-line position dependent extrinsic state (e. g. errors). These wrapper objects are by default unique.

Also derived from ElementBase there is a class AlignWrapper, which can be used to store misalignment errors or deliberate displacements. Any element can have only one AlignWrapper and one FieldWrapper. To be processed in the correct order, the AlignWrapper must be the outermost.
`AlignWrapper --$>$ FieldWrapper --$>$ Element`
To ensure this structure, wrapper elements cannot be constructed directly, one should rather call one of:

makeAlignWrapper() make an align wrapper.

makeFieldWrapper() make a field wrapper.

makeWrappers() make both wrappers.

An existing wrapper can be removed by

removeAlignWrapper() remove align wrapper.

removeFieldWrapper() remove field wrapper.

Instances of the concrete classes for single elements are by default sharable. Instances of beam lines, wrappers and integrators are by default non-sharable, but they may be made sharable by a call to **makeSharable()**.
An ElementBase object can return two lengths, which may be different:

The arc length along the geometry.

The design length, often measured along a straight line.

Class ElementBase contains a map of name versus value for user-defined attributes (see file AbsBeamline/AttributeSet.hh). The map is primarily intended for processes that require algorithm-specific data in the accelerator model.
The class ElementBase has as base class the abstract class RCObject. Virtual derivation is used to make multiple inheritance possible for derived concrete classes. ElementBase implements three copy modes:

Copy by reference: Call RCObject::addReference() and use **this**.

Copy structure: use ElementBase::copyStructure(). During copying of the structure, all sharable items are re-used, while all non-sharable ones are cloned.

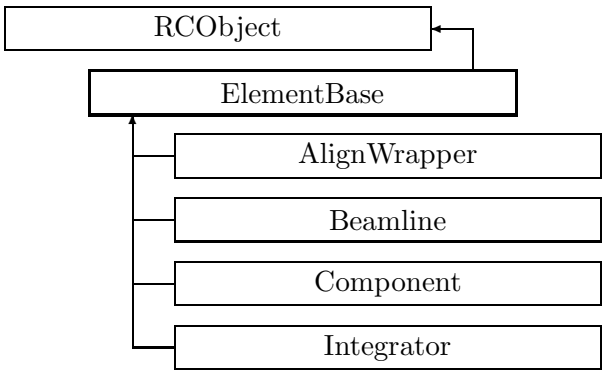

Figure 2.43: Inheritance for class ElementBase

Copy by cloning: use ElementBase::clone(). This returns a full deep copy.

Type:	abstract
Superclasses:	public RCObject
Include file:	./AbsBeamline/ElementBase.hh

Synopsis (including inherited members)

Public members

- **ElementBase (const string&)**
  Constructor with given name. (see Section 2.52.1)
- **ElementBase ()**
  (see Section 2.52.1)
- **ElementBase (const ElementBase&)**
  (see Section 2.52.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor. (see Section 2.52.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)

- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)

- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~ElementBase ()**
  (see Section 2.52.1)

Protected members

- **RCObject& operator= (const RCObject&)**
  (see Section 2.126.1)
- **mutable bool shareFlag**
  (see Section 2.52.1)

2.52.1 Detailed descriptions

Public members

- **ElementBase (const string&)**
  Constructor with given name.
- **ElementBase ()**
- **ElementBase (const ElementBase&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor.

  This method must be overridden by derived classes. It should call the method of the visitor corresponding to the element class. If any error occurs, this method throws an exception.
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual ElementBase* copyStructure ()**
  Make a structural copy.

  Return a fresh copy of any beam line structure is made, but sharable elements remain shared.
- **virtual double getArcLength ()const**
  Get arc length.

  Return the entire arc length measured along the design orbit
- **virtual double getAttribute (const string&)const**
  Get attribute value.

  If the attribute does not exist, return zero.
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.

- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel.

  This method constructs a Channel permitting read-only access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.

- **virtual double getElementLength ()const**
  Get design length.

  Return the design length defined by the geometry. This may be the arc length or the straight length.

- **virtual double getEntrance ()const**
  Get entrance position.

  Return the arc length from the origin to the entrance of the element (entrance <= 0)

- **virtual Euclid3D getEntranceFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getEntrance()). Return the transform of the local coordinate system from the origin to the entrance of the element.

- **virtual Euclid3D getEntrancePatch ()const**
  Get patch.

  Returns the entrance patch (transformation) which is used to transform the global geometry to the local geometry for a misaligned element at its entrance. The default behaviour returns identity transformation. This function should be overridden by derived concrete classes which model complex geometries.

- **virtual double getExit ()const**
  Get exit position.

  Return the arc length from the origin to the exit of the element (exit >= 0)

- **virtual Euclid3D getExitFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getExit()). Return the transform of the local coordinate system from the origin to the exit of the element.

- **virtual Euclid3D getExitPatch ()const**
  Get patch.

  Returns the entrance patch (transformation) which is used to transform the local geometry to the global geometry for a misaligned element at its exit. The default behaviour returns identity transformation. This function should be overridden by derived concrete classes which model complex geometries.

- **virtual Geometry& getGeometry ()**
  Get geometry.

  Return the element geometry. Version for non-constant object.

- **virtual const Geometry& getGeometry ()const**
  Get geometry.

  Return the element geometry Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual const string& getName ()const**
  Get element name.

- **virtual double getOrigin ()const**
  Get origin position.

  Return the arc length from the entrance to the origin of the element (origin >= 0)

- **virtual Euclid3D getTotalTransform ()const**
  Get transform.

  Equivalent to getTransform(getEntrance(), getExit()). Return the transform of the local coordinate system from the entrance to the exit of the element.

- **virtual Euclid3D getTransform (double,double)const**
  Get transform.

  Return the transform of the local coordinate system from the position **fromS** to the position **toS**.

- **virtual Euclid3D getTransform (double)const**
  Get transform.

  Equivalent to getTransform(0.0, s). Return the transform of the local coordinate system from the origin and **s**.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute.

  If the attribute exists, return true, otherwise false.

- **bool isSharable ()const**
  Test if the element can be shared.

- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment.

  Build an AlignWrapper pointing to the element and return a pointer to that wrapper. If the element cannot be misaligned, or already has an AlignWrapper, return a pointer to the element. Wrappers are non-sharable, unless otherwise defined.

- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors.

  Build a FieldWrapper pointing to the element and return a pointer to that wrapper. If the element cannot have field errors, or already has a FieldWrapper, return a pointer to the element. Wrappers are non-sharable, unless otherwise defined.

- **virtual void makeSharable ()**
  Set sharable flag.

  The whole structure depending on **this** is marked as sharable. After this call a **copyStructure()** call reuses the element.

- **virtual ElementBase* makeWrappers ()**
  Allow errors.

  Equivalent to the calls

  ```
  makeFieldWrapper()->$makeAlignWrapper()
  ```

  . Wrappers are non-sharable, unless otherwise defined.

- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper.

  Remove the align wrapper.

- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper.

  Remove the align wrapper.

- **virtual void removeAttribute (const string&)**
  Remove an existing attribute.

- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper.

  Remove the field wrapper.

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper.

  Remove the field wrapper for constant object.

- **virtual ElementBase* removeWrappers ()**
  Return the design element.

  Return **this**, if the element is not wrapped.

- **virtual const ElementBase* removeWrappers ()const**
  Return the design element.

  Return **this**, if the element is not wrapped.

- **virtual void setAttribute (const string&,double)**
  Set value of an attribute.

- **virtual void setElementLength (double)**
  Set design length.

  Set the design length defined by the geometry. This may be the arc length or the straight length.

- **virtual void setName (const string&)**
  Set element name.
- **bool update (const AttributeSet&)**
  Update element.

  This method stores all attributes contained in the AttributeSet to "*this". The return value **true** indicates success.
- **virtual ~ElementBase ()**

Protected members

- **mutable bool shareFlag**

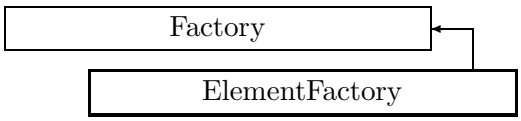

Figure 2.44: Inheritance for class ElementFactory

2.53 class ElementFactory

Concrete factory class for CLASSIC elements.
Defines the beamline element creation via the factory pattern. When the factory is constructed, empty elements are first created and stored. p With the makeElement() method, these elements can be cloned and then filled in with from an ElementImage. The factory also implements an element repository which can store beam lines.

Type:	Instantiable
Superclasses:	public Factory
Include file:	./Construction/ElementFactory.hh

Synopsis (including inherited members)

Public members

- **ElementFactory ()**
  Default constructor. (see Section 2.53.1)
- **typedef std::map<string,ElementBase*,std::less<string> > MapType**
  (see Section 2.53.1)
- **virtual bool define (ElementBase*)**
  Define a new element. (see Section 2.53.1)
- **virtual void erase (const string&)**
  Erase element by name. (see Section 2.53.1)
- **virtual ElementBase* find (const string&)const**
  Find element by name. (see Section 2.53.1)
- **virtual ElementBase* makeElement (const string&,const string&,const AttributeSet&)**
  Make new element. (see Section 2.53.1)
- **virtual bool storeElement (ElementBase*)**
  Define a new element. (see Section 2.53.1)
- **virtual ~ElementFactory ()**
  (see Section 2.53.1)

2.53.1 Detailed descriptions

Public members

- **ElementFactory ()**
  Default constructor.

Fills the repository with all standard element definitions.

- **typedef std::map<string,ElementBase*,std::less<string> > MapType**

- **virtual bool define (ElementBase*)**
  Define a new element.

  The element **newElement** is linked to the repository. If an element with the same name exists already, replacement is rejected, and **newElement** is deleted.

- **virtual void erase (const string&)**
  Erase element by name.

  If there is no element with the given **name**, the request is ignored.

- **virtual ElementBase* find (const string&)const**
  Find element by name.

  If an element with the name **name** exists, return a pointer to this element, otherwise return NULL.

- **virtual ElementBase* makeElement (const string&,const string&,const AttributeSet&)**
  Make new element.

  Create a new element with the type **type**, the name **name** and the attributes in **set**. If an element with the name **name** already exists, it is replaced.

- **virtual bool storeElement (ElementBase*)**
  Define a new element.

  The element **newElement** is linked to the repository. If an element with the same name exists already, it is replaced.

- **virtual ~ElementFactory ()**

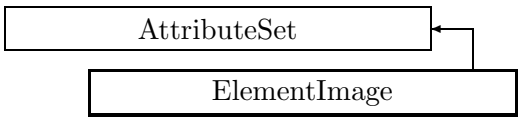

Figure 2.45: Inheritance for class ElementImage

2.54 class ElementImage

An image of an element.
Class ElementImage implements an image of an element. It contains two strings, the name and the type of the element, and a map of name versus value for all attributes of the element.

Type:	Instantiable
Superclasses:	public AttributeSet
Include file:	./AbsBeamline/ElementImage.hh

Synopsis (including inherited members)

Public members

- **ElementImage (const string&,const string&,const AttributeSet&)**
  Constructor. (see Section 2.54.1)
- **ElementImage ()**
  (see Section 2.54.1)
- **ElementImage (const ElementImage&)**
  (see Section 2.54.1)
- **typedef std::map<string,double,std::less<string> > NameMap**
  A map of name versus value. (see Section 2.10.1)
- **const_iterator begin ()const**
  Iterator accessing first member. (see Section 2.10.1)
- **typedef NameMap::const_iterator const_iterator**
  An iterator for a map of name versus value. (see Section 2.10.1)
- **const_iterator end ()const**
  Iterator marking the end of the list. (see Section 2.10.1)
- **double getAttribute (const string&)const**
  Get attribute value. (see Section 2.10.1)
- **Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.10.1)
- **const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.10.1)
- **const string& getName ()const**
  Get element name. (see Section 2.54.1)

- **const string& getType ()const**
  Get element type string. (see Section 2.54.1)
- **bool hasAttribute (const string&)const**
  Test for presence of an attribute. (see Section 2.10.1)
- **const ElementImage& operator= (const ElementImage&)**
  Assignment operator. (see Section 2.54.1)
- **const AttributeSet& operator= (const AttributeSet&)**
  (see Section 2.10.1)
- **void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.10.1)
- **void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.10.1)
- **void setName (const string&)**
  Set element name. (see Section 2.54.1)
- **void setType (const string&)**
  Set element type string. (see Section 2.54.1)
- **virtual ~ElementImage ()**
  (see Section 2.54.1)

2.54.1 Detailed descriptions

Public members

- **ElementImage (const string&,const string&,const AttributeSet&)**
  Constructor.

  This constructor takes the **name** and **type** strings as arguments, as well as an AttributeSet mapping attribute names to values.
- **ElementImage ()**
- **ElementImage (const ElementImage&)**
- **const string& getName ()const**
  Get element name.
- **const string& getType ()const**
  Get element type string.
- **const ElementImage& operator= (const ElementImage&)**
  Assignment operator.
- **void setName (const string&)**
  Set element name.

- **void setType (const string&)**
  Set element type string.
- **virtual ~ElementImage ()**

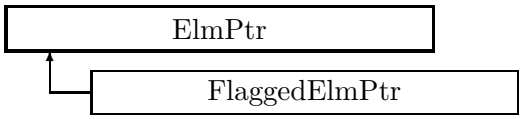

Figure 2.46: Inheritance for class ElmPtr

2.55 class ElmPtr

A section of a beam line.
A beam line is built as a list of ElmPtr.

Type:	Instantiable
Include file:	./Beamlines/ElmPtr.hh

Synopsis (including inherited members)

Public members

- **ElmPtr ()**
  (see Section 2.55.1)
- **ElmPtr (const ElmPtr&)**
  (see Section 2.55.1)
- **ElmPtr (ElementBase*)**
  (see Section 2.55.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor. (see Section 2.55.1)
- **inline ElementBase* getElement ()const**
  Get the element pointer. (see Section 2.55.1)
- **inline void setElement (ElementBase*)**
  Set the element pointer. (see Section 2.55.1)
- **virtual ~ElmPtr ()**
  (see Section 2.55.1)

Protected members

- **Pointer<ElementBase> itsElement**
  (see Section 2.55.1)

2.55.1 Detailed descriptions

Public members

- **ElmPtr ()**

- **ElmPtr (const ElmPtr&)**

- **ElmPtr (ElementBase*)**

- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor.

  If any error occurs, this method throws an exception.

- **inline ElementBase* getElement ()const**
  Get the element pointer.

- **inline void setElement (ElementBase*)**
  Set the element pointer.

- **virtual ~ElmPtr ()**

Protected members

- **Pointer<ElementBase> itsElement**

2.56 class Euclid3D

Displacement and rotation in space.
An arbitrary 3-dimension translation and rotation. The translation is a 3-dimensional vector, the rotation is represented as a 3-dimensional orthonormal matrix. P If we think of an Euler3D object as an operator E which transforms a point in the super-frame X to the local frame X': P X' = E(X), P then we can define a dot product of two successive operations in the following way: Given two Euclid3D objects (operators) E1 and E2, we can define P E3 = E2 . E1 P using P X"= E3(X) = E2(E1(X)). P The definition of a multiplicative inverse naturally follows as P E . inv(E) = I. P If R represents the 3-d rotation and X the vector specified by (x,y,z), then a transformation of a point P in the super-frame by the Euclid3D is given by P P -> R . P + X. P For the representation of rotations, see Rotation3D. The copy constructor, destructor, and assignment operator generated by the compiler perform the correct operation. For speed reasons they are not implemented. An alternate representation for the rotation is given by a vector pointing in the direction of the axis of rotation, whose sense is given by the right-hand rule, and whose length is equal to the angle of rotation.

Type:	Instantiable
Include file:	./BeamlineGeometry/Euclid3D.hh

Synopsis (including inherited members)

Public members

- **Euclid3D ()**
  Default constructor. (see Section 2.56.1)
- **Euclid3D (const Vector3D&,const Rotation3D&)**
  Constructor/ (see Section 2.56.1)
- **Euclid3D (double,double,double,double,double,double)**
  Constructor. (see Section 2.56.1)
- **double M (int,int)const**
  Get component. (see Section 2.56.1)
- **static Euclid3D XRotation (double)**
  Make rotation. (see Section 2.56.1)
- **static Euclid3D YRotation (double)**
  Make rotation. (see Section 2.56.1)
- **static Euclid3D ZRotation (double)**
  Make rotation. (see Section 2.56.1)
- **Euclid3D dot (const Euclid3D&)const**
  Dot product. (see Section 2.56.1)
- **const Euclid3D& dotBy (const Euclid3D&)**
  Dot product with assign. (see Section 2.56.1)
- **void getAll (double&,double&,double&,double&,double&,double&)const**
  Unpack. (see Section 2.56.1)

- **const Rotation3D& getRotation ()const**
  Get rotation. (see Section 2.56.1)
- **const Vector3D& getVector ()const**
  Get displacement. (see Section 2.56.1)
- **double getX ()const**
  Get displacement. (see Section 2.56.1)
- **double getY ()const**
  Get displacement. (see Section 2.56.1)
- **double getZ ()const**
  Get displacement. (see Section 2.56.1)
- **static Euclid3D identity ()**
  Make identity. (see Section 2.56.1)
- **Euclid3D inverse ()const**
  Inverse. (see Section 2.56.1)
- **bool isIdentity ()const**
  Test for identity. (see Section 2.56.1)
- **bool isPureTranslation ()const**
  Test for translation. (see Section 2.56.1)
- **bool isPureXRotation ()const**
  Test for rotation. (see Section 2.56.1)
- **bool isPureYRotation ()const**
  Test for rotation. (see Section 2.56.1)
- **bool isPureZRotation ()const**
  Test for rotation. (see Section 2.56.1)
- **bool operator!= (const Euclid3D&)const**
  (see Section 2.56.1)
- **Euclid3D operator* (const Euclid3D&)const**
  Dot product. (see Section 2.56.1)
- **const Euclid3D& operator*= (const Euclid3D&)**
  Dot product with assign. (see Section 2.56.1)
- **bool operator== (const Euclid3D&)const**
  (see Section 2.56.1)
- **void setDisplacement (const Vector3D&)**
  Set displacement. (see Section 2.56.1)
- **void setRotation (const Rotation3D&)**
  Set rotation. (see Section 2.56.1)

- **void setX (double)**
  Set displacement. (see Section 2.56.1)
- **void setY (double)**
  Set displacement. (see Section 2.56.1)
- **void setZ (double)**
  Set displacement. (see Section 2.56.1)
- **static Euclid3D translation (double,double,double)**
  Make translation. (see Section 2.56.1)

2.56.1 Detailed descriptions

Public members

- **Euclid3D ()**
  Default constructor.

  Construct identity.
- **Euclid3D (const Vector3D&,const Rotation3D&)**
  Constructor/

  Use displacement **V** and rotation **R**.
- **Euclid3D (double,double,double,double,double,double)**
  Constructor.

  Use displacement (x,y,z) and rotation (vx,vy,vz).
- **double M (int,int)const**
  Get component.

  Return the component (row,col) of the rotation matrix.
- **static Euclid3D XRotation (double)**
  Make rotation.

  Construct pure rotation by **angle** around the x-axis;
- **static Euclid3D YRotation (double)**
  Make rotation.

  Construct pure rotation by **angle** around the y-axis;
- **static Euclid3D ZRotation (double)**
  Make rotation.

  Construct pure rotation by **angle** around the z-axis;
- **Euclid3D dot (const Euclid3D&)const**
  Dot product.

  Return composition of **this** with **rhs**.

- **const Euclid3D& dotBy (const Euclid3D&)**
  Dot product with assign.

  Return composition of **this** with **rhs**.

- **void getAll (double&,double&,double&,double&,double&,double&)const**
  Unpack.

  Return the displacement in (x,y,z) and the rotation axis in (vx,vy,vz).

- **const Rotation3D& getRotation ()const**
  Get rotation.

  Return rotation as an orthogonal matrix.

- **const Vector3D& getVector ()const**
  Get displacement.

  Return displacement as a vector.

- **double getX ()const**
  Get displacement.

  Return x-component of displacement.

- **double getY ()const**
  Get displacement.

  Return x-component of displacement.

- **double getZ ()const**
  Get displacement.

  Return x-component of displacement.

- **static Euclid3D identity ()**
  Make identity.

- **Euclid3D inverse ()const**
  Inverse.

- **bool isIdentity ()const**
  Test for identity.

- **bool isPureTranslation ()const**
  Test for translation.

  Return true, if **this** is a pure translation.

- **bool isPureXRotation ()const**
  Test for rotation.

  Return true, if **this** is a pure x-rotation.

- **bool isPureYRotation ()const**
  Test for rotation.

  Return true, if **this** is a pure y-rotation.

- **bool isPureZRotation ()const**
  Test for rotation.

  Return true, if **this** is a pure z-rotation.

- **bool operator!= (const Euclid3D&)const**

- **Euclid3D operator* (const Euclid3D&)const**
  Dot product.

  Return composition of **this** with **rhs**.

- **const Euclid3D& operator*= (const Euclid3D&)**
  Dot product with assign.

- **bool operator== (const Euclid3D&)const**

- **void setDisplacement (const Vector3D&)**
  Set displacement.

  Assign displacement as a vector.

- **void setRotation (const Rotation3D&)**
  Set rotation.

  Assign rotation as an orthogonal matrix.

- **void setX (double)**
  Set displacement.

  Assign x-component of displacement.

- **void setY (double)**
  Set displacement.

  Assign y-component of displacement.

- **void setZ (double)**
  Set displacement.

  Assign z-component of displacement.

- **static Euclid3D translation (double,double,double)**
  Make translation.

  Use displacement (x,y,z).

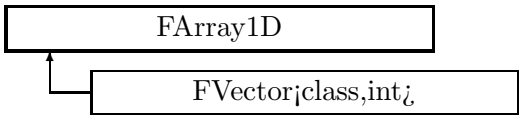

Figure 2.47: Inheritance for class FArray1D

2.57 template class FArray1D <class,int>

A templated representation for one-dimensional arrays.
This version has fixed dimensions. It implements array access, but contains no arithmetic operations. The destructor generated by the compiler performs the correct operation. For speed reasons it is not implemented.

Type:	Instantiable
Include file:	./FixedAlgebra/FArray1D.hh

Synopsis (including inherited members)

Public members

- **FArray1D ()**
  Default constructor. (see Section 2.57.1)
- **FArray1D (const T&)**
  Constructor. (see Section 2.57.1)
- **FArray1D (const FArray1D&)**
  Copy constructor. (see Section 2.57.1)
- **iterator begin ()**
  Get iterator pointing to beginning of array. (see Section 2.57.1)
- **const_iterator begin ()const**
  Get iterator pointing to beginning of array. (see Section 2.57.1)
- **typedef const T* const_iterator**
  Iterator for constant array. (see Section 2.57.1)
- **iterator end ()**
  Get iterator pointing past end of array. (see Section 2.57.1)
- **const_iterator end ()const**
  Get iterator pointing past end of array. (see Section 2.57.1)
- **typedef T* iterator**
  Iterator for the array. (see Section 2.57.1)
- **T& operator() (int)**
  Get element. (see Section 2.57.1)
- **const T& operator() (int)const**
  Get element. (see Section 2.57.1)

- **const FArray1D& operator= (const FArray1D&)**
  Assignment. (see Section 2.57.1)
- **T& operator[] (int)**
  Get element. (see Section 2.57.1)
- **const T& operator[] (int)const**
  Get element. (see Section 2.57.1)
- **int size ()const**
  Get array size. (see Section 2.57.1)
- **typedef T value_type**
  The value type of this array. (see Section 2.57.1)

Protected members

- **T data [N]**
  (see Section 2.57.1)

2.57.1 Detailed descriptions

Public members

- **FArray1D ()**
  Default constructor.

  Constructs a zero array.
- **FArray1D (const T&)**
  Constructor.

  Set all array elements to **t**.
- **FArray1D (const FArray1D&)**
  Copy constructor.
- **iterator begin ()**
  Get iterator pointing to beginning of array.

  Version for non-constant array.
- **const_iterator begin ()const**
  Get iterator pointing to beginning of array.

  Version for constant array.
- **typedef const T* const_iterator**
  Iterator for constant array.
- **iterator end ()**
  Get iterator pointing past end of array.

  Version for non-constant array.

- **const_iterator end ()const**
  Get iterator pointing past end of array.

  Version for constant array.

- **typedef T* iterator**
  Iterator for the array.

- **T& operator() (int)**
  Get element.

  Return a reference to element **n**. Throw RangeError, if **n** is out of range.

- **const T& operator() (int)const**
  Get element.

  Return a constant reference to element **n**. Throw RangeError, if **n** is out of range.

- **const FArray1D& operator= (const FArray1D&)**
  Assignment.

- **T& operator[] (int)**
  Get element.

  Return a reference to element **n**. Result is undefined, if **n** is out of range.

- **const T& operator[] (int)const**
  Get element.

  Return a reference to element **n**. Result is undefined, if **n** is out of range.

- **int size ()const**
  Get array size.

- **typedef T value_type**
  The value type of this array.

Protected members

- **T data [N]**

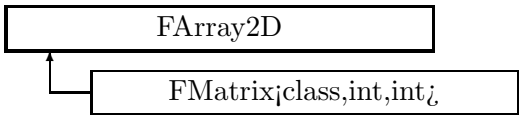

Figure 2.48: Inheritance for class FArray2D

2.58 template class FArray2D <class,int,int>

A templated representation for 2-dimensional arrays.
This version has fixed dimensions. It implements storage management and access, but contains no arithmetic operations. The destructor generated by the compiler performs the correct operation. For speed reasons it is not implemented.

Type:	Instantiable
Include file:	./FixedAlgebra/FArray2D.hh

Synopsis (including inherited members)

Public members

- **FArray2D ()**
  Default constructor. (see Section 2.58.1)
- **FArray2D (const T&)**
  Constructor. (see Section 2.58.1)
- **FArray2D (const FArray2D&)**
  Copy constructor. (see Section 2.58.1)
- **iterator begin ()**
  Get beginning of data. (see Section 2.58.1)
- **const_iterator begin ()const**
  Get beginning of data. (see Section 2.58.1)
- **col_iterator col_begin (int)**
  Get column iterator. (see Section 2.58.1)
- **const_col_iterator col_begin (int)const**
  Get column iterator. (see Section 2.58.1)
- **col_iterator col_end (int)**
  Get column iterator. (see Section 2.58.1)
- **const_col_iterator col_end (int)const**
  Get column iterator. (see Section 2.58.1)
- **typedef FSlice<T,N> col_iterator**
  Iterator for access by columns. (see Section 2.58.1)
- **typedef FConstSlice<T,N> const_col_iterator**
  Iterator for access by columns. (see Section 2.58.1)

- **typedef const T* const_iterator**
  Iterator for constant array. (see Section 2.58.1)
- **typedef const T* const_row_iterator**
  Iterator for access by rows. (see Section 2.58.1)
- **iterator end ()**
  Get pointer past end of data. (see Section 2.58.1)
- **const_iterator end ()const**
  Get pointer past end of data. (see Section 2.58.1)
- **void getColumn (FArray1D<T,M>&,int)const**
  Fetch column. (see Section 2.58.1)
- **void getRow (FArray1D<T,N>&,int)const**
  Fetch row. (see Section 2.58.1)
- **typedef T* iterator**
  Iterator for the array. (see Section 2.58.1)
- **int ncols ()const**
  Get number of columns. (see Section 2.58.1)
- **int nrows ()const**
  Get number of rows. (see Section 2.58.1)
- **T& operator() (int,int)**
  Get element. (see Section 2.58.1)
- **const T& operator() (int,int)const**
  Get element. (see Section 2.58.1)
- **const FArray2D<T,M,N>& operator= (const FArray2D<T,M,N>&)**
  Assignment. (see Section 2.58.1)
- **row_iterator operator[] (int)**
  Get row iterator. (see Section 2.58.1)
- **const_row_iterator operator[] (int)const**
  Get row iterator. (see Section 2.58.1)
- **void putColumn (const FArray1D<T,M>&,int)**
  Store column. (see Section 2.58.1)
- **void putRow (const FArray1D<T,N>&,int)**
  Store row. (see Section 2.58.1)
- **row_iterator row_begin (int)**
  Get row iterator. (see Section 2.58.1)
- **const_row_iterator row_begin (int)const**
  Get row iterator. (see Section 2.58.1)

- **row_iterator row_end (int)**
  Get row iterator. (see Section 2.58.1)
- **const_row_iterator row_end (int)const**
  Get row iterator. (see Section 2.58.1)
- **typedef T* row_iterator**
  Iterator for access by rows. (see Section 2.58.1)
- **int size ()const**
  Get total size. (see Section 2.58.1)
- **void swapColumns (int,int)**
  Exchange columns. (see Section 2.58.1)
- **void swapRows (int,int)**
  Exchange rows. (see Section 2.58.1)
- **typedef T value_type**
  The value type of the array. (see Section 2.58.1)

Protected members

- **static const int SIZE**
  (see Section 2.58.1)
- **T data [M*N]**
  (see Section 2.58.1)

2.58.1 Detailed descriptions

Public members

- **FArray2D ()**
  Default constructor.

  Constructs zero array.
- **FArray2D (const T&)**
  Constructor.

  Set all array elements to **t**.
- **FArray2D (const FArray2D&)**
  Copy constructor.
- **iterator begin ()**
  Get beginning of data.

  The array is interpreted as a one-dimensional array. Version for non-constant array.
- **const_iterator begin ()const**
  Get beginning of data.

  The array is interpreted as a one-dimensional array. Version for constant array.

- **col_iterator col_begin (int)**
  Get column iterator.

  Return pointer to beginning of column **c**. Throw RangeError if **c** is out of range.

- **const_col_iterator col_begin (int)const**
  Get column iterator.

  Return pointer to beginning of column **c**. Throw RangeError if **c** is out of range.

- **col_iterator col_end (int)**
  Get column iterator.

  Return pointer past end of column **c**. Throw RangeError if **c** is out of range.

- **const_col_iterator col_end (int)const**
  Get column iterator.

  Return pointer past end of column **c**. Throw RangeError if **c** is out of range.

- **typedef FSlice<T,N> col_iterator**
  Iterator for access by columns.

- **typedef FConstSlice<T,N> const_col_iterator**
  Iterator for access by columns.

- **typedef const T* const_iterator**
  Iterator for constant array.

- **typedef const T* const_row_iterator**
  Iterator for access by rows.

- **iterator end ()**
  Get pointer past end of data.

  The array is interpreted as a one-dimensional array. Version for non-constant array.

- **const_iterator end ()const**
  Get pointer past end of data.

  The array is interpreted as a one-dimensional array. Version for constant array.

- **void getColumn (FArray1D<T,M>&,int)const**
  Fetch column.

  Copy column **c** into **toArray**. Throw RangeError if **c** is out of range.

- **void getRow (FArray1D<T,N>&,int)const**
  Fetch row.

  Copy row **r** into **toArray**.

- **typedef T* iterator**
  Iterator for the array.

- **int ncols ()const**
  Get number of columns.

- **int nrows ()const**
  Get number of rows.

- **T& operator() (int,int)**
  Get element.

  Return a reference to the element in row **r** and column **c**.

- **const T& operator() (int,int)const**
  Get element.

  Return a constant reference to the element in row **r** and column **c**.

- **const FArray2D<T,M,N>& operator= (const FArray2D<T,M,N>&)**
  Assignment.

- **row_iterator operator[] (int)**
  Get row iterator.

  Return pointer to beginning of row **r**. Result is undefined, if **r** is out of range.

- **const_row_iterator operator[] (int)const**
  Get row iterator.

  Return pointer to beginning of row **r**. Result is undefined, if **r** is out of range.

- **void putColumn (const FArray1D<T,M>&,int)**
  Store column.

  Copy **fromArray** to column **c**. Throw RangeError if **c** is out of range.

- **void putRow (const FArray1D<T,N>&,int)**
  Store row.

  Copy **fromArray** to row **r**. Throw RangeError if **r** is out of range.

- **row_iterator row_begin (int)**
  Get row iterator.

  Return pointer to beginning of row **r**. Throw RangeError, if **r** is out of range.

- **const_row_iterator row_begin (int)const**
  Get row iterator.

  Return pointer to beginning of constant row **r**. Throw RangeError, if **r** is out of range.

- **row_iterator row_end (int)**
  Get row iterator.

  Return pointer past end of row **r**. Throw RangeError, if **r** is out of range.

- **const_row_iterator row_end (int)const**
  Get row iterator.

  Return pointer past end of constant row **r**. Throw RangeError, if **r** is out of range.

- **typedef T* row_iterator**
  Iterator for access by rows.

- **int size ()const**
  Get total size.

- **void swapColumns (int,int)**
  Exchange columns.

  Exchange columns **c1** and **c2**. Throw RangeError, if either index is out of range.

- **void swapRows (int,int)**
  Exchange rows.

  Exchange rows **r1** and **r2**. Throw RangeError, if either index is out of range.

- **typedef T value_type**
  The value type of the array.

Protected members

- **static const int SIZE**

- **T data [M*N]**

2.59 template class FComplexEigen <int>

Eigenvalues and eigenvectors for a complex general matrix.
Translated to FORTRAN by Burton S. Garbov, ANL Adapted March 1997 by F. Christoph Iselin, CERN, SL/AP Changed to template December 1998 by F. Christoph Iselin, CERN, SL/AP

Type:	Instantiable
Include file:	./FixedAlgebra/FComplexEigen.hh

Synopsis (including inherited members)

Public members

- **FComplexEigen (const FMatrix<complex<double>,N,N>&,bool)**
  Constructor. (see Section 2.59.1)
- **FComplexEigen ()**
  (see Section 2.59.1)
- **FComplexEigen (const FComplexEigen&)**
  (see Section 2.59.1)
- **const FVector<complex<double>,N>& eigenValues ()const**
  Get eigenvalues. (see Section 2.59.1)
- **const FMatrix<complex<double>,N,N>& eigenVectors ()const**
  Get eigenvectors. (see Section 2.59.1)
- **~FComplexEigen ()**
  (see Section 2.59.1)

2.59.1 Detailed descriptions

Public members

- **FComplexEigen (const FMatrix<complex<double>,N,N>&,bool)**
  Constructor.

  Find eigenvalues for matrix **M**. If **vec** is true, the eigenvectors are also computed.
- **FComplexEigen ()**
- **FComplexEigen (const FComplexEigen&)**
- **const FVector<complex<double>,N>& eigenValues ()const**
  Get eigenvalues.

  Return eigenvalues as complex vector.
- **const FMatrix<complex<double>,N,N>& eigenVectors ()const**
  Get eigenvectors.

  Return eigenvectors as a complex matrix.

- **~FComplexEigen ()**

2.60 template class FConstSlice <class,int>

Constant version of FSlice

Type:	Instantiable
Include file:	./FixedAlgebra/FSlice.hh

Synopsis (including inherited members)

Public members

- **FConstSlice (const T*)**
  Constructor from array and stride. (see Section 2.60.1)
- **FConstSlice (const FConstSlice&)**
  (see Section 2.60.1)
- **typedef ptrdiff_t difference_type**
  The pointer difference type. (see Section 2.60.1)
- **typedef std::random_access_iterator_tag iterator_category**
  The iterator tag, taken from the standard template library. (see Section 2.60.1)
- **bool operator!= (const FConstSlice&)const**
  (see Section 2.60.1)
- **const T& operator* ()const**
  Dereference. (see Section 2.60.1)
- **FConstSlice operator+ (ptrdiff_t)const**
  Add multiple stride. (see Section 2.60.1)
- **FConstSlice& operator++ ()**
  Increment (iterate forward). (see Section 2.60.1)
- **FConstSlice operator++ (int)**
  Increment (iterate forward). (see Section 2.60.1)
- **FConstSlice& operator+= (ptrdiff_t)**
  Add multiple stride and assign. (see Section 2.60.1)
- **FConstSlice operator- (ptrdiff_t)const**
  Subtract multiple stride. (see Section 2.60.1)
- **difference_type operator- (const FConstSlice&)const**
  Get pointer difference. (see Section 2.60.1)
- **FConstSlice& operator– ()**
  Decrement (iterate backward). (see Section 2.60.1)
- **FConstSlice operator– (int)**
  Decrement (iterate backward). (see Section 2.60.1)
- **FConstSlice& operator-= (ptrdiff_t)**
  Subtract multiple stride and assign. (see Section 2.60.1)

- **FConstSlice& operator= (const FConstSlice&)**
  (see Section 2.60.1)
- **bool operator== (const FConstSlice&)const**
  (see Section 2.60.1)
- **const T& operator[] (int)const**
  Delegate. (see Section 2.60.1)
- **typedef const T* pointer**
  The pointer type. (see Section 2.60.1)
- **typedef const T& reference**
  The reference type. (see Section 2.60.1)
- **typedef const T value_type**
  The value type. (see Section 2.60.1)

2.60.1 Detailed descriptions

Public members

- **FConstSlice (const T*)**
  Constructor from array and stride.
- **FConstSlice (const FConstSlice&)**
- **typedef ptrdiff_t difference_type**
  The pointer difference type.
- **typedef std::random_access_iterator_tag iterator_category**
  The iterator tag, taken from the standard template library.
- **bool operator!= (const FConstSlice&)const**
- **const T& operator* ()const**
  Dereference.
- **FConstSlice operator+ (ptrdiff_t)const**
  Add multiple stride.
- **FConstSlice& operator++ ()**
  Increment (iterate forward).
- **FConstSlice operator++ (int)**
  Increment (iterate forward).
- **FConstSlice& operator+= (ptrdiff_t)**
  Add multiple stride and assign.
- **FConstSlice operator- (ptrdiff_t)const**
  Subtract multiple stride.

- **difference_type operator- (const FConstSlice&)const**
  Get pointer difference.
- **FConstSlice& operator– ()**
  Decrement (iterate backward).
- **FConstSlice operator– (int)**
  Decrement (iterate backward).
- **FConstSlice& operator-= (ptrdiff_t)**
  Subtract multiple stride and assign.
- **FConstSlice& operator= (const FConstSlice&)**
- **bool operator== (const FConstSlice&)const**
- **const T& operator[] (int)const**
  Delegate.
- **typedef const T* pointer**
  The pointer type.
- **typedef const T& reference**
  The reference type.
- **typedef const T value_type**
  The value type.

2.61 template class FDoubleEigen <int>

Eigenvalues and eigenvectors for a real general matrix.
Translated to FORTRAN by Burton S. Garbow, ANL. Adapted March 1997 by F. Christoph Iselin, CERN, SL/AP. Changed December 1998 to template by F. Christoph Iselin, CERN, SL/AP.

Type:	Instantiable
Include file:	./FixedAlgebra/FDoubleEigen.hh

Synopsis (including inherited members)

Public members

- **FDoubleEigen (const FMatrix<double,N,N>&,bool)**
  Constructor. (see Section 2.61.1)
- **FDoubleEigen ()**
  (see Section 2.61.1)
- **FDoubleEigen (const FDoubleEigen&)**
  (see Section 2.61.1)
- **FVector<complex<double>,N> eigenValues ()const**
  Get eigenvalues. (see Section 2.61.1)
- **FMatrix<complex<double>,N,N> eigenVectors ()const**
  Get eigenvectors. (see Section 2.61.1)
- **FMatrix<double,N,N> packedEigenVectors ()const**
  Get eigenvectors. (see Section 2.61.1)
- **~FDoubleEigen ()**
  (see Section 2.61.1)

2.61.1 Detailed descriptions

Public members

- **FDoubleEigen (const FMatrix<double,N,N>&,bool)**
  Constructor.

  Find eigenvalues for matrix **M**. If **vec** is true, the eigenvectors are also computed.
- **FDoubleEigen ()**
- **FDoubleEigen (const FDoubleEigen&)**
- **FVector<complex<double>,N> eigenValues ()const**
  Get eigenvalues.

  Return eigenvalues as a complex vector.

- **FMatrix<complex<double>,N,N> eigenVectors ()const**
  Get eigenvectors.

  Return eigenvectors as a complex matrix.

- **FMatrix<double,N,N> packedEigenVectors ()const**
  Get eigenvectors.

  Return eigenvectors packed in a real matrix. Real eigenvectors appear a single column in the same position as the corresponding real eigenvalue. Complex eigenvalues occur in conjugate pairs, and the corresponding eigenvectors appear as two real columns containing the real and imaginary parts respectively.

- **~FDoubleEigen ()**

2.62 template class FDynamicFP <int>

Dynamic fixed point of a Truncated power series map.

Type:	Instantiable
Include file:	./FixedAlgebra/FDynamicFP.hh

Synopsis (including inherited members)

Public members

- **FDynamicFP (const FVps<double,N>&)**
  Constructor. (see Section 2.62.1)
- **FDynamicFP ()**
  (see Section 2.62.1)
- **FDynamicFP (const FDynamicFP&)**
  (see Section 2.62.1)
- **const FVector<double,N>& getFixedPoint ()const**
  Get the transformation to the fixed point. (see Section 2.62.1)
- **const FVps<double,N>& getFixedPointMap ()const**
  Get the map around the fixed point. (see Section 2.62.1)
- **~FDynamicFP ()**
  (see Section 2.62.1)

2.62.1 Detailed descriptions

Public members

- **FDynamicFP (const FVps<double,N>&)**
  Constructor.

  Find fixed point of **map**.
- **FDynamicFP ()**
- **FDynamicFP (const FDynamicFP&)**
- **const FVector<double,N>& getFixedPoint ()const**
  Get the transformation to the fixed point.
- **const FVps<double,N>& getFixedPointMap ()const**
  Get the map around the fixed point.
- **~FDynamicFP ()**

2.63 template class FLUMatrix <class,int>

A templated representation of a LU-decomposition.
Constructed from a FMatrix<T,N,N>, this class remembers the triangular decomposition and can return the back-substitution or the inverse. When FLUMatrix is used on a complex matrix, then the statement
`include std::abs;`
is required in the calling file, so as to inject the `abs()` function to the global namespace.

Type:	Instantiable
Include file:	./FixedAlgebra/FLUMatrix.hh

Synopsis (including inherited members)

Public members

- **FLUMatrix (const FMatrix<T,N,N>&)**
  Constructor. (see Section 2.63.1)
- **FLUMatrix ()**
  (see Section 2.63.1)
- **FLUMatrix (const FLUMatrix<T,N>&)**
  (see Section 2.63.1)
- **void backSubstitute (FVector<T,N>&)const**
  Back substitution. (see Section 2.63.1)
- **void backSubstitute (FMatrix<T,N,M>&)const**
  Back substitution. (see Section 2.63.1)
- **void backSubstitute (Iterator)const**
  (see Section 2.63.1)
- **FMatrix<T,N,N> inverse ()const**
  Get inverse. (see Section 2.63.1)
- **FLUMatrix<T,N>& operator= (const FLUMatrix&)**
  (see Section 2.63.1)
- **~FLUMatrix ()**
  (see Section 2.63.1)

2.63.1 Detailed descriptions

Public members

- **FLUMatrix (const FMatrix<T,N,N>&)**
  Constructor.

  Find triangel decomposition of the matrix **M**. Throw SingularMatrixError if **M** is singular.
- **FLUMatrix ()**

- **FLUMatrix (const FLUMatrix<T,N>&)**

- **void backSubstitute (FVector<T,N>&)const**
  Back substitution.

  Perform back substitution on the vector **B**. The new **B** is the old **B**, premultiplied by the inverse.

- **void backSubstitute (FMatrix<T,N,M>&)const**
  Back substitution.

  Perform back substitution on the matrix **M**. The new **M** is the old **M**, premultiplied by the inverse.

- **void backSubstitute (Iterator)const**

- **FMatrix<T,N,N> inverse ()const**
  Get inverse.

  Construct and return the inverse.

- **FLUMatrix<T,N>& operator= (const FLUMatrix&)**

- **~FLUMatrix ()**

2.64 template class FLieGenerator <class,int>

A representation for a homogeneous polynomial, used as a Lie generator.
The number of variables must be even, it is equal to 2*N, where N is a template parameter. The indices have the same range as the coefficients of the same order in a FTps<double,2*N> object.

Type:	Instantiable
Include file:	./FixedAlgebra/FLieGenerator.hh

Synopsis (including inherited members)

Public members

- **FLieGenerator (int)**
  Construct a zero generator of given order. (see Section 2.64.1)
- **FLieGenerator (const FTps<T,2*N>&,int)**
  Construct a zero generator of given order by extraction (see Section 2.64.1)
- **FLieGenerator ()**
  (see Section 2.64.1)
- **FLieGenerator (const FLieGenerator&)**
  (see Section 2.64.1)
- **inline T* begin ()**
  Get pointer to beginning of generator. (see Section 2.64.1)
- **inline const T* begin ()const**
  Get pointer to beginning of generator. (see Section 2.64.1)
- **void clear ()**
  Clear all coefficients. (see Section 2.64.1)
- **inline T* end ()**
  Get pointer past end of generator. (see Section 2.64.1)
- **inline const T* end ()const**
  Get pointer past end of generator. (see Section 2.64.1)
- **inline int getBottomIndex ()const**
  Return bottom index of this generator. (see Section 2.64.1)
- **inline int getOrder ()const**
  Return order of this generator. (see Section 2.64.1)
- **inline int getTopIndex ()const**
  Return top index of this generator. (see Section 2.64.1)
- **bool isZero ()const**
  Test for zero. (see Section 2.64.1)
- **FLieGenerator& operator*= (const T&)**
  Multiply by scalar and assign. (see Section 2.64.1)

- **FLieGenerator& operator+= (const FLieGenerator&)**
  Add vector and assign. (see Section 2.64.1)
- **FLieGenerator operator- ()const**
  Change sign of generator. (see Section 2.64.1)
- **FLieGenerator& operator-= (const FLieGenerator&)**
  Subtract vector and assign. (see Section 2.64.1)
- **FLieGenerator& operator/= (const T&)**
  Divide by scalar and assign. (see Section 2.64.1)
- **const FLieGenerator& operator= (const FLieGenerator&)**
  (see Section 2.64.1)
- **bool operator== (const FLieGenerator&)const**
  (see Section 2.64.1)
- **T& operator[] (int)**
  Get element. (see Section 2.64.1)
- **const T& operator[] (int)const**
  Get element. (see Section 2.64.1)
- **FLieGenerator scale (const FLieGenerator&)const**
  Scale monomial-wise. (see Section 2.64.1)
- **FLieGenerator<U,N> transform (const FMatrix<U,2*N,2*N>&)const**
  Substitute matrix in Lie generator. (see Section 2.64.1)
- **~FLieGenerator ()**
  (see Section 2.64.1)

2.64.1 Detailed descriptions

Public members

- **FLieGenerator (int)**
  Construct a zero generator of given order.
- **FLieGenerator (const FTps<T,2*N>&,int)**
  Construct a zero generator of given order by extraction

  from an FTps<T,2*N> object.
- **FLieGenerator ()**
- **FLieGenerator (const FLieGenerator&)**
- **inline T* begin ()**
  Get pointer to beginning of generator.

  Version for non-constant generator.

- **inline const T* begin ()const**
  Get pointer to beginning of generator.

  Version for constant generator.

- **void clear ()**
  Clear all coefficients.

- **inline T* end ()**
  Get pointer past end of generator.

  Version for non-constant generator.

- **inline const T* end ()const**
  Get pointer past end of generator.

  Version for constant generator.

- **inline int getBottomIndex ()const**
  Return bottom index of this generator.

- **inline int getOrder ()const**
  Return order of this generator.

- **inline int getTopIndex ()const**
  Return top index of this generator.

- **bool isZero ()const**
  Test for zero.

- **FLieGenerator& operator*= (const T&)**
  Multiply by scalar and assign.

- **FLieGenerator& operator+= (const FLieGenerator&)**
  Add vector and assign.

- **FLieGenerator operator- ()const**
  Change sign of generator.

- **FLieGenerator& operator-= (const FLieGenerator&)**
  Subtract vector and assign.

- **FLieGenerator& operator/= (const T&)**
  Divide by scalar and assign.

- **const FLieGenerator& operator= (const FLieGenerator&)**

- **bool operator== (const FLieGenerator&)const**

- **T& operator[] (int)**
  Get element.

  Return a reference to element **n**. Result is undefined, if **n** is out of range.

- **const T& operator[] (int)const**
  Get element.

  Return a reference to element **n**. Result is undefined, if **n** is out of range.

- **FLieGenerator scale (const FLieGenerator&)const**
  Scale monomial-wise.

- **FLieGenerator<U,N> transform (const FMatrix<U,2*N,2*N>&)const**
  Substitute matrix in Lie generator.

  The coefficients of the source generator have type T, the elements of the matrix have type U, and the result of multiplying a T by a U must return a U.

- **~FLieGenerator ()**

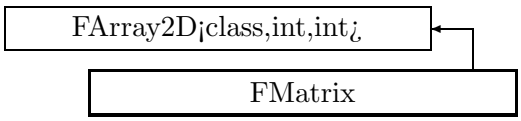

Figure 2.49: Inheritance for class FMatrix

2.65 template class FMatrix <class,int,int>

A templated representation for matrices.
This class implements the basic algebraic operations on general matrices, which need not be square. The destructor generated by the compiler does the correct thing, and is not defined for speed.

Type:	Instantiable
Superclasses:	public FArray2D<class,int,int>
Include file:	./FixedAlgebra/FMatrix.hh

Synopsis (including inherited members)

Public members

- **FMatrix ()**
  Default constructor. (see Section 2.65.1)
- **FMatrix (const FMatrix&)**
  Copy constructor. (see Section 2.65.1)
- **FMatrix (const FArray2D<T,R,C>&)**
  Conversion from two-dimensional array. (see Section 2.65.1)
- **FMatrix (const T&)**
  Constructor. (see Section 2.65.1)
- **FMatrix (const T*)**
  Constructor. (see Section 2.65.1)
- **iterator begin ()**
  Get beginning of data. (see Section 2.58.1)
- **const_iterator begin ()const**
  Get beginning of data. (see Section 2.58.1)
- **col_iterator col_begin (int)**
  Get column iterator. (see Section 2.58.1)
- **const_col_iterator col_begin (int)const**
  Get column iterator. (see Section 2.58.1)
- **col_iterator col_end (int)**
  Get column iterator. (see Section 2.58.1)
- **const_col_iterator col_end (int)const**
  Get column iterator. (see Section 2.58.1)

- **typedef FSlice<T,N> col_iterator**
  Iterator for access by columns. (see Section 2.58.1)
- **typedef FConstSlice<T,N> const_col_iterator**
  Iterator for access by columns. (see Section 2.58.1)
- **typedef const T* const_iterator**
  Iterator for constant array. (see Section 2.58.1)
- **typedef const T* const_row_iterator**
  Iterator for access by rows. (see Section 2.58.1)
- **iterator end ()**
  Get pointer past end of data. (see Section 2.58.1)
- **const_iterator end ()const**
  Get pointer past end of data. (see Section 2.58.1)
- **void getColumn (FArray1D<T,M>&,int)const**
  Fetch column. (see Section 2.58.1)
- **void getRow (FArray1D<T,N>&,int)const**
  Fetch row. (see Section 2.58.1)
- **typedef T* iterator**
  Iterator for the array. (see Section 2.58.1)
- **int ncols ()const**
  Get number of columns. (see Section 2.58.1)
- **int nrows ()const**
  Get number of rows. (see Section 2.58.1)
- **T& operator() (int,int)**
  Get element. (see Section 2.58.1)
- **const T& operator() (int,int)const**
  Get element. (see Section 2.58.1)
- **FMatrix& operator*= (const T&)**
  Multiply by scalar and assign. (see Section 2.65.1)
- **FMatrix& operator+= (const FMatrix&)**
  Add matrix and assign. (see Section 2.65.1)
- **FMatrix& operator-= (const FMatrix&)**
  Subtract ,atrix and assign. (see Section 2.65.1)
- **FMatrix& operator/= (const T&)**
  Divide by scalar and assign. (see Section 2.65.1)
- **FMatrix& operator= (const FMatrix&)**
  Assignment. (see Section 2.65.1)

- **FMatrix& operator= (const FArray2D<T,R,C>&)**
  Convert and assign. (see Section 2.65.1)
- **const FArray2D<T,M,N>& operator= (const FArray2D<T,M,N>&)**
  Assignment. (see Section 2.58.1)
- **row_iterator operator[] (int)**
  Get row iterator. (see Section 2.58.1)
- **const_row_iterator operator[] (int)const**
  Get row iterator. (see Section 2.58.1)
- **void putColumn (const FArray1D<T,M>&,int)**
  Store column. (see Section 2.58.1)
- **void putRow (const FArray1D<T,N>&,int)**
  Store row. (see Section 2.58.1)
- **row_iterator row_begin (int)**
  Get row iterator. (see Section 2.58.1)
- **const_row_iterator row_begin (int)const**
  Get row iterator. (see Section 2.58.1)
- **row_iterator row_end (int)**
  Get row iterator. (see Section 2.58.1)
- **const_row_iterator row_end (int)const**
  Get row iterator. (see Section 2.58.1)
- **typedef T* row_iterator**
  Iterator for access by rows. (see Section 2.58.1)
- **int size ()const**
  Get total size. (see Section 2.58.1)
- **void swapColumns (int,int)**
  Exchange columns. (see Section 2.58.1)
- **void swapRows (int,int)**
  Exchange rows. (see Section 2.58.1)
- **FMatrix<T,C,R> transpose ()const**
  FMatrix transpose. (see Section 2.65.1)
- **typedef T value_type**
  The value type of the array. (see Section 2.58.1)

2.65.1 Detailed descriptions

Public members

- **FMatrix ()**
  Default constructor.

  Construct zero matrix.

- **FMatrix (const FMatrix&)**
  Copy constructor.

- **FMatrix (const FArray2D<T,R,C>&)**
  Conversion from two-dimensional array.

- **FMatrix (const T&)**
  Constructor.

  Set all matrix elements to **t**.

- **FMatrix (const T*)**
  Constructor.

  Fill all matrix element form the C-array **t**.

- **FMatrix& operator*= (const T&)**
  Multiply by scalar and assign.

- **FMatrix& operator+= (const FMatrix&)**
  Add matrix and assign.

- **FMatrix& operator-= (const FMatrix&)**
  Subtract ,atrix and assign.

- **FMatrix& operator/= (const T&)**
  Divide by scalar and assign.

- **FMatrix& operator= (const FMatrix&)**
  Assignment.

- **FMatrix& operator= (const FArray2D<T,R,C>&)**
  Convert and assign.

- **FMatrix<T,C,R> transpose ()const**
  FMatrix transpose.

2.66 template class FMonomial <int>

Representation of the exponents for a monomial with fixed dimension.

Type:	Instantiable
Include file:	./FixedAlgebra/FMonomial.hh

Synopsis (including inherited members)

Public members

- **FMonomial (int)**
  Constructor, defines variable var. (see Section 2.66.1)
- **FMonomial ()**
  (see Section 2.66.1)
- **FMonomial (const FMonomial&)**
  (see Section 2.66.1)
- **int getOrder ()const**
  Compute the monomial's order. (see Section 2.66.1)
- **int getVariables ()const**
  Get the monomial's number of variables. (see Section 2.66.1)
- **FMonomial operator* (const FMonomial&)const**
  Get exponent set of a product. (see Section 2.66.1)
- **const FMonomial& operator= (const FMonomial&)**
  (see Section 2.66.1)
- **int& operator[] (int)**
  Return reference to exponent. (see Section 2.66.1)
- **int operator[] (int)const**
  Return value of exponent. (see Section 2.66.1)
- **~FMonomial ()**
  (see Section 2.66.1)

2.66.1 Detailed descriptions

Public members

- **FMonomial (int)**
  Constructor, defines variable var.
- **FMonomial ()**
- **FMonomial (const FMonomial&)**

- **int getOrder ()const**
  Compute the monomial's order.
- **int getVariables ()const**
  Get the monomial's number of variables.
- **FMonomial operator* (const FMonomial&)const**
  Get exponent set of a product.
- **const FMonomial& operator= (const FMonomial&)**
- **int& operator[] (int)**
  Return reference to exponent.
- **int operator[] (int)const**
  Return value of exponent.
- **˜FMonomial ()**

2.67 template class FNormalForm <int>

Normal form of a truncated Taylor series map.
Compute from representation of a map which can be a nil-potent, static or dynamic symplectic map. Implementation of an algorithm described in center M. Berz, E. Forest and J. Irwin,br Particle Accelerators, 1989, Vol. 24, pp. 91-107. /center

Type:	Instantiable
Include file:	./FixedAlgebra/FNormalForm.hh

Synopsis (including inherited members)

Public members

- **FNormalForm (const FVps<double,N>&)**
  Constructor. (see Section 2.67.1)
- **FNormalForm ()**
  (see Section 2.67.1)
- **FNormalForm (const FNormalForm&)**
  (see Section 2.67.1)
- **FMatrix<double,N/2,N/2> anharmonicity ()const**
  Get anharmonicities as a matrix. (see Section 2.67.1)
- **int degreesOfFreedom ()const**
  Get number of stable degrees of freedom. (see Section 2.67.1)
- **const FVector<complex<double>,N>& eigenValues ()const**
  Get eigenvalues of the linear part as a complex vector. (see Section 2.67.1)
- **const FMatrix<double,N,N>& eigenVectors ()const**
  Get eigenvectors of the linear part in packed form. (see Section 2.67.1)
- **FTps<double,N> invariant (int)const**
  Get invariant polynomial for mode i. (see Section 2.67.1)
- **const FTps<double,N>& normalForm ()const**
  Get normal-form map as a Lie transform. (see Section 2.67.1)
- **const FTps<double,N>& normalisingMap ()const**
  Get normalising map as a Lie transform. (see Section 2.67.1)
- **˜FNormalForm ()**
  (see Section 2.67.1)

Protected members

- **void orderModes (FVector<complex<double>,N>,FMatrix<double,N,N>)**
  (see Section 2.67.1)

2.67.1 Detailed descriptions

Public members

- **FNormalForm (const FVps<double,N>&)**
  Constructor.

  Perform normal-form analysis of **map**.

- **FNormalForm ()**

- **FNormalForm (const FNormalForm&)**

- **FMatrix<double,N/2,N/2> anharmonicity ()const**
  Get anharmonicities as a matrix.

- **int degreesOfFreedom ()const**
  Get number of stable degrees of freedom.

- **const FVector<complex<double>,N>& eigenValues ()const**
  Get eigenvalues of the linear part as a complex vector.

- **const FMatrix<double,N,N>& eigenVectors ()const**
  Get eigenvectors of the linear part in packed form.

- **FTps<double,N> invariant (int)const**
  Get invariant polynomial for mode i.

- **const FTps<double,N>& normalForm ()const**
  Get normal-form map as a Lie transform.

- **const FTps<double,N>& normalisingMap ()const**
  Get normalising map as a Lie transform.

- **~FNormalForm ()**

Protected members

- **void orderModes (FVector<complex<double>,N>,FMatrix<double,N,N>)**

2.68 template class FSlice <class,int>

An iterator permitting to iterate with a stride different from 1.

Type:	Instantiable
Include file:	./FixedAlgebra/FSlice.hh

Synopsis (including inherited members)

Public members

- **FSlice (T*)**
  Constructor for array. (see Section 2.68.1)
- **FSlice (const FSlice&)**
  (see Section 2.68.1)
- **typedef ptrdiff_t difference_type**
  The pointer difference type. (see Section 2.68.1)
- **typedef std::random_access_iterator_tag iterator_category**
  The iterator tag, taken from the standard template library. (see Section 2.68.1)
- **bool operator!= (const FSlice&)const**
  (see Section 2.68.1)
- **T& operator* ()const**
  Dereference. (see Section 2.68.1)
- **FSlice operator+ (ptrdiff_t)**
  Add multiple stride. (see Section 2.68.1)
- **FSlice& operator++ ()**
  Increment (iterate forward). (see Section 2.68.1)
- **FSlice operator++ (int)**
  Increment (iterate forward). (see Section 2.68.1)
- **FSlice& operator+= (ptrdiff_t)**
  Increment by multiple stride. (see Section 2.68.1)
- **FSlice operator- (ptrdiff_t)**
  Subtract multiple stride. (see Section 2.68.1)
- **difference_type operator- (const FSlice&)const**
  Difference. (see Section 2.68.1)
- **FSlice& operator– ()**
  Decrement (iterate backward). (see Section 2.68.1)
- **FSlice operator– (int)**
  Decrement (iterate backward). (see Section 2.68.1)
- **FSlice& operator-= (ptrdiff_t)**
  Decrement by multiple stride. (see Section 2.68.1)

- **FSlice& operator= (const FSlice&)**
  (see Section 2.68.1)
- **bool operator== (const FSlice&)const**
  (see Section 2.68.1)
- **T& operator[] (int)const**
  Delegate. (see Section 2.68.1)
- **typedef T* pointer**
  The pointer type. (see Section 2.68.1)
- **typedef T& reference**
  The reference type. (see Section 2.68.1)
- **typedef T value_type**
  The value type. (see Section 2.68.1)

2.68.1 Detailed descriptions

Public members

- **FSlice (T*)**
  Constructor for array.
- **FSlice (const FSlice&)**

- **typedef ptrdiff_t difference_type**
  The pointer difference type.
- **typedef std::random_access_iterator_tag iterator_category**
  The iterator tag, taken from the standard template library.
- **bool operator!= (const FSlice&)const**

- **T& operator* ()const**
  Dereference.
- **FSlice operator+ (ptrdiff_t)**
  Add multiple stride.
- **FSlice& operator++ ()**
  Increment (iterate forward).
- **FSlice operator++ (int)**
  Increment (iterate forward).
- **FSlice& operator+= (ptrdiff_t)**
  Increment by multiple stride.
- **FSlice operator- (ptrdiff_t)**
  Subtract multiple stride.

- **difference_type operator- (const FSlice&)const**
  Difference.
- **FSlice& operator– ()**
  Decrement (iterate backward).
- **FSlice operator– (int)**
  Decrement (iterate backward).
- **FSlice& operator-= (ptrdiff_t)**
  Decrement by multiple stride.
- **FSlice& operator= (const FSlice&)**
- **bool operator== (const FSlice&)const**
- **T& operator[] (int)const**
  Delegate.
- **typedef T* pointer**
  The pointer type.
- **typedef T& reference**
  The reference type.
- **typedef T value_type**
  The value type.

2.69 template class FStaticFP <int>

Static fixed point of a Truncated power series map.

Type:	Instantiable
Include file:	./FixedAlgebra/FStaticFP.hh

Synopsis (including inherited members)

Public members

- **FStaticFP (const FVps<double,N>&)**
  Constructor. (see Section 2.69.1)
- **FStaticFP ()**
  (see Section 2.69.1)
- **FStaticFP (const FStaticFP&)**
  (see Section 2.69.1)
- **const FVps<double,N>& getDispersion ()const**
  Get the dispersion map. (see Section 2.69.1)
- **const FVps<double,N>& getFixedPoint ()const**
  Get the transformation to the fixed point. (see Section 2.69.1)
- **const FVps<double,N>& getFixedPointMap ()const**
  Get the map around the fixed point. (see Section 2.69.1)
- **~FStaticFP ()**
  (see Section 2.69.1)

2.69.1 Detailed descriptions

Public members

- **FStaticFP (const FVps<double,N>&)**
  Constructor.

  Find fixed point for **map**.
- **FStaticFP ()**
- **FStaticFP (const FStaticFP&)**
- **const FVps<double,N>& getDispersion ()const**
  Get the dispersion map.
- **const FVps<double,N>& getFixedPoint ()const**
  Get the transformation to the fixed point.
- **const FVps<double,N>& getFixedPointMap ()const**
  Get the map around the fixed point.

- **~FStaticFP ()**

2.70 template class FTps <class,int>

Truncated power series in N variables of type T.
All divide operations throw DivideError, if the constant part of the divisor is zero.

Type:	Instantiable
Include file:	./FixedAlgebra/FTps.hh

Synopsis (including inherited members)

Public members

- **static const int EXACT**
  Representation of infinite precision. (see Section 2.70.1)
- **FTps ()**
  Default constructor. (see Section 2.70.1)
- **FTps (const T&)**
  Conversion. (see Section 2.70.1)
- **FTps (int)**
  Conversion. (see Section 2.70.1)
- **FTps (int,int)**
  Constructor. (see Section 2.70.1)
- **FTps (const FTps&)**
  (see Section 2.70.1)
- **T* begin ()const**
  Return the beginning of the monomial array. Return the end of the monomial array. (see Section 2.70.1)
- **FTps derivative (int)const**
  Partial derivative. (see Section 2.70.1)
- **T* end ()const**
  Get exponents for given order. (see Section 2.70.1)
- **T evaluate (const FVector<T,N>&)const**
  Evaluate FTps at point. (see Section 2.70.1)
- **FTps filter (int,int)const**
  (see Section 2.70.1)
- **std::istream& get (std::istream&)**
  Read FTps on the stream **is**. (see Section 2.70.1)
- **const T getCoefficient (int)const**
  Get coefficient. (see Section 2.70.1)
- **const T getCoefficient (const FMonomial<N>&)const**
  Get coefficient. (see Section 2.70.1)

- **static const FMonomial<N>& getExponents (int)**
  (see Section 2.70.1)
- **static int getGlobalTruncOrder ()**
  Return the global truncation order. Extract given range of orders. (see Section 2.70.1)
- **static int getIndex (const FMonomial<N>&)**
  Get Giorgelli index for monomial. (see Section 2.70.1)
- **int getMaxOrder ()const**
  Get maximal order. Get number of coefficients. (see Section 2.70.1)
- **static inline const Array1D<int>& getProductArray (int)**
  Index array for products of monomial "index". (see Section 2.70.1)
- **int getSize ()const**
  Get number of coefficients for given order. (see Section 2.70.1)
- **static int getSize (int)**
  (see Section 2.70.1)
- **static inline const Array1D<TpsSubstitution>& getSubTable ()**
  Return the substitution table. (see Section 2.70.1)
- **int getTruncOrder ()const**
  Get truncation order. Get number of variables. (see Section 2.70.1)
- **int getVariables ()const**
  Set the global truncation order. (see Section 2.70.1)
- **FTps integral (int)const**
  Partial integral. (see Section 2.70.1)
- **FTps inverse (int)const**
  Reciprocal value 1/(*this). (see Section 2.70.1)
- **static FTps makeMonomial (const FMonomial<N>&,const T&)**
  Make monomial. (see Section 2.70.1)
- **static FTps makeVarPower (int,int)**
  Make power. (see Section 2.70.1)
- **static FTps makeVariable (int)**
  Make variable. (see Section 2.70.1)
- **FTps multiply (const FTps&,int)const**
  Truncated multiplication. (see Section 2.70.1)
- **FTps multiplyVariable (int)const**
  Multiply by variable var. (see Section 2.70.1)
- **bool operator!= (const FTps&)const**
  Inequality operator. (see Section 2.70.1)

- **bool operator!= (const T&)const**
  Inequality with constant. (see Section 2.70.1)
- **FTps& operator*= (const FTps&)**
  Multiply and assign. (see Section 2.70.1)
- **FTps& operator*= (const T&)**
  Multiply by constant and assign. (see Section 2.70.1)
- **FTps operator+ ()const**
  Unary plus. (see Section 2.70.1)
- **FTps& operator+= (const FTps&)**
  Add and assign. (see Section 2.70.1)
- **FTps& operator+= (const T&)**
  Add constant and assign. (see Section 2.70.1)
- **FTps operator- ()const**
  Unary minus. (see Section 2.70.1)
- **FTps& operator-= (const FTps&)**
  Subtract and assign. (see Section 2.70.1)
- **FTps& operator-= (const T&)**
  Subtract constant and assign. (see Section 2.70.1)
- **FTps& operator/= (const FTps&)**
  Divide and assign. (see Section 2.70.1)
- **FTps& operator/= (const T&)**
  Divide by constant and assign. (see Section 2.70.1)
- **FTps& operator= (const FTps&)**
  (see Section 2.70.1)
- **FTps& operator= (const T&)**
  Convert and assign. (see Section 2.70.1)
- **bool operator== (const FTps&)const**
  Equality operator. (see Section 2.70.1)
- **bool operator== (const T&)const**
  Equality with constant. (see Section 2.70.1)
- **inline const T operator[] (int)const**
  Get coefficient. (see Section 2.70.1)
- **inline T& operator[] (int)**
  Get coefficient. (see Section 2.70.1)
- **const T operator[] (const FMonomial<N>&)const**
  Get coefficient. (see Section 2.70.1)

- **T& operator[] (const FMonomial<N>&)**
  Get coefficient. (see Section 2.70.1)
- **std::ostream& put (std::ostream&)const**
  Write FTps on the stream **os**. (see Section 2.70.1)
- **FTps scaleMonomials (const FTps&)const**
  Multiply by variable **var**. (see Section 2.70.1)
- **void setCoefficient (int,const T&)**
  Set coefficient. (see Section 2.70.1)
- **void setCoefficient (const FMonomial<N>&,const T&)**
  Set coefficient. (see Section 2.70.1)
- **static void setGlobalTruncOrder (int)**
  (see Section 2.70.1)
- **FTps<T,N> substitute (const FVps<T,N>&)const**
  Substitute. (see Section 2.70.1)
- **FTps<T,N> substitute (const FMatrix<T,N,N>&)const**
  Substitute. (see Section 2.70.1)
- **FTps taylor (const Array1D<T>&,int)const**
  Taylor series. (see Section 2.70.1)
- **FTps truncate (int)**
  Truncate. (see Section 2.70.1)
- **inline void unique ()**
  Make representation unique. (see Section 2.70.1)
- **~FTps ()**
  (see Section 2.70.1)

2.70.1 Detailed descriptions

Public members

- **static const int EXACT**
  Representation of infinite precision.
- **FTps ()**
  Default constructor.

  Constructs zero value.
- **FTps (const T&)**
  Conversion.
- **FTps (int)**
  Conversion.

- **FTps (int,int)**
  Constructor.

  Define the internal orders.

- **FTps (const FTps&)**

- **T* begin ()const**
  Return the beginning of the monomial array. Return the end of the monomial array.

- **FTps derivative (int)const**
  Partial derivative.

  Return partial derivative with respect to variable **var**. Return zero for a constant.

- **T* end ()const**
  Get exponents for given order.

- **T evaluate (const FVector<T,N>&)const**
  Evaluate FTps at point.

- **FTps filter (int,int)const**

- **std::istream& get (std::istream&)**
  Read FTps on the stream is.

- **const T getCoefficient (int)const**
  Get coefficient.

  Return value of the coefficient denoted by the Giorgelli index. Return zero, if index is out of range.

- **const T getCoefficient (const FMonomial<N>&)const**
  Get coefficient.

  Return value of the coefficient denoted by monomial exponents. Return zero, if index is out of range.

- **static const FMonomial<N>& getExponents (int)**

- **static int getGlobalTruncOrder ()**
  Return the global truncation order. Extract given range of orders.

- **static int getIndex (const FMonomial<N>&)**
  Get Giorgelli index for monomial.

- **int getMaxOrder ()const**
  Get maximal order. Get number of coefficients.

- **static inline const Array1D<int>& getProductArray (int)**
  Index array for products of monomial "index".

- **int getSize ()const**
  Get number of coefficients for given order.
- **static int getSize (int)**
- **static inline const Array1D<TpsSubstitution>& getSubTable ()**
  Return the substitution table.
- **int getTruncOrder ()const**
  Get truncation order. Get number of variables.
- **int getVariables ()const**
  Set the global truncation order.
- **FTps integral (int)const**
  Partial integral.

  Return partial integral with respect to variable **var**. Throw LogicalError for a constant.
- **FTps inverse (int)const**
  Reciprocal value 1/(*this).
- **static FTps makeMonomial (const FMonomial<N>&,const T&)**
  Make monomial.

  Construct the monomial with the exponents in **m** and coefficient **t**.
- **static FTps makeVarPower (int,int)**
  Make power.

  Construct **power** of variable **var**.
- **static FTps makeVariable (int)**
  Make variable.

  Construct the variable identified by the index **var**, with total of **nVar** variables.
- **FTps multiply (const FTps&,int)const**
  Truncated multiplication.
- **FTps multiplyVariable (int)const**
  Multiply by variable var.
- **bool operator!= (const FTps&)const**
  Inequality operator.
- **bool operator!= (const T&)const**
  Inequality with constant.
- **FTps& operator*= (const FTps&)**
  Multiply and assign.
- **FTps& operator*= (const T&)**
  Multiply by constant and assign.

- **FTps operator+ ()const**
  Unary plus.
- **FTps& operator+= (const FTps&)**
  Add and assign.
- **FTps& operator+= (const T&)**
  Add constant and assign.
- **FTps operator- ()const**
  Unary minus.
- **FTps& operator-= (const FTps&)**
  Subtract and assign.
- **FTps& operator-= (const T&)**
  Subtract constant and assign.
- **FTps& operator/= (const FTps&)**
  Divide and assign.
- **FTps& operator/= (const T&)**
  Divide by constant and assign.
- **FTps& operator= (const FTps&)**
- **FTps& operator= (const T&)**
  Convert and assign.
- **bool operator== (const FTps&)const**
  Equality operator.
- **bool operator== (const T&)const**
  Equality with constant.
- **inline const T operator[] (int)const**
  Get coefficient.

  Return value of the coefficient denoted by the Giorgelli index. Result is undefined, if index is out of range.
- **inline T& operator[] (int)**
  Get coefficient.

  Return a reference to the coefficient denoted by the Giorgelli index. Result is undefined, if index is out of range. Use as a lvalue is only allowed when the FTps is unique().
- **const T operator[] (const FMonomial<N>&)const**
  Get coefficient.

  Return value of the coefficient denoted by monomial exponents. Result is undefined, if index is out of range.

- **T& operator[] (const FMonomial<N>&)**
  Get coefficient.

  Return a reference to the coefficient denoted by monomial exponents. Result is undefined, if index is out of range. Use as a lvalue is only allowed when the FTps is unique().

- **std::ostream& put (std::ostream&)const**
  Write FTps on the stream os.

- **FTps scaleMonomials (const FTps&)const**
  Multiply by variable var.

  Throw LogicalError for a constant.

- **void setCoefficient (int,const T&)**
  Set coefficient.

  Assign value of the coefficient denoted by the Giorgelli index. Ignore, if index is out of range.

- **void setCoefficient (const FMonomial<N>&,const T&)**
  Set coefficient.

  Assign value of the coeffient denoted by monomial exponents. Ignore, if index is out of range.

- **static void setGlobalTruncOrder (int)**

- **FTps<T,N> substitute (const FVps<T,N>&)const**
  Substitute.

  Substitute map **m** in FTps, giving new FTps.

- **FTps<T,N> substitute (const FMatrix<T,N,N>&)const**
  Substitute.

  Substitute matrix **M** in FTps, giving new FTps.

- **FTps taylor (const Array1D<T>&,int)const**
  Taylor series.

  Expand truncated Taylor series with coefficiens **series** and order **n**.

- **FTps truncate (int)**
  Truncate.

  Change the maximum order to **trunc**; may also reserve more space.

- **inline void unique ()**
  Make representation unique.

- **~FTps ()**

2.71 template class FTpsData <int>

Internal utility class for FTps<T,N> class.

Type:	Instantiable
Include file:	./FixedAlgebra/FTpsData.hh

Synopsis (including inherited members)

Public members

- **FTpsData ()**
  (see Section 2.71.1)
- **static inline const FMonomial<N>& getExponents (int)**
  (see Section 2.71.1)
- **static int getIndex (const FMonomial<N>&)**
  (see Section 2.71.1)
- **static inline int getOrder (int)**
  (see Section 2.71.1)
- **static inline const Array1D<int>& getProductArray (int)**
  (see Section 2.71.1)
- **static inline int getSize (int)**
  (see Section 2.71.1)
- **static inline const Array1D<TpsSubstitution>& getSubTable ()**
  (see Section 2.71.1)
- **static void setup (int)**
  (see Section 2.71.1)
- **~FTpsData ()**
  (see Section 2.71.1)

2.71.1 Detailed descriptions

Public members

- **FTpsData ()**
- **static inline const FMonomial<N>& getExponents (int)**
- **static int getIndex (const FMonomial<N>&)**
- **static inline int getOrder (int)**

- **static inline const Array1D<int>& getProductArray (int)**
- **static inline int getSize (int)**
- **static inline const Array1D<TpsSubstitution>& getSubTable ()**
- **static void setup (int)**
- **~FTpsData ()**

2.72 template class FTpsRep <class,int>

Type:	Instantiable

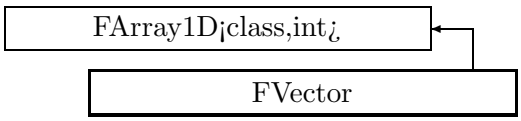

Figure 2.50: Inheritance for class FVector

2.73 template class FVector <class,int>

A templated representation for vectors.
This class implements the arithmetic operations. The copy constructor, destructor, and assignment operator generated by the compiler perform the correct operation. For speed reasons they are not implemented.

Type:	Instantiable
Superclasses:	public FArray1D<class,int>
Include file:	./FixedAlgebra/FVector.hh

Synopsis (including inherited members)

Public members

- **FVector ()**
  Constructor. (see Section 2.73.1)
- **FVector (const T&)**
  Constructor. (see Section 2.73.1)
- **FVector (const T*)**
  Constructor. (see Section 2.73.1)
- **FVector (const FArray1D<T,N>&)**
  Conversion from one-dimensional array. (see Section 2.73.1)
- **iterator begin ()**
  Get iterator pointing to beginning of array. (see Section 2.57.1)
- **const_iterator begin ()const**
  Get iterator pointing to beginning of array. (see Section 2.57.1)
- **typedef const T* const_iterator**
  Iterator for constant array. (see Section 2.57.1)
- **iterator end ()**
  Get iterator pointing past end of array. (see Section 2.57.1)
- **const_iterator end ()const**
  Get iterator pointing past end of array. (see Section 2.57.1)
- **typedef T* iterator**
  Iterator for the array. (see Section 2.57.1)
- **T& operator() (int)**
  Get element. (see Section 2.57.1)

- **const T& operator() (int)const**
  Get element. (see Section 2.57.1)
- **FVector& operator*= (const T&)**
  Multiply by scalar and assign. (see Section 2.73.1)
- **FVector& operator+= (const FVector&)**
  Add FVector and assign. (see Section 2.73.1)
- **FVector operator- ()const**
  Change sign. (see Section 2.73.1)
- **FVector& operator-= (const FVector&)**
  Subtract FVector and assign. (see Section 2.73.1)
- **FVector& operator/= (const T&)**
  Divide by scalar and assign. (see Section 2.73.1)
- **const FArray1D& operator= (const FArray1D&)**
  Assignment. (see Section 2.57.1)
- **T& operator[] (int)**
  Get element. (see Section 2.57.1)
- **const T& operator[] (int)const**
  Get element. (see Section 2.57.1)
- **int size ()const**
  Get array size. (see Section 2.57.1)
- **typedef T value_type**
  The value type of this array. (see Section 2.57.1)

2.73.1 Detailed descriptions

Public members

- **FVector ()**
  Constructor.

  Construct zero FVector.
- **FVector (const T&)**
  Constructor.

  Set all vector elements to **t**.
- **FVector (const T*)**
  Constructor.

  Fill all vector elements from the C-array **t**.
- **FVector (const FArray1D<T,N>&)**
  Conversion from one-dimensional array.

- **FVector& operator*= (const T&)**
  Multiply by scalar and assign.
- **FVector& operator+= (const FVector&)**
  Add FVector and assign.
- **FVector operator- ()const**
  Change sign.
- **FVector& operator-= (const FVector&)**
  Subtract FVector and assign.
- **FVector& operator/= (const T&)**
  Divide by scalar and assign.

2.74 template class FVps <class,int>

Vector truncated power series in n variables.

Type:	Instantiable
Include file:	./FixedAlgebra/FVps.hh

Synopsis (including inherited members)

Public members

- **FVps (int,int)**
  Constructor. (see Section 2.74.1)
- **FVps (const FMatrix<T,N,N>&)**
  Convert from matrix. (see Section 2.74.1)
- **FVps (const FVector<T,N>&)**
  Convert from vector. (see Section 2.74.1)
- **FVps (const LinearMap<T,N>&)**
  Convert from linear map. (see Section 2.74.1)
- **FVps (const TransportMap<T,N>&)**
  Convert from second-order map. (see Section 2.74.1)
- **FVps ()**
  (see Section 2.74.1)
- **FVps (const FVps&)**
  (see Section 2.74.1)
- **FVector<T,N> constantTerm (const FVector<T,N>&)const**
  Evaluate map at point **P**. (see Section 2.74.1)
- **FVector<T,N> constantTerm ()const**
  Evaluate map at origin. (see Section 2.74.1)
- **FVps derivative (int)const**
  Partial derivative. (see Section 2.74.1)
- **FVps filter (int,int)const**
  Extract range of orders, set others to zero. (see Section 2.74.1)
- **std::istream& get (std::istream&)**
  Get a FVps from stream **is**. (see Section 2.74.1)
- **const FTps<T,N>& getComponent (int)const**
  Get component. (see Section 2.74.1)
- **int getDimension ()const**
  Get dimension. (see Section 2.74.1)
- **int getTopOrder ()const**
  Get highest order contained in any component. (see Section 2.74.1)

- **int getTruncOrder ()const**
  Get lowest truncation order in any component. (see Section 2.74.1)
- **int getVariables ()const**
  Get number of variables. (see Section 2.74.1)
- **void identity ()**
  Set to identity. (see Section 2.74.1)
- **FVps integral (int)const**
  Partial integral. (see Section 2.74.1)
- **FVps inverse ()const**
  Inverse. (see Section 2.74.1)
- **FMatrix<T,N,N> linearTerms (const FVector<T,N>&)const**
  Extract linear terms at point **P**. (see Section 2.74.1)
- **FMatrix<T,N,N> linearTerms ()const**
  Extract linear terms at origin. (see Section 2.74.1)
- **FVps& operator*= (const FTps<T,N>&)**
  Multiply and assign. (see Section 2.74.1)
- **FVps& operator*= (const T&)**
  Multiply and assign. (see Section 2.74.1)
- **FVps operator+ ()const**
  Unary plus. (see Section 2.74.1)
- **FVps& operator+= (const FVps&)**
  Add. (see Section 2.74.1)
- **FVps& operator+= (const FVector<T,N>&)**
  Add and assign. (see Section 2.74.1)
- **FVps operator- ()const**
  Unary minus. (see Section 2.74.1)
- **FVps& operator-= (const FVps&)**
  Subtract. (see Section 2.74.1)
- **FVps& operator-= (const FVector<T,N>&)**
  Subtract and assign. (see Section 2.74.1)
- **FVps& operator/= (const FTps<T,N>&)**
  Divide and assign. (see Section 2.74.1)
- **FVps& operator/= (const T&)**
  Divide and assign. (see Section 2.74.1)
- **const FVps& operator= (const FVps&)**
  (see Section 2.74.1)

- **FTps<T,N>& operator[] (int)**
  Get component. (see Section 2.74.1)
- **const FTps<T,N>& operator[] (int)const**
  Get Component. (see Section 2.74.1)
- **std::ostream& put (std::ostream&)const**
  Put a FVps to stream **os**. (see Section 2.74.1)
- **void setComponent (int,const FTps<T,N>&)**
  Set component. (see Section 2.74.1)
- **FVps substitute (const FMatrix<T,N,N>&)const**
  Substitute matrix into map. (see Section 2.74.1)
- **FVps substitute (const FVps&)const**
  Substitute map into map. (see Section 2.74.1)
- **FVps substitute (const FVps&,int)const**
  Substitute map into map and truncate. (see Section 2.74.1)
- **FVps substituteInto (const FMatrix<T,N,N>&)const**
  Substitute map into matrix. (see Section 2.74.1)
- **FVps truncate (int)**
  Truncate. (see Section 2.74.1)
- **~FVps ()**
  (see Section 2.74.1)

Protected members

- **FTps<T,N> data [N]**
  (see Section 2.74.1)

2.74.1 Detailed descriptions

Public members

- **FVps (int,int)**
  Constructor.

  Construct zero map with given order **order** and truncation **trunc**.
- **FVps (const FMatrix<T,N,N>&)**
  Convert from matrix.

  The constant part is set to zero. The linear part is filled from **M**.
- **FVps (const FVector<T,N>&)**
  Convert from vector.

  The constant part is filled from **V**. The linear part is set to the identity.
- **FVps (const LinearMap<T,N>&)**
  Convert from linear map.

- **FVps (const TransportMap<T,N>&)**
  Convert from second-order map.
- **FVps ()**
- **FVps (const FVps&)**
- **FVector<T,N> constantTerm (const FVector<T,N>&)const**
  Evaluate map at point P.
- **FVector<T,N> constantTerm ()const**
  Evaluate map at origin.

  / This is equivalent to extracting constant part.
- **FVps derivative (int)const**
  Partial derivative.

  Return partial derivative with respect to variable **var**.
- **FVps filter (int,int)const**
  Extract range of orders, set others to zero.
- **std::istream& get (std::istream&)**
  Get a FVps from stream is.
- **const FTps<T,N>& getComponent (int)const**
  Get component.

  Return value of component **n**. Throw RangeError, if **n** is out of range.
- **int getDimension ()const**
  Get dimension.

  Return the number of components.
- **int getTopOrder ()const**
  Get highest order contained in any component.
- **int getTruncOrder ()const**
  Get lowest truncation order in any component.
- **int getVariables ()const**
  Get number of variables.

  This is the same in all components.
- **void identity ()**
  Set to identity.
- **FVps integral (int)const**
  Partial integral.

  Return partial integral with respect to variable **var**. Throw LogicalError for a constant.

- **FVps inverse ()const**
  Inverse.
- **FMatrix<T,N,N> linearTerms (const FVector<T,N>&)const**
  Extract linear terms at point P.
- **FMatrix<T,N,N> linearTerms ()const**
  Extract linear terms at origin.

  This is equivalent to extracting linear part.
- **FVps& operator*= (const FTps<T,N>&)**
  Multiply and assign.
- **FVps& operator*= (const T&)**
  Multiply and assign.
- **FVps operator+ ()const**
  Unary plus.
- **FVps& operator+= (const FVps&)**
  Add.
- **FVps& operator+= (const FVector<T,N>&)**
  Add and assign.
- **FVps operator- ()const**
  Unary minus.
- **FVps& operator-= (const FVps&)**
  Subtract.
- **FVps& operator-= (const FVector<T,N>&)**
  Subtract and assign.
- **FVps& operator/= (const FTps<T,N>&)**
  Divide and assign.

  Throw DivideError if constant part of **rhs** is zero.
- **FVps& operator/= (const T&)**
  Divide and assign.

  Throw DivideError if **rhs** is zero.
- **const FVps& operator= (const FVps&)**

- **FTps<T,N>& operator[] (int)**
  Get component.

  Return reference to component **n**. Result is undefined, if index is out of range.
- **const FTps<T,N>& operator[] (int)const**
  Get Component.

  Return constant reference to component **n**. Result is undefined, if index is out of range.

- **std::ostream& put (std::ostream&)const**
  Put a FVps to stream os.
- **void setComponent (int,const FTps<T,N>&)**
  Set component.

  Assign value of component **n**. Throw RangeError, if **n** is out of range.
- **FVps substitute (const FMatrix<T,N,N>&)const**
  Substitute matrix into map.
- **FVps substitute (const FVps&)const**
  Substitute map into map.
- **FVps substitute (const FVps&,int)const**
  Substitute map into map and truncate.
- **FVps substituteInto (const FMatrix<T,N,N>&)const**
  Substitute map into matrix.
- **FVps truncate (int)**
  Truncate.

  Change the maximum order to **trunc**; may also reserve more space.
- **~FVps ()**

Protected members

- **FTps<T,N> data [N]**

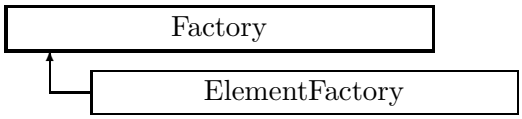

Figure 2.51: Inheritance for class Factory

2.75 class Factory

Abstract interface for an element factory.

Type:	abstract
Include file:	./Construction/Factory.hh

Synopsis (including inherited members)

Public members

- **Factory ()**
  (see Section 2.75.1)
- **virtual bool define (ElementBase*)**
  Define a new element. (see Section 2.75.1)
- **virtual void erase (const string&)**
  Erase element by name. (see Section 2.75.1)
- **virtual ElementBase* find (const string&)const**
  Find element by name. (see Section 2.75.1)
- **virtual ElementBase* makeElement (const string&,const string&,const AttributeSet&)**
  Make new element. (see Section 2.75.1)
- **virtual bool storeElement (ElementBase*)**
  Define a new element. (see Section 2.75.1)
- **virtual ~Factory ()**
  (see Section 2.75.1)

2.75.1 Detailed descriptions

Public members

- **Factory ()**

- **virtual bool define (ElementBase*)**
  Define a new element.

  The element **newElement** is linked to the repository. If an element with the same name exists already, replacement is rejected, and **newElement** is deleted.

- **virtual void erase (const string&)**
  Erase element by name.

  If there is no element with the given **name**, the request is ignored.

- **virtual ElementBase* find (const string&)const**
  Find element by name.

  If an element with the name **name** exists, return a pointer to this element, otherwise return NULL.

- **virtual ElementBase* makeElement (const string&,const string&,const AttributeSet&)**
  Make new element.

  Create a new element with the type **type**, the name **name** and the attributes in **set**. If an element with the name **name** already exists, it is replaced.

- **virtual bool storeElement (ElementBase*)**
  Define a new element.

  The element **newElement** is linked to the repository. If an element with the same name exists already, it is replaced.

- **virtual ~Factory ()**

Figure 2.52: Inheritance for class FileStream

2.76 class FileStream

A stream of input tokens.
The source of tokens is a named disk file.

Type:	Instantiable
Superclasses:	public AbsFileStream
Include file:	./Parser/FileStream.hh

Synopsis (including inherited members)

Public members

- **FileStream (const string&)**
  Constructor. (see Section 2.76.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual bool fillLine ()**
  Read next input line. (see Section 2.76.1)
- **static bool getEcho ()**
  Return echo flag. (see Section 2.76.1)
- **int getLine ()const**
  Return line number. (see Section 2.160.1)
- **const string& getName ()const**
  Return stream name. (see Section 2.160.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **void putBack (const Token&)**
  Put token back to stream. (see Section 2.160.1)
- **virtual Token readToken ()**
  Read single token from file. (see Section 2.2.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **static void setEcho (bool)**
  Set echo flag. (see Section 2.76.1)
- **virtual ~FileStream ()**
  Destructor. (see Section 2.76.1)

2.76.1 Detailed descriptions

Public members

- **FileStream (const string&)**
  Constructor.

  Open associated file.

- **virtual bool fillLine ()**
  Read next input line.

- **static bool getEcho ()**
  Return echo flag.

- **static void setEcho (bool)**
  Set echo flag.

  If **flag** is true, the subsequent input lines are echoed to the standard error stream.

- **virtual ~FileStream ()**
  Destructor.

  Close associated file.

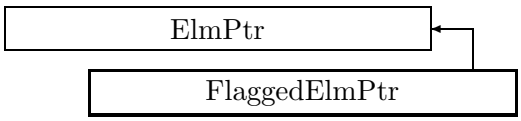

Figure 2.53: Inheritance for class FlaggedElmPtr

2.77 class FlaggedElmPtr

A section of a beam line.
A FlaggedBeamline is built as a list of FlaggedElmPtr objects. A FlaggedElmPtr contains two flags:

reflected If true, this section of the line is reversed.

selected Can be set to indicate a selected status for algorithms.

Type:	Instantiable
Superclasses:	public ElmPtr
Include file:	./Beamlines/FlaggedElmPtr.hh

Synopsis (including inherited members)

Public members

- **FlaggedElmPtr (const ElmPtr&,bool,bool)**
  Constructor. (see Section 2.77.1)
- **FlaggedElmPtr (const FlaggedElmPtr&)**
  Copy constructor. (see Section 2.77.1)
- **FlaggedElmPtr ()**
  (see Section 2.77.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor. (see Section 2.77.1)
- **inline int getCounter ()const**
  Get clone counter. (see Section 2.77.1)
- **inline ElementBase* getElement ()const**
  Get the element pointer. (see Section 2.55.1)
- **inline bool getReflectionFlag ()const**
  Get reflection flag. (see Section 2.77.1)
- **inline bool getSelectionFlag ()const**
  Get selection flag. (see Section 2.77.1)
- **inline void setCounter (int)const**
  Set clone counter. (see Section 2.77.1)
- **inline void setElement (ElementBase*)**
  Set the element pointer. (see Section 2.55.1)

- **inline void setReflectionFlag (bool)const**
  Set reflection flag. (see Section 2.77.1)
- **inline void setSelectionFlag (bool)const**
  Get selection flag. (see Section 2.77.1)
- **virtual ~FlaggedElmPtr ()**
  (see Section 2.77.1)

Protected members

- **mutable bool isReflected**
  The reflection flag. (see Section 2.77.1)
- **mutable bool isSelected**
  The selection flag. (see Section 2.77.1)
- **mutable int itsCounter**
  Clone counter. (see Section 2.77.1)
- **Pointer<ElementBase> itsElement**
  (see Section 2.55.1)

2.77.1 Detailed descriptions

Public members

- **FlaggedElmPtr (const ElmPtr&,bool,bool)**
  Constructor.

  Assign an element to this beamline position.
- **FlaggedElmPtr (const FlaggedElmPtr&)**
  Copy constructor.
- **FlaggedElmPtr ()**

- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor.

  If any error occurs, this method throws an exception.
- **inline int getCounter ()const**
  Get clone counter.

  Return the value set by the last call to `setCounter()`. See **FlaggedElmPtr::itsCounter** for details.
- **inline bool getReflectionFlag ()const**
  Get reflection flag.
- **inline bool getSelectionFlag ()const**
  Get selection flag.

- **inline void setCounter (int)const**
  Set clone counter.

  The value set is the entire responsibility of any algorithm using it. See **FlaggedElmPtr::itsCounter** for details.

- **inline void setReflectionFlag (bool)const**
  Set reflection flag.

- **inline void setSelectionFlag (bool)const**
  Get selection flag.

- **virtual ~FlaggedElmPtr ()**

Protected members

- **mutable bool isReflected**
  The reflection flag.

  If true, the portion of the line pointed at by **itsElement** is reflected, i.e. its elements occur in reverse order.

- **mutable bool isSelected**
  The selection flag.

  This flag can be set to indicate a "selected" status for certain algorithms.

- **mutable int itsCounter**
  Clone counter.

  This value can be set and interrogated by an algorithm. It is not used by the CLASSIC library.

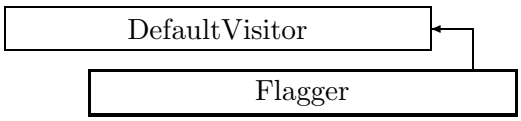

Figure 2.54: Inheritance for class Flagger

2.78 class Flagger

Set/reset all selection flags in a beam line built from FlaggedElmPtr objects.

Type:	Instantiable
Superclasses:	public DefaultVisitor
Include file:	./Algorithms/Flagger.hh

Synopsis (including inherited members)

Public members

- **Flagger (const Beamline&,bool)**
  Constructor. (see Section 2.78.1)
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper.. (see Section 2.37.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam. (see Section 2.37.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.37.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.37.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector. (see Section 2.37.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.37.1)
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic. (see Section 2.37.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift. (see Section 2.37.1)

- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Set selection flag in the given FlaggedElmPtr. (see Section 2.78.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson. (see Section 2.37.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.37.1)
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker. (see Section 2.37.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor. (see Section 2.37.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole. (see Section 2.37.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.37.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.37.1)
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend. (see Section 2.37.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.37.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity. (see Section 2.37.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole. (see Section 2.37.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend. (see Section 2.37.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.37.1)
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator. (see Section 2.37.1)
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum. (see Section 2.37.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid. (see Section 2.37.1)

- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.37.1)
- **virtual ~Flagger ()**
  (see Section 2.78.1)

2.78.1 Detailed descriptions

Public members

- **Flagger (const Beamline&,bool)**
  Constructor.

  Attach this visitor to **bl**, remember the **set** flag.
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Set selection flag in the given FlaggedElmPtr.
- **virtual ~Flagger ()**

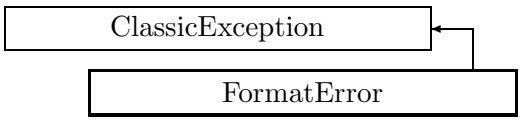

Figure 2.55: Inheritance for class FormatError

2.79 class FormatError

Format error exception.
This exception is thrown, when an input routine detects a format error.

Type:	Instantiable
Superclasses:	public ClassicException
Include file:	./Utilities/FormatError.hh

Synopsis (including inherited members)

Public members

- **FormatError (const string&,const string&)**
  The usual constructor. (see Section 2.79.1)
- **FormatError (const FormatError&)**
  (see Section 2.79.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~FormatError ()**
  (see Section 2.79.1)

2.79.1 Detailed descriptions

Public members

- **FormatError (const string&,const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  msg the message string identifying the exception
- **FormatError (const FormatError&)**
- **virtual ~FormatError ()**

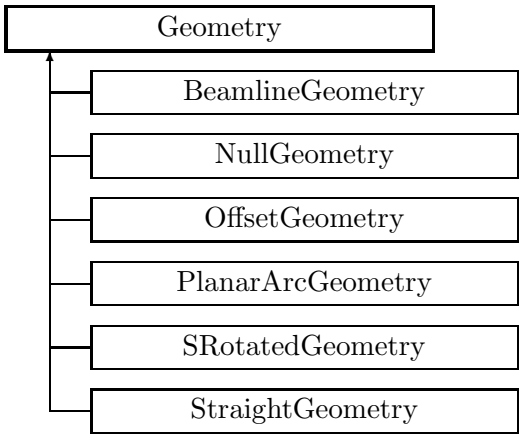

Figure 2.56: Inheritance for class Geometry

2.80 class Geometry

Abstract base class for accelerator geometry classes.
A Geometry can be considered a 3-dimensional space line parameterised by the distance along the line (arc length) s. All Geometries have an exit and entrance plane and an origin. At any position s, a Geometry can define a unique 3-d rectilinear coordinate frame whose origin is on the geometry at s, and whose local z-axis is tangential to the geometry at s. The orientation of the local x- and y-axes are arbitrarilly specified by the Geometry. A special frame, referred to as the Geometry Local Frame (or Local Frame when it is unambiguous) is specified at s = origin. The Local Frame is is used to define that frame about which translations and rotations can be applied to the Geometry. The entrance and exit planes are defined as those x-y planes (z=0, s=constant) in the frames defined at s=entrance and s=exit.

Type:	abstract
Include file:	./BeamlineGeometry/Geometry.hh

Synopsis (including inherited members)

Public members

- **Geometry ()**
  (see Section 2.80.1)
- **Geometry (const Geometry&)**
  (see Section 2.80.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.80.1)
- **virtual double getElementLength ()const**
  Get geometry length. (see Section 2.80.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.80.1)

- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.80.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.80.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.80.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.80.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.80.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.80.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.80.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.80.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.80.1)
- **const Geometry& operator= (const Geometry&)**
  (see Section 2.80.1)
- **virtual void setElementLength (double)**
  Set geometry length. (see Section 2.80.1)
- **virtual ~Geometry ()**
  (see Section 2.80.1)

2.80.1 Detailed descriptions

Public members

- **Geometry ()**

- **Geometry (const Geometry&)**

- **virtual double getArcLength ()const**
  Get arc length.

  Return the length of the geometry, measured along the design arc.
- **virtual double getElementLength ()const**
  Get geometry length.

  Return or the design length of the geometry. Depending on the element this may be the arc length or the straight length.

- **virtual double getEntrance ()const**
  Get entrance position.

  Return the arc length from the origin to the entrance of the geometry (non-positive).

- **virtual Euclid3D getEntranceFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getEntrance()). Return the transform of the local coordinate system from the origin to the entrance of the element.

- **virtual Euclid3D getEntrancePatch ()const**
  Get patch.

  Returns the entrance patch (transformation) which is used to transform the global geometry to the local geometry for a misaligned element at its entrance. The default behaviour returns identity transformation. This function should be overidden by derived concrete classes which model complex geometries.

- **virtual double getExit ()const**
  Get exit position.

  Return the arc length from the origin to the exit of the geometry (non-negative).

- **virtual Euclid3D getExitFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getExit()). Return the transform of the local coordinate system from the origin to the exit of the element.

- **virtual Euclid3D getExitPatch ()const**
  Get patch.

  Returns the entrance patch (transformation) which is used to transform the local geometry to the global geometry for a misaligned element at its exit. The default behaviour returns identity transformation. This function should be overidden by derived concrete classes which model complex geometries.

- **virtual double getOrigin ()const**
  Get origin position.

  Return the arc length from the entrance to the origin of the geometry (non-negative).

- **virtual Euclid3D getTotalTransform ()const**
  Get transform.

  Equivalent to getTransform(getEntrance(), getExit()). Return the transform of the local coordinate system from the entrance to the exit of the element.

- **virtual Euclid3D getTransform (double,double)const**
  Get transform.

  Return the transform of the local coordinate system from the position **fromS** to the position **toS**.

- **virtual Euclid3D getTransform (double)const**
  Get transform.

  Equivalent to getTransform(0.0, s). Return the transform of the local coordinate system from the origin and **s**.

- **const Geometry& operator= (const Geometry&)**

- **virtual void setElementLength (double)**
  Set geometry length.

  Assign the design length of the geometry. Depending on the element this may be the arc length or the straight length.

- **virtual ˜Geometry ()**

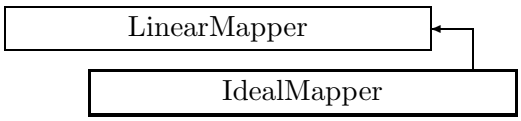

Figure 2.57: Inheritance for class IdealMapper

2.81 class IdealMapper

Build a map using the linear map around the design orbit for each element.
Ignores the following:

Any misalignments,

Any field errors,

Any momentum errors.

Phase space coordinates numbering:

number	name	unit
0	x	metres
1	p_x/p_r	1
2	y	metres
3	p_y/p_r	1
4	v*delta_t	metres
5	delta_p/p_r	1

Where p_r is the constant reference momentum defining the reference frame velocity, m is the rest mass of the particles, and v is the instantaneous velocity of the particle.
Other units used:

quantity	unit
reference momentum	electron-volts
velocity	metres/second
accelerating voltage	volts
separator voltage	volts
frequencies	hertz
phase lags	$2 * pi$

Approximations used:

All elements are represented by maps for finite-length elements.

Geometric transformations ignore rotations about transverse axes and translations along the design orbit and truncate after first order.

Beam-beam elements are ignored.

Type:	Instantiable
Superclasses:	public LinearMapper
Include file:	./Algorithms/IdealMapper.hh

Synopsis (including inherited members)

Public members

- **IdealMapper (const Beamline&,const PartData&,bool,bool)**
  Constructor. (see Section 2.81.1)
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **virtual void getMap (LinearMap<double,6>&)const**
  Return the linear part of the accumulated map. (see Section 2.90.1)
- **virtual void getMap (FVps<double,6>&)const**
  Return the full map accumulated so far. (see Section 2.90.1)
- **virtual void getMatrix (FMatrix<double,6,6>&)const**
  Return the linear part of the accumulated map. (see Section 2.81.1)
- **virtual void setMap (const LinearMap<double,6>&)**
  Reset the linear part of the accumulated map for restart. (see Section 2.90.1)
- **virtual void setMap (const FVps<double,6>&)**
  Reset the full map for restart. (see Section 2.90.1)
- **virtual void setMatrix (const FMatrix<double,6,6>&)**
  Reset the linear part of the accumulated map for restart. (see Section 2.81.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an offset beamline object wrapper. (see Section 2.81.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a BeamBeam. (see Section 2.90.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.90.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.90.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a Corrector. (see Section 2.81.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.81.1)
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a Diagnostic. (see Section 2.90.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a Drift. (see Section 2.90.1)

- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.37.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson. (see Section 2.90.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.90.1)
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a Marker. (see Section 2.90.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a Monitor. (see Section 2.90.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a Multipole. (see Section 2.90.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.81.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.81.1)
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a RBend. (see Section 2.90.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.81.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RFCavity. (see Section 2.90.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RFQuadrupole. (see Section 2.90.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a SBend. (see Section 2.90.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.81.1)
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a Separator. (see Section 2.81.1)
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a Septum. (see Section 2.90.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a Solenoid. (see Section 2.90.1)

- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.37.1)
- **virtual ~IdealMapper ()**
  (see Section 2.81.1)

Protected members

- **void applyDrift (double)**
  Apply drift length. (see Section 2.90.1)
- **void applyEntranceFringe (double,const BMultipoleField&,double)**
  Transforms fringing fields. (see Section 2.90.1)
- **void applyExitFringe (double,const BMultipoleField&,double)**
  (see Section 2.90.1)
- **void applyLinearMap (double,double,double,const FTps<double,2>&,const FTps<double,2>&**
  Apply linear map, defined by the linear expansions Fx and Fy. (see Section 2.90.1)
- **virtual void applyMultipoleBody (double,double,const BMultipoleField&,double)**
  (see Section 2.81.1)
- **virtual void applySBendBody (double,double,double,const BMultipoleField&,double)**
  (see Section 2.81.1)
- **virtual void applyThinMultipole (const BMultipoleField&,double)**
  Thin multipole kick. (see Section 2.81.1)
- **virtual void applyThinSBend (const BMultipoleField&,double,double)**
  Thin SBend kick. (see Section 2.81.1)
- **virtual void applyTransform (const Euclid3D&,double)**
  Apply transform. (see Section 2.81.1)
- **bool back_beam**
  (see Section 2.37.1)
- **bool back_track**
  (see Section 2.37.1)
- **FTps<double,6> buildMultipoleVectorPotential (const BMultipoleField&)**
  Construct the vector potential for a Multipole. (see Section 2.3.1)
- **FTps<double,2> buildSBendVectorPotential (const BMultipoleField&,double)**
  Construct the vector potential for a SBend. (see Section 2.90.1)
- **double flip_B**
  (see Section 2.37.1)
- **double flip_s**
  (see Section 2.37.1)

- **const Beamline& itsLine**
  (see Section 2.37.1)
- **LinearMap<double,6> itsMap**
  (see Section 2.90.1)
- **const PartData itsReference**
  The reference information. (see Section 2.3.1)
- **bool local_flip**
  (see Section 2.37.1)

2.81.1 Detailed descriptions

Public members

- **IdealMapper (const Beamline&,const PartData&,bool,bool)**
  Constructor.

  The beam line to be tracked is **bl**. The particle reference data are taken from **data**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.
- **virtual void getMatrix (FMatrix<double,6,6>&)const**
  Return the linear part of the accumulated map.
- **virtual void setMatrix (const FMatrix<double,6,6>&)**
  Reset the linear part of the accumulated map for restart.
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an offset beamline object wrapper.

  Override to ignore misalignment.
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a Corrector.

  Override to ignore corrector kick.
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper..
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper..
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch.

  Override to ignore patch.
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper..
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper..

- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a Separator.

  Override to ignore separator kick.

- **virtual ~IdealMapper ()**

Protected members

- **virtual void applyMultipoleBody (double,double,const BMultipoleField&,double)**

- **virtual void applySBendBody (double,double,double,const BMultipoleField&,double)**

- **virtual void applyThinMultipole (const BMultipoleField&,double)**
  Thin multipole kick.

  Apply a thin multipole kick (integrated over length) to current map.

- **virtual void applyThinSBend (const BMultipoleField&,double,double)**
  Thin SBend kick.

  Special kick routine for thin SBend.

- **virtual void applyTransform (const Euclid3D&,double)**
  Apply transform.

  Propagate current map through a geometric transformation.

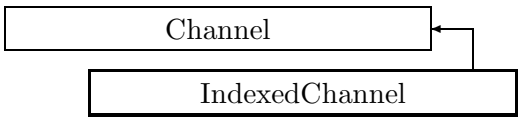

Figure 2.58: Inheritance for class IndexedChannel

2.82 template class IndexedChannel <class>

Access to an indexed double data member.
Template class IndirectChannel allows access to an indexed **double** data member of an object.

Type:	Instantiable
Superclasses:	public Channel
Include file:	./Channels/IndexedChannel.hh

Synopsis (including inherited members)

Public members

- **IndexedChannel (T&,double(int)const,void(int,double),int)**
  Constructor. (see Section 2.82.1)
- **IndexedChannel (const IndexedChannel&)**
  (see Section 2.82.1)
- **virtual IndexedChannel* clone ()const**
  Duplicate the channel. (see Section 2.82.1)
- **virtual bool get (double&)const**
  Fetch from channel. (see Section 2.82.1)
- **virtual bool isSettable ()const**
  Test if settable. (see Section 2.82.1)
- **operator double ()const**
  Read channel. Check if settable. (see Section 2.29.1)
- **double operator+= (double)**
  Subtract and assign **value** to channel. (see Section 2.20.1)
- **double operator-= (double)**
  Store **value** into channel. (see Section 2.20.1)
- **double operator= (double)**
  Assign **value** to channel. Add and assign **value** to channel. (see Section 2.20.1)
- **virtual bool set (double)**
  Store into channel. (see Section 2.82.1)
- **virtual ~IndexedChannel ()**
  (see Section 2.82.1)

2.82.1 Detailed descriptions

Public members

- **IndexedChannel (T&,double(int)const,void(int,double),int)**
  Constructor.

  The constructed channel provides access to an array of an object of class **T**. The channel keeps a reference to **object**, the pointers to member **getF** and **setF**, and the index **index**.

  Values set are transmitted via `object.*setF(index,value)` and read via `value = object.*getF(index)`.

- **IndexedChannel (const IndexedChannel&)**

- **virtual IndexedChannel* clone ()const**
  Duplicate the channel.

- **virtual bool get (double&)const**
  Fetch from channel.

  If the channel can be read, set **value** and return true, otherwise return false.

- **virtual bool isSettable ()const**
  Test if settable.

  Return true, if the channel can receive values, i.e. if the **seffF** pointer is not NULL.

- **virtual bool set (double)**
  Store into channel.

  If the channel can be written, store **value** into it and return true, otherwise return false.

- **virtual ~IndexedChannel ()**

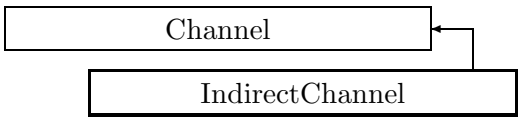

Figure 2.59: Inheritance for class IndirectChannel

2.83 template class IndirectChannel <class>

Access to a double data member.
Template class IndirectChannel allows access to a **double** data member of an object.

Type:	Instantiable
Superclasses:	public Channel
Include file:	./Channels/IndirectChannel.hh

Synopsis (including inherited members)

Public members

- **IndirectChannel (T&,double()const,void(double))**
  Constructor. (see Section 2.83.1)
- **IndirectChannel (const IndirectChannel&)**
  (see Section 2.83.1)
- **virtual IndirectChannel* clone ()const**
  Duplicate the channel. (see Section 2.83.1)
- **virtual bool get (double&)const**
  Fetch from channel. (see Section 2.83.1)
- **virtual bool isSettable ()const**
  Test if settable. (see Section 2.83.1)
- **operator double ()const**
  Read channel. Check if settable. (see Section 2.29.1)
- **double operator+= (double)**
  Subtract and assign **value** to channel. (see Section 2.20.1)
- **double operator-= (double)**
  Store **value** into channel. (see Section 2.20.1)
- **double operator= (double)**
  Assign **value** to channel. Add and assign **value** to channel. (see Section 2.20.1)
- **virtual bool set (double)**
  Store into channel. (see Section 2.83.1)
- **virtual ~IndirectChannel ()**
  (see Section 2.83.1)

2.83.1 Detailed descriptions

Public members

- **IndirectChannel (T&,double()const,void(double))**
  Constructor.

  The constructed channel provides access to a member of an object of class **T**. The channel keeps a reference to **object** and the pointers to member **getF** and **setF**. Values set are transmitted via object.*setF(value) and read via value = object.*getF().

- **IndirectChannel (const IndirectChannel&)**

- **virtual IndirectChannel* clone ()const**
  Duplicate the channel.

- **virtual bool get (double&)const**
  Fetch from channel.

  If the channel can be read, set **value** and return true, otherwise return false.

- **virtual bool isSettable ()const**
  Test if settable.

  Return true, if the channel can be written, i.e. if the set method pointer is not NULL

- **virtual bool set (double)**
  Store into channel.

  If the channel can be written, store **value** into it and return true, otherwise return false.

- **virtual ~IndirectChannel ()**

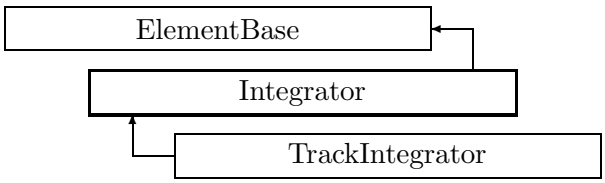

Figure 2.60: Inheritance for class Integrator

2.84 class Integrator

Base class for special integrators.
Integrator is a pure abstract class. It forms the base class for all special propagators through an element or a beam line. If present, it overrides a complete Component or Beamline object.

Type:	abstract
Superclasses:	public ElementBase
Include file:	./AbsBeamline/Integrator.hh

Synopsis (including inherited members)

Public members

- **Integrator (ElementBase*)**
  (see Section 2.84.1)
- **Integrator (const Integrator&)**
  (see Section 2.84.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor. (see Section 2.52.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)

- **inline ElementBase* getElement ()const**
  Return the embedded element. (see Section 2.84.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)

- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.84.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track a particle bunch. (see Section 2.84.1)

- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.84.1)
- **virtual void trackParticle (Particle&,const PartData&,bool,bool)const**
  Track a particle. (see Section 2.84.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Integrator ()**
  (see Section 2.84.1)

Protected members

- **Pointer<ElementBase> itsElement**
  Pointer to the replaced element. (see Section 2.84.1)
- **RCObject& operator= (const RCObject&)**
  (see Section 2.126.1)
- **mutable bool shareFlag**
  (see Section 2.52.1)

2.84.1 Detailed descriptions

Public members

- **Integrator (ElementBase*)**

- **Integrator (const Integrator&)**

- **inline ElementBase* getElement ()const**
  Return the embedded element.
- **virtual void makeSharable ()**
  Set sharable flag.

  The whole structure depending on **this** is marked as sharable. After this call a **copyStructure()** call reuses the element.
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track a particle bunch.

  The first argument describes the particles in the bunch, the second argument describes the reference momentum and mass, **revBeam** true, means that the beam runs backwards, and **revTrack** true, means that we track against the beam.
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map.

  The first argument describes the map to be tracked, the second argument describes the reference momentum and mass, **revBeam** true, means that the beam runs backwards, and **revTrack** true, means that we track against the beam.

- **virtual void trackParticle (Particle&,const PartData&,bool,bool)const**
  Track a particle.

  The first argument describes the particle's phase space position, the second argument describes the particle's momentum and mass, **revBeam** true, means that the beam runs backwards, and **revTrack** true, means that we track against the beam.

- **virtual ~Integrator ()**

Protected members

- **Pointer<ElementBase> itsElement**
  Pointer to the replaced element.

2.85 template class LUMatrix <class>

Triangle decomposition of a matrix.
A templated representation of a LU-decomposition. Constructed from a Matrix<T>, this class remembers the triangular decomposition and can return the back-substitution or the inverse. When LUMatrix is used on a complex matrix, then the statement
`include std::abs;`
is required in the calling file, so as to inject the `abs()` function to the global namespace.

Type:	Instantiable
Include file:	./Algebra/LUMatrix.hh

Synopsis (including inherited members)

Public members

- **LUMatrix (const Matrix<T>&)**
  Constructor. (see Section 2.85.1)
- **LUMatrix ()**
  (see Section 2.85.1)
- **LUMatrix (const LUMatrix<T>&)**
  (see Section 2.85.1)
- **void backSubstitute (Vector<T>&)const**
  Back substitution. (see Section 2.85.1)
- **void backSubstitute (Matrix<T>&)const**
  Back substitution. (see Section 2.85.1)
- **void backSubstitute (Iterator)const**
  (see Section 2.85.1)
- **Matrix<T> inverse ()const**
  Get inverse. (see Section 2.85.1)
- **LUMatrix<T>& operator= (const LUMatrix<T>&)**
  (see Section 2.85.1)
- **~LUMatrix ()**
  (see Section 2.85.1)

2.85.1 Detailed descriptions

Public members

- **LUMatrix (const Matrix<T>&)**
  Constructor.

  Construct triangular decomposition of **m**. Throw **SingularMatrixError** if the **m** is singular.
- **LUMatrix ()**

- **LUMatrix (const LUMatrix<T>&)**

- **void backSubstitute (Vector<T>&)const**
  Back substitution.

  Perform back substitution on the vector **B**. The new **B** is the old **B**, pre-multiplied by the inverse.

- **void backSubstitute (Matrix<T>&)const**
  Back substitution.

  Perform back substitution on the matrix **M**. The new **M** is the old **M**, pre-multiplied by the inverse.

- **void backSubstitute (Iterator)const**

- **Matrix<T> inverse ()const**
  Get inverse.

  Construct and return the inverse.

- **LUMatrix<T>& operator= (const LUMatrix<T>&)**

- **~LUMatrix ()**

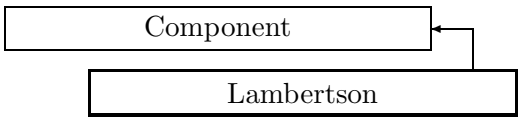

Figure 2.61: Inheritance for class Lambertson

2.86 class Lambertson

Interface for a Lambertson septum.
Class Lambertson defines the abstract interface for a Lambertson septum magnet.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Lambertson.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Lambertson (const string&)**
  Constructor with given name. (see Section 2.86.1)
- **Lambertson ()**
  (see Section 2.86.1)
- **Lambertson (const Lambertson&)**
  (see Section 2.86.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Lambertson. (see Section 2.86.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)

- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Lambertson ()**
  (see Section 2.86.1)

2.86.1 Detailed descriptions

Public members

- **Lambertson (const string&)**
  Constructor with given name.
- **Lambertson ()**
- **Lambertson (const Lambertson&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Lambertson.
- **virtual ~Lambertson ()**

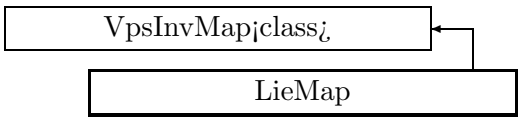

Figure 2.62: Inheritance for class LieMap

2.87 template class LieMap <class>

Lie algebraic map.
LieMap<T> is a truncated Taylor series map with coefficients of type **T**. It implements operations available only for symplectic maps.

Type:	Instantiable
Superclasses:	public VpsInvMap<class>
Include file:	./Algebra/LieMap.hh

Synopsis (including inherited members)

Public members

- **static LieMap<T> ExpMap (const Tps<T>&)**
  Lie series. (see Section 2.87.1)
- **static LieMap<T> ExpMap (const Tps<T>&,const LieMap<T>&)**
  Lie series. (see Section 2.87.1)
- **LieMap (int)**
  Constructor. (see Section 2.87.1)
- **LieMap (const Matrix<T>&)**
  Convert. (see Section 2.87.1)
- **LieMap (const Vector<T>&)**
  Convert. (see Section 2.87.1)
- **LieMap (const Vps<T>&)**
  Conversion. (see Section 2.87.1)
- **LieMap ()**
  (see Section 2.87.1)
- **LieMap (const LieMap<T>&)**
  (see Section 2.87.1)
- **void check ()const**
  Check consistency. (see Section 2.87.1)
- **Vector<T> constantTerm (const Vector<T>&)const**
  Evaluate map at point. (see Section 2.173.1)
- **Vector<T> constantTerm ()const**
  Evaluate map at origin. (see Section 2.173.1)

- **VpsMap<T> derivative (int)const**
  Derivative with respect to variable **var**. (see Section 2.173.1)
- **Vps<T> filter (int,int)const**
  Extract range of orders, set others to zero. (see Section 2.171.1)
- **std::istream& get (std::istream&)**
  Get a Vps<T> from stream is. (see Section 2.171.1)
- **const Tps<T>& getComponent (int)const**
  Get component. (see Section 2.171.1)
- **int getDimension ()const**
  Get dimension (number of Tps<T> components). (see Section 2.171.1)
- **int getTopOrder ()const**
  Get highest order contained in any component. (see Section 2.171.1)
- **int getTruncOrder ()const**
  Get lowest truncation order in any component. (see Section 2.171.1)
- **int getVariables ()const**
  Get number of variables (the same in all components). (see Section 2.171.1)
- **static VpsInvMap<T> identity (int)**
  Set to identity. (see Section 2.172.1)
- **VpsMap<T> integral (int)const**
  Integral with respect to variable **var**. (see Section 2.173.1)
- **VpsInvMap<T> inverse ()const**
  Inverse. (see Section 2.172.1)
- **Matrix<T> linearTerms (const Vector<T>&)const**
  Extract linear terms at point. (see Section 2.173.1)
- **Matrix<T> linearTerms ()const**
  Extract linear terms at origin. (see Section 2.173.1)
- **Vps<T>& operator*= (const Tps<T>&)**
  Multiply by Tps<T> and assign. (see Section 2.171.1)
- **Vps<T>& operator*= (const T&)**
  Multiply by constant and assign. (see Section 2.171.1)
- **Vps<T> operator+ ()const**
  Unary plus. (see Section 2.171.1)
- **Vps<T>& operator+= (const Vps<T>&)**
  Addition. (see Section 2.171.1)
- **Vps<T>& operator+= (const Vector<T>&)**
  Add and assign. (see Section 2.171.1)

- **Vps<T> operator- ()const**
  Unary minus. (see Section 2.171.1)
- **Vps<T>& operator-= (const Vps<T>&)**
  Subtraction. (see Section 2.171.1)
- **Vps<T>& operator-= (const Vector<T>&)**
  Subtract and assign. (see Section 2.171.1)
- **Vps<T>& operator/= (const Tps<T>&)**
  Divide by Tps<T> and assign. (see Section 2.171.1)
- **Vps<T>& operator/= (const T&)**
  Divide by constant and assign. (see Section 2.171.1)
- **LieMap<T>& operator= (const LieMap<T>&)**
  (see Section 2.87.1)
- **VpsInvMap<T>& operator= (const VpsInvMap<T>&)**
  (see Section 2.172.1)
- **VpsMap<T>& operator= (const VpsMap<T>&)**
  (see Section 2.173.1)
- **Vps<T>& operator= (const Vps<T>&)**
  (see Section 2.171.1)
- **Tps<T>& operator[] (int)**
  Get component. (see Section 2.171.1)
- **const Tps<T>& operator[] (int)const**
  Set component. (see Section 2.171.1)
- **std::ostream& put (std::ostream&)const**
  Put a Vps<T> to stream os. (see Section 2.171.1)
- **void setComponent (int,const Tps<T>&)**
  Set component. (see Section 2.171.1)
- **VpsMap<T> substitute (const VpsMap<T>&)const**
  Substitute. (see Section 2.173.1)
- **VpsMap<T> substitute (const Matrix<T>&)const**
  Substitute. (see Section 2.173.1)
- **VpsMap<T> substitute (const VpsMap<T>&,int)const**
  Substitute and truncate. (see Section 2.173.1)
- **VpsMap<T> substituteInto (const Matrix<T>&)const**
  Substitute. (see Section 2.173.1)
- **Vps<T> truncate (int)**
  Truncate, may also increase truncation order. (see Section 2.171.1)
- **~LieMap ()**
  (see Section 2.87.1)

2.87.1 Detailed descriptions

Public members

- **static LieMap<T> ExpMap (const Tps<T>&)**
  Lie series.

  Build the exponential series exp(:H:)Z from the Tps<T> H, acting on the identity map **Z**.

- **static LieMap<T> ExpMap (const Tps<T>&,const LieMap<T>&)**
  Lie series.

  Build the exponential series exp(:H:)M from the Tps<T> H, acting on the map M.

- **LieMap (int)**
  Constructor.

  Construct identity with **nDim** variables.

- **LieMap (const Matrix<T>&)**
  Convert.

  Construct a map from the matrix **M**, which should be symplectic.

- **LieMap (const Vector<T>&)**
  Convert.

  Construct an identity map with a displacement vector **V**.

- **LieMap (const Vps<T>&)**
  Conversion.

  Convert possibly non-symplectic map.

- **LieMap ()**

- **LieMap (const LieMap<T>&)**

- **void check ()const**
  Check consistency.

- **LieMap<T>& operator= (const LieMap<T>&)**

- **~LieMap ()**

2.88 template class LinearFun <class,int>

Linear function in N variables of type T.
All divide operations throw DivideError, if the constant part of the divisor is zero. The copy constructor, destructor, and assignment operator generated by the compiler do the correct thing, and are not defined for speed.

Type:	Instantiable
Include file:	./FixedAlgebra/LinearFun.hh

Synopsis (including inherited members)

Public members

- **LinearFun ()**
  Default constructor. (see Section 2.88.1)
- **LinearFun (const T&)**
  Conversion. (see Section 2.88.1)
- **LinearFun (int)**
  Conversion. (see Section 2.88.1)
- **T evaluate (const FVector<T,N>&)const**
  Evaluate LinearFun at point. (see Section 2.88.1)
- **std::istream& get (std::istream&)**
  Read LinearFun on the stream **is**. (see Section 2.88.1)
- **const T getCoefficient (int)const**
  Get coefficient. (see Section 2.88.1)
- **LinearFun inverse ()const**
  Approximate reciprocal value 1/(*this). (see Section 2.88.1)
- **static LinearFun makeVariable (int)**
  Make variable. (see Section 2.88.1)
- **LinearFun multiply (const LinearFun&)const**
  Multiplication truncated to order one. (see Section 2.88.1)
- **bool operator!= (const LinearFun&)const**
  Inequality operator. (see Section 2.88.1)
- **bool operator!= (const T&)const**
  Inequality with constant. (see Section 2.88.1)
- **LinearFun& operator*= (const LinearFun&)**
  Multiply and assign. (see Section 2.88.1)
- **LinearFun& operator*= (const T&)**
  Multiply by constant and assign. (see Section 2.88.1)
- **LinearFun operator+ ()const**
  Unary plus. (see Section 2.88.1)

- **LinearFun& operator+= (const LinearFun&)**
  Add and assign. (see Section 2.88.1)
- **LinearFun& operator+= (const T&)**
  Add constant and assign. (see Section 2.88.1)
- **LinearFun operator- ()const**
  Unary minus. (see Section 2.88.1)
- **LinearFun& operator-= (const LinearFun&)**
  Subtract and assign. (see Section 2.88.1)
- **LinearFun& operator-= (const T&)**
  Subtract constant and assign. (see Section 2.88.1)
- **LinearFun& operator/= (const LinearFun&)**
  Approximate division and assignation. (see Section 2.88.1)
- **LinearFun& operator/= (const T&)**
  Divide by constant and assign. (see Section 2.88.1)
- **LinearFun& operator= (const T&)**
  Convert and assign. (see Section 2.88.1)
- **bool operator== (const LinearFun&)const**
  Equality operator. (see Section 2.88.1)
- **bool operator== (const T&)const**
  Equality with constant. (see Section 2.88.1)
- **inline const T operator[] (int)const**
  Get coefficient. (see Section 2.88.1)
- **inline T& operator[] (int)**
  Get coefficient. (see Section 2.88.1)
- **std::ostream& put (std::ostream&)const**
  Write LinearFun on the stream **os**. (see Section 2.88.1)
- **void setCoefficient (int,const T&)**
  Set coefficient. (see Section 2.88.1)
- **LinearFun<T,N> substitute (const LinearMap<T,N>&)const**
  Substitute. (see Section 2.88.1)
- **LinearFun<T,N> substitute (const FMatrix<T,N,N>&)const**
  Substitute. (see Section 2.88.1)
- **LinearFun taylor (const T[2])const**
  Taylor series. (see Section 2.88.1)

2.88.1 Detailed descriptions

Public members

- **LinearFun ()**
  Default constructor.

  Construct zero value.

- **LinearFun (const T&)**
  Conversion.

- **LinearFun (int)**
  Conversion.

- **T evaluate (const FVector<T,N>&)const**
  Evaluate LinearFun at point.

- **std::istream& get (std::istream&)**
  Read LinearFun on the stream is.

- **const T getCoefficient (int)const**
  Get coefficient.

  Return value of the coefficient denoted by the index. The index zero stands for the constant term, any other value selects the derivative by the variable index+1. Return zero, if index is out of range.

- **LinearFun inverse ()const**
  Approximate reciprocal value 1/(*this).

- **static LinearFun makeVariable (int)**
  Make variable.

  Construct the variable identified by the index **var**, with total of **nVar** variables.

- **LinearFun multiply (const LinearFun&)const**
  Multiplication truncated to order one.

- **bool operator!= (const LinearFun&)const**
  Inequality operator.

- **bool operator!= (const T&)const**
  Inequality with constant.

- **LinearFun& operator*= (const LinearFun&)**
  Multiply and assign.

- **LinearFun& operator*= (const T&)**
  Multiply by constant and assign.

- **LinearFun operator+ ()const**
  Unary plus.

- **LinearFun& operator+= (const LinearFun&)**
  Add and assign.

- **LinearFun& operator+= (const T&)**
  Add constant and assign.
- **LinearFun operator- ()const**
  Unary minus.
- **LinearFun& operator-= (const LinearFun&)**
  Subtract and assign.
- **LinearFun& operator-= (const T&)**
  Subtract constant and assign.
- **LinearFun& operator/= (const LinearFun&)**
  Approximate division and assignation.
- **LinearFun& operator/= (const T&)**
  Divide by constant and assign.
- **LinearFun& operator= (const T&)**
  Convert and assign.
- **bool operator== (const LinearFun&)const**
  Equality operator.
- **bool operator== (const T&)const**
  Equality with constant.
- **inline const T operator[] (int)const**
  Get coefficient.

  Return value of the coefficient denoted by the index. The index zero stands for the constant term, any other value selects the derivative by the variable index+1. Result is undefined, if index is out of range.
- **inline T& operator[] (int)**
  Get coefficient.

  Return a reference to the coefficient denoted by the index. Result is undefined, if index is out of range. The index zero stands for the constant term, any other value selects the derivative by the variable index+1. Use as a lvalue is only allowed when the LinearFun is unique().
- **std::ostream& put (std::ostream&)const**
  Write LinearFun on the stream os.
- **void setCoefficient (int,const T&)**
  Set coefficient.

  Assign value of the coefficient denoted by the index. The index zero stands for the constant term, any other value selects the derivative by the variable index+1. Ignore, if index is out of range.
- **LinearFun<T,N> substitute (const LinearMap<T,N>&)const**
  Substitute.

  Substitute map **m** in LinearFun, giving new LinearFun.

- **LinearFun<T,N> substitute (const FMatrix<T,N,N>&)const**
  Substitute.

  Substitute matrix **M** in LinearFun, giving new LinearFun.

- **LinearFun taylor (const T[2])const**
  Taylor series.

  Expand Taylor series with coefficiens **series** and order one.

2.89 template class LinearMap <class,int>

Linear map with values of type T in N variables.
The copy constructor, destructor, and assignment operator generated by the compiler do the correct thing, and are not defined for speed.

Type:	Instantiable
Include file:	./FixedAlgebra/LinearMap.hh

Synopsis (including inherited members)

Public members

- **LinearMap ()**
  Default constructor. (see Section 2.89.1)
- **LinearMap (const FVps<T,N>&)**
  Convert from general map. (see Section 2.89.1)
- **LinearMap (const FMatrix<T,N,N>&)**
  Convert from matrix. (see Section 2.89.1)
- **LinearMap (const FVector<T,N>&)**
  Convert from vector. (see Section 2.89.1)
- **FVector<T,N> constantTerm (const FVector<T,N>&)const**
  Evaluate map at point **P**. (see Section 2.89.1)
- **FVector<T,N> constantTerm ()const**
  Evaluate map at origin. (see Section 2.89.1)
- **std::istream& get (std::istream&)**
  Get a LinearMap from stream **is**. (see Section 2.89.1)
- **const LinearFun<T,N>& getComponent (int)const**
  Get component. (see Section 2.89.1)
- **void identity ()**
  Set to identity. (see Section 2.89.1)
- **LinearMap inverse ()const**
  Inverse. (see Section 2.89.1)
- **FMatrix<T,N,N> linearTerms ()const**
  Extract linear terms at origin. (see Section 2.89.1)
- **LinearMap& operator*= (const LinearFun<T,N>&)**
  Multiply and assign. (see Section 2.89.1)
- **LinearMap& operator*= (const T&)**
  Multiply and assign. (see Section 2.89.1)
- **LinearMap operator+ ()const**
  Unary plus. (see Section 2.89.1)

- **LinearMap& operator+= (const LinearMap&)**
  Add. (see Section 2.89.1)
- **LinearMap& operator+= (const FVector<T,N>&)**
  Add and assign. (see Section 2.89.1)
- **LinearMap operator- ()const**
  Unary minus. (see Section 2.89.1)
- **LinearMap& operator-= (const LinearMap&)**
  Subtract. (see Section 2.89.1)
- **LinearMap& operator-= (const FVector<T,N>&)**
  Subtract and assign. (see Section 2.89.1)
- **LinearMap& operator/= (const LinearFun<T,N>&)**
  Divide and assign. (see Section 2.89.1)
- **LinearMap& operator/= (const T&)**
  Divide and assign. (see Section 2.89.1)
- **LinearFun<T,N>& operator[] (int)**
  Get component. (see Section 2.89.1)
- **const LinearFun<T,N>& operator[] (int)const**
  Get Component. (see Section 2.89.1)
- **std::ostream& put (std::ostream&)const**
  Put a LinearMap to stream **os**. (see Section 2.89.1)
- **void setComponent (int,const LinearFun<T,N>&)**
  Set component. (see Section 2.89.1)
- **LinearMap substitute (const FMatrix<T,N,N>&)const**
  Substitute matrix into map. (see Section 2.89.1)
- **LinearMap substitute (const LinearMap&)const**
  Substitute map into map. (see Section 2.89.1)
- **LinearMap substituteInto (const FMatrix<T,N,N>&)const**
  Substitute map into matrix. (see Section 2.89.1)

Protected members

- **LinearFun<T,N> data [N]**
  (see Section 2.89.1)

2.89.1 Detailed descriptions

Public members

- **LinearMap ()**
  Default constructor.

  Construct identity map.

- **LinearMap (const FVps<T,N>&)**
  Convert from general map.
- **LinearMap (const FMatrix<T,N,N>&)**
  Convert from matrix.

  The constant part is set to zero. The linear part is filled from **M**.
- **LinearMap (const FVector<T,N>&)**
  Convert from vector.

  The constant part is filled from **V**. The linear part is set to the identity.
- **FVector<T,N> constantTerm (const FVector<T,N>&)const**
  Evaluate map at point P.
- **FVector<T,N> constantTerm ()const**
  Evaluate map at origin.

  / This is equivalent to extracting constant part.
- **std::istream& get (std::istream&)**
  Get a LinearMap from stream is.
- **const LinearFun<T,N>& getComponent (int)const**
  Get component.

  Return value of component **n**. Throw RangeError, if **n** is out of range.
- **void identity ()**
  Set to identity.
- **LinearMap inverse ()const**
  Inverse.
- **FMatrix<T,N,N> linearTerms ()const**
  Extract linear terms at origin.

  This is equivalent to extracting linear part.
- **LinearMap& operator*= (const LinearFun<T,N>&)**
  Multiply and assign.
- **LinearMap& operator*= (const T&)**
  Multiply and assign.
- **LinearMap operator+ ()const**
  Unary plus.
- **LinearMap& operator+= (const LinearMap&)**
  Add.
- **LinearMap& operator+= (const FVector<T,N>&)**
  Add and assign.
- **LinearMap operator- ()const**
  Unary minus.

- **LinearMap& operator-= (const LinearMap&)**
  Subtract.
- **LinearMap& operator-= (const FVector<T,N>&)**
  Subtract and assign.
- **LinearMap& operator/= (const LinearFun<T,N>&)**
  Divide and assign.

  Throw DivideError if constant part of **rhs** is zero.
- **LinearMap& operator/= (const T&)**
  Divide and assign.

  Throw DivideError if **rhs** is zero.
- **LinearFun<T,N>& operator[] (int)**
  Get component.

  Return reference to component **n**. Result is undefined, if index is out of range.
- **const LinearFun<T,N>& operator[] (int)const**
  Get Component.

  Return constant reference to component **n**. Result is undefined, if index is out of range.
- **std::ostream& put (std::ostream&)const**
  Put a LinearMap to stream os.
- **void setComponent (int,const LinearFun<T,N>&)**
  Set component.

  Assign value of component **n**. Throw RangeError, if **n** is out of range.
- **LinearMap substitute (const FMatrix<T,N,N>&)const**
  Substitute matrix into map.
- **LinearMap substitute (const LinearMap&)const**
  Substitute map into map.
- **LinearMap substituteInto (const FMatrix<T,N,N>&)const**
  Substitute map into matrix.

Protected members

- **LinearFun<T,N> data [N]**

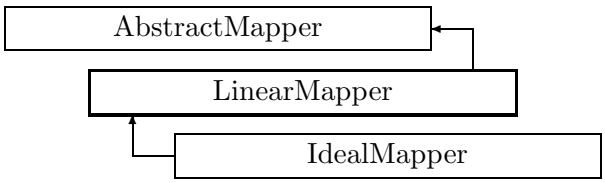

Figure 2.63: Inheritance for class LinearMapper

2.90 class LinearMapper

Build a map using a linear map for each element.
Phase space coordinates numbering:

number	name	unit
0	x	metres
1	p_x/p_r	1
2	y	metres
3	p_y/p_r	1
4	v*delta_t	metres
5	delta_p/p_r	1

Where p_r is the constant reference momentum defining the reference frame velocity, m is the rest mass of the particles, and v is the instantaneous velocity of the particle.
Other units used:

quantity	unit
reference momentum	electron-volts
velocity	metres/second
accelerating voltage	volts
separator voltage	volts
frequencies	hertz
phase lags	$2 * pi$

Approximations used:

All elements are represented by maps for finite-length elements.

Geometric transformations ignore rotations about transverse axes and translations along the design orbit and truncate after second order.

Beam-beam elements are two-dimensional, and the second moment <x,y> of the opposite bunches vanish.

Type:	Instantiable
Superclasses:	public AbstractMapper
Include file:	./Algorithms/LinearMapper.hh

Synopsis (including inherited members)

Public members

- **LinearMapper (const Beamline&,const PartData&,bool,bool)**
  Constructor. (see Section 2.90.1)

- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **virtual void getMap (LinearMap<double,6>&)const**
  Return the linear part of the accumulated map. (see Section 2.90.1)
- **virtual void getMap (FVps<double,6>&)const**
  Return the full map accumulated so far. (see Section 2.90.1)
- **virtual void setMap (const LinearMap<double,6>&)**
  Reset the linear part of the accumulated map for restart. (see Section 2.90.1)
- **virtual void setMap (const FVps<double,6>&)**
  Reset the full map for restart. (see Section 2.90.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an offset beamline object wrapper. (see Section 2.90.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a BeamBeam. (see Section 2.90.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.90.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.90.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a Corrector. (see Section 2.90.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.37.1)
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a Diagnostic. (see Section 2.90.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a Drift. (see Section 2.90.1)
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.37.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson. (see Section 2.90.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.90.1)

- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a Marker. (see Section 2.90.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a Monitor. (see Section 2.90.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a Multipole. (see Section 2.90.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.37.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.90.1)
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a RBend. (see Section 2.90.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.37.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RFCavity. (see Section 2.90.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RFQuadrupole. (see Section 2.90.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a SBend. (see Section 2.90.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.37.1)
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a Separator. (see Section 2.90.1)
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a Septum. (see Section 2.90.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a Solenoid. (see Section 2.90.1)
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.37.1)
- **virtual ~LinearMapper ()**
  (see Section 2.90.1)

Protected members

- **void applyDrift (double)**
  Apply drift length. (see Section 2.90.1)
- **void applyEntranceFringe (double,const BMultipoleField&,double)**
  Transforms fringing fields. (see Section 2.90.1)
- **void applyExitFringe (double,const BMultipoleField&,double)**
  (see Section 2.90.1)
- **void applyLinearMap (double,double,double,const FTps<double,2>&,const FTps<double,2>&**
  Apply linear map, defined by the linear expansions Fx and Fy. (see Section 2.90.1)
- **void applyMultipoleBody (double,double,const BMultipoleField&,double)**
  Apply body of SBend. (see Section 2.90.1)
- **void applySBendBody (double,double,double,const BMultipoleField&,double)**
  Apply thin multipole kick (integrated over length) to all particles. (see Section 2.90.1)
- **void applyThinMultipole (const BMultipoleField&,double)**
  Thin multipole kick. (see Section 2.90.1)
- **void applyTransform (const Euclid3D&,double)**
  Apply transform. (see Section 2.90.1)
- **bool back_beam**
  (see Section 2.37.1)
- **bool back_track**
  (see Section 2.37.1)
- **FTps<double,6> buildMultipoleVectorPotential (const BMultipoleField&)**
  Construct the vector potential for a Multipole. (see Section 2.3.1)
- **FTps<double,2> buildSBendVectorPotential (const BMultipoleField&,double)**
  Construct the vector potential for a SBend. (see Section 2.90.1)
- **double flip_B**
  (see Section 2.37.1)
- **double flip_s**
  (see Section 2.37.1)
- **const Beamline& itsLine**
  (see Section 2.37.1)
- **LinearMap<double,6> itsMap**
  (see Section 2.90.1)
- **const PartData itsReference**
  The reference information. (see Section 2.3.1)
- **bool local_flip**
  (see Section 2.37.1)

2.90.1 Detailed descriptions

Public members

- **LinearMapper (const Beamline&,const PartData&,bool,bool)**
  Constructor.

  The beam line to be tracked is **bl**. The particle reference data are taken from **data**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.

- **virtual void getMap (LinearMap<double,6>&)const**
  Return the linear part of the accumulated map.

- **virtual void getMap (FVps<double,6>&)const**
  Return the full map accumulated so far.

- **virtual void setMap (const LinearMap<double,6>&)**
  Reset the linear part of the accumulated map for restart.

- **virtual void setMap (const FVps<double,6>&)**
  Reset the full map for restart.

- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an offset beamline object wrapper.

- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a BeamBeam.

- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator.

- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component.

  This override calls the component to track the map.

- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a Corrector.

- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a Diagnostic.

- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a Drift.

- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson.

- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping.

- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a Marker.

- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a Monitor.
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a Multipole.
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch.
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a RBend.
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RFCavity.
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RFQuadrupole.
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a SBend.
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a Separator.
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a Septum.
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a Solenoid.
- **virtual ~LinearMapper ()**

Protected members

- **void applyDrift (double)**
  Apply drift length.

  Propagate current map through a drift.
- **void applyEntranceFringe (double,const BMultipoleField&,double)**
  Transforms fringing fields.
- **void applyExitFringe (double,const BMultipoleField&,double)**
- **void applyLinearMap (double,double,double,const FTps<double,2>&,const FTps<double,2>&**
  Apply linear map, defined by the linear expansions Fx and Fy.
- **void applyMultipoleBody (double,double,const BMultipoleField&,double)**
  Apply body of SBend.
- **void applySBendBody (double,double,double,const BMultipoleField&,double)**
  Apply thin multipole kick (integrated over length) to all particles.

- **void applyThinMultipole (const BMultipoleField&,double)**
  Thin multipole kick.

  Apply a thin multipole kick (integrated over length) to current map.

- **void applyTransform (const Euclid3D&,double)**
  Apply transform.

  Propagate current map through a geometric transformation.

- **FTps<double,2> buildSBendVectorPotential (const BMultipoleField&,double)**
  Construct the vector potential for a SBend.

- **LinearMap<double,6> itsMap**

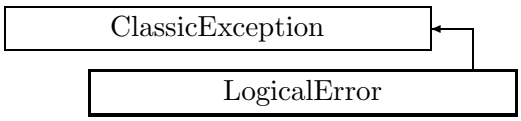

Figure 2.64: Inheritance for class LogicalError

2.91 class LogicalError

Logical error exception.
This exception is thrown, when CLASSIC detects an inconsistent call to a routine or method.

Type:	Instantiable
Superclasses:	public ClassicException
Include file:	./Utilities/LogicalError.hh

Synopsis (including inherited members)

Public members

- **LogicalError (const string&,const string&)**
  The usual constructor. (see Section 2.91.1)
- **LogicalError (const LogicalError&)**
  (see Section 2.91.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~LogicalError ()**
  (see Section 2.91.1)

2.91.1 Detailed descriptions

Public members

- **LogicalError (const string&,const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  msg the message string identifying the exception
- **LogicalError (const LogicalError&)**
- **virtual ~LogicalError ()**

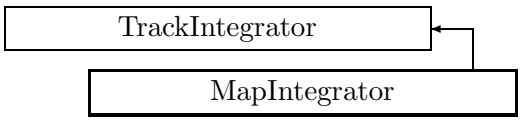

Figure 2.65: Inheritance for class MapIntegrator

2.92 class MapIntegrator

Integrate a map.
An abstract base class for all integrators capable of tracking a transfer map through a beam element. Implements some default behaviour for such integrators.

Type:	abstract
Superclasses:	public TrackIntegrator
Include file:	./Algorithms/MapIntegrator.hh

Synopsis (including inherited members)

Public members

- **MapIntegrator (ElementBase*)**
  (see Section 2.92.1)
- **MapIntegrator (const MapIntegrator&)**
  (see Section 2.92.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor. (see Section 2.92.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual MapIntegrator* clone ()const**
  Make a clone. (see Section 2.92.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **inline ElementBase* getElement ()const**
  Return the embedded element. (see Section 2.84.1)

- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)

- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.84.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track a particle bunch. (see Section 2.92.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.92.1)

- **virtual void trackParticle (Particle&,const PartData&,bool,bool)const**
  Track a particle. (see Section 2.92.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~MapIntegrator ()**
  (see Section 2.92.1)

2.92.1 Detailed descriptions

Public members

- **MapIntegrator (ElementBase*)**
- **MapIntegrator (const MapIntegrator&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor.
- **virtual MapIntegrator* clone ()const**
  Make a clone.
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track a particle bunch.

  The bunch is stored in **bunch**. The particle reference data are taken from **data**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map.

  The map is stored in **map**. The particle reference data are taken from **data**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.
- **virtual void trackParticle (Particle&,const PartData&,bool,bool)const**
  Track a particle.

  The particle is stored in **part**. The particle reference data are taken from **data**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.
- **virtual ~MapIntegrator ()**

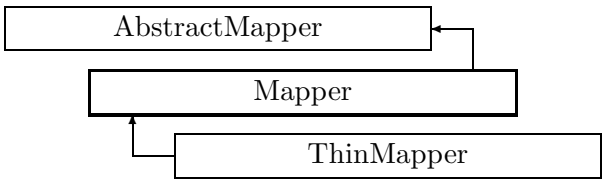

Figure 2.66: Inheritance for class Mapper

2.93 class Mapper

Build transfer map.
An abstract visitor class implementing the default behaviour for all visitors capable of tracking a transfer map through a beam line. It implements some default behaviour for such visitors.
Phase space coordinates (in this order):

x: horizontal displacement (metres).

p_x/p_r: horizontal canonical momentum (no dimension).

y: vertical displacement (metres).

p_y/p_r: vertical canonical momentum (no dimension).

delta_p/p_r: relative momentum error (no dimension).

v*delta_t: time difference delta_t w.r.t. the reference frame which moves with uniform velocity

v_r = c*beta_r = p_r/m

along the design orbit, multiplied by the instantaneous velocity v of the particle (metres).

Where

p_r: is the constant reference momentum defining the reference frame velocity.

m: is the rest mass of the particles.

Other units used:

reference momentum: electron-volts.

accelerating voltage: volts.

separator voltage: volts.

frequencies: hertz.

phase lags: multiples of (2*pi).

Type:	abstract
Superclasses:	public AbstractMapper
Include file:	./Algorithms/Mapper.hh

Synopsis (including inherited members)

Public members

- **Mapper (const Beamline&,const PartData&,bool,bool)**
  Constructor. (see Section 2.93.1)
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **virtual void getMap (LinearMap<double,6>&)const**
  Return the linear part of the accumulated map. (see Section 2.93.1)
- **virtual void getMap (TransportMap<double,6>&)const**
  Return the second-order part of the accumulated map. (see Section 2.93.1)
- **virtual void getMap (FVps<double,6>&)const**
  Return the full map accumulated so far. (see Section 2.93.1)
- **virtual void setMap (const LinearMap<double,6>&)**
  Reset the linear part of the accumulated map for restart. (see Section 2.93.1)
- **virtual void setMap (const TransportMap<double,6>&)**
  Reset the second-order part of the accumulated map for restart. (see Section 2.93.1)
- **virtual void setMap (const FVps<double,6>&)**
  Reset the full map for restart. (see Section 2.93.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper. (see Section 2.93.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam. (see Section 2.3.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.3.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.93.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector. (see Section 2.3.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.37.1)
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic. (see Section 2.3.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift. (see Section 2.3.1)

- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.37.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson. (see Section 2.3.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.93.1)
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker. (see Section 2.3.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor. (see Section 2.3.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole. (see Section 2.3.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.37.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.93.1)
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend. (see Section 2.3.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.37.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity. (see Section 2.3.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole. (see Section 2.3.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend. (see Section 2.3.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.37.1)
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator. (see Section 2.3.1)
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum. (see Section 2.3.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid. (see Section 2.3.1)

- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.37.1)
- **virtual ~Mapper ()**
  (see Section 2.93.1)

Protected members

- **void applyDrift (double)**
  Apply drift length. (see Section 2.93.1)
- **void applyThinMultipole (const BMultipoleField&,double)**
  Thin multipole kick. (see Section 2.93.1)
- **void applyThinSBend (const BMultipoleField&,double,double)**
  Thin SBend kick. (see Section 2.93.1)
- **void applyTransform (const Euclid3D&,double)**
  Apply transform. (see Section 2.93.1)
- **bool back_beam**
  (see Section 2.37.1)
- **bool back_track**
  (see Section 2.37.1)
- **FTps<double,6> buildMultipoleVectorPotential (const BMultipoleField&)**
  Construct the vector potential for a Multipole. (see Section 2.3.1)
- **FTps<double,6> buildSBendVectorPotential (const BMultipoleField&,double)**
  Construct the vector potential for a SBend. (see Section 2.3.1)
- **double flip_B**
  (see Section 2.37.1)
- **double flip_s**
  (see Section 2.37.1)
- **const Beamline& itsLine**
  (see Section 2.37.1)
- **FVps<double,6> itsMap**
  The transfer map being built. (see Section 2.93.1)
- **const PartData itsReference**
  The reference information. (see Section 2.3.1)
- **bool local_flip**
  (see Section 2.37.1)

2.93.1 Detailed descriptions

Public members

- **Mapper (const Beamline&,const PartData&,bool,bool)**
  Constructor.

  The beam line to be tracked is **bl**. The particle reference data are taken from **data**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.

- **virtual void getMap (LinearMap<double,6>&)const**
  Return the linear part of the accumulated map.

- **virtual void getMap (TransportMap<double,6>&)const**
  Return the second-order part of the accumulated map.

- **virtual void getMap (FVps<double,6>&)const**
  Return the full map accumulated so far.

- **virtual void setMap (const LinearMap<double,6>&)**
  Reset the linear part of the accumulated map for restart.

- **virtual void setMap (const TransportMap<double,6>&)**
  Reset the second-order part of the accumulated map for restart.

- **virtual void setMap (const FVps<double,6>&)**
  Reset the full map for restart.

- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper.

- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component.

  This override calls the component to track the map.

- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping.

- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch.

- **virtual ~Mapper ()**

Protected members

- **void applyDrift (double)**
  Apply drift length.

  Propagate current map through a drift.

- **void applyThinMultipole (const BMultipoleField&,double)**
  Thin multipole kick.

  Apply a thin multipole kick (integrated over length) to current map.

- **void applyThinSBend (const BMultipoleField&,double,double)**
  Thin SBend kick.

  Special kick routine for thin SBend.

- **void applyTransform (const Euclid3D&,double)**
  Apply transform.

  Propagate current map through a geometric transformation.

- **FVps<double,6> itsMap**
  The transfer map being built.

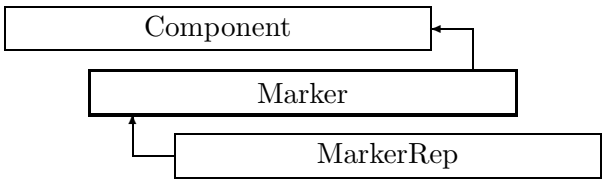

Figure 2.67: Inheritance for class Marker

2.94 class Marker

Interface for a marker.
Class Marker defines the abstract interface for a marker element.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Marker.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Marker (const string&)**
  Constructor with given name. (see Section 2.94.1)
- **Marker ()**
  (see Section 2.94.1)
- **Marker (const Marker&)**
  (see Section 2.94.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Marker. (see Section 2.94.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)

- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Marker ()**
  (see Section 2.94.1)

2.94.1 Detailed descriptions

Public members

- **Marker (const string&)**
  Constructor with given name.
- **Marker ()**
- **Marker (const Marker&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Marker.
- **virtual ~Marker ()**

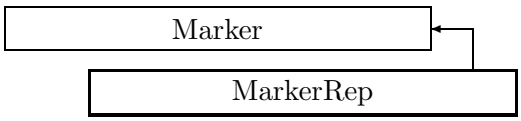

Figure 2.68: Inheritance for class MarkerRep

2.95 class MarkerRep

Representation for a marker element.

Type:	Instantiable
Superclasses:	public Marker
Include file:	./BeamlineCore/MarkerRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **MarkerRep (const string&)**
  Constructor with given name. (see Section 2.95.1)
- **MarkerRep ()**
  (see Section 2.95.1)
- **MarkerRep (const MarkerRep&)**
  (see Section 2.95.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Marker. (see Section 2.94.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.95.1)

- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.95.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.95.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual NullField& getField ()**
  Get field. (see Section 2.95.1)
- **virtual const NullField& getField ()const**
  Get field. (see Section 2.95.1)
- **virtual NullGeometry& getGeometry ()**
  Get geometry. (see Section 2.95.1)
- **virtual const NullGeometry& getGeometry ()const**
  Get geometry. (see Section 2.95.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)

- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.95.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)

- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~MarkerRep ()**
  (see Section 2.95.1)

2.95.1 Detailed descriptions

Public members

- **MarkerRep (const string&)**
  Constructor with given name.
- **MarkerRep ()**
- **MarkerRep (const MarkerRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual double getArcLength ()const**
  Get arc length.

  Always return zero. :return always zero

- **virtual double getElementLength ()const**
  Get design length.

  Always return zero. :return always zero

- **virtual NullField& getField ()**
  Get field.

  Version for non-constant object.

- **virtual const NullField& getField ()const**
  Get field.

  Version for constant object.

- **virtual NullGeometry& getGeometry ()**
  Get geometry.

  Return the element geometry. Version for non-constant object.

- **virtual const NullGeometry& getGeometry ()const**
  Get geometry.

  Return the element geometry Version for constant object.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual ~MarkerRep ()**

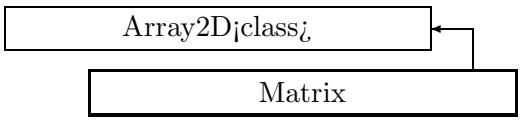

Figure 2.69: Inheritance for class Matrix

2.96 template class Matrix <class>

Matrix.
A templated representation for matrices. This class implements the basic algebraic operations on general matrices, which need not be square.

Type:	Instantiable
Superclasses:	public Array2D<class>
Include file:	./Algebra/Matrix.hh

Synopsis (including inherited members)

Public members

- **Matrix ()**
  Default constructor. (see Section 2.96.1)
- **Matrix (const Array2D<T>&)**
  Conversion. (see Section 2.96.1)
- **Matrix (int,int)**
  Constructor. (see Section 2.96.1)
- **Matrix (int,int,const T&)**
  Constructor. (see Section 2.96.1)
- **Matrix (const Matrix&)**
  (see Section 2.96.1)
- **iterator begin ()**
  Get pointer to beginning of data. (see Section 2.8.1)
- **const_iterator begin ()const**
  Get pointer to beginning of data. (see Section 2.8.1)
- **col_iterator col_begin (int)**
  Get column iterator. (see Section 2.8.1)
- **const_col_iterator col_begin (int)const**
  Get column iterator. (see Section 2.8.1)
- **col_iterator col_end (int)**
  Get column iterator. (see Section 2.8.1)
- **const_col_iterator col_end (int)const**
  Get column iterator. (see Section 2.8.1)

- **typedef SliceIterator<T> col_iterator**
  The iterator type for access by columns. (see Section 2.8.1)
- **typedef ConstSliceIterator<T> const_col_iterator**
  The iterator type for access by columns for a constant array. (see Section 2.8.1)
- **typedef const T* const_iterator**
  The iterator type for sequential access to all elements (see Section 2.8.1)
- **typedef const T* const_row_iterator**
  The iterator type for access by rows for a constant array. (see Section 2.8.1)
- **Vector<T> dotcv (const Vector<T>&)const**
  Matrix times column vector. (see Section 2.96.1)
- **Matrix<T> dotm (const Matrix<T>&)const**
  Matrix product. (see Section 2.96.1)
- **Vector<T> dotrv (const Vector<T>&)const**
  Row vector times matrix. (see Section 2.96.1)
- **iterator end ()**
  Get pointer past end of data. (see Section 2.8.1)
- **const_iterator end ()const**
  Get pointer past end of data. (see Section 2.8.1)
- **void getColumn (Array1D<T>&,int)const**
  Fetch column. (see Section 2.8.1)
- **void getRow (Array1D<T>&,int)const**
  Fetch row. (see Section 2.8.1)
- **typedef T* iterator**
  The iterator type for sequential access to all elements. (see Section 2.8.1)
- **int ncols ()const**
  Get number of columns. (see Section 2.8.1)
- **int nrows ()const**
  Get number of rows. (see Section 2.8.1)
- **T& operator() (int,int)**
  Get element reference. (see Section 2.8.1)
- **const T& operator() (int,int)const**
  Get element value. (see Section 2.8.1)
- **Matrix<T>& operator*= (const T&)**
  Multiply by scalar and assign. (see Section 2.96.1)
- **Matrix<T>& operator*= (const Matrix<T>&)**
  Multiply by matrix and assign. (see Section 2.96.1)

- **Matrix<T>& operator+= (const Matrix<T>&)**
  Add matrix and assign.. (see Section 2.96.1)
- **Matrix<T>& operator+= (const T&)**
  Add scalar. (see Section 2.96.1)
- **Matrix<T>& operator-= (const Matrix<T>&)**
  Subtract matrix and assign. (see Section 2.96.1)
- **Matrix<T>& operator-= (const T&)**
  Subtract scalar. (see Section 2.96.1)
- **Matrix<T>& operator/= (const T&)**
  Divide by scalar and assign. (see Section 2.96.1)
- **Matrix<T>& operator= (const Matrix<T>&)**
  (see Section 2.96.1)
- **Matrix<T>& operator= (const Array2D<T>&)**
  Convert and assign. (see Section 2.96.1)
- **row_iterator operator[] (int)**
  Get row iterator. (see Section 2.8.1)
- **const_row_iterator operator[] (int)const**
  Get row iterator. (see Section 2.8.1)
- **void putColumn (const Array1D<T>&,int)**
  Store column. (see Section 2.8.1)
- **void putRow (const Array1D<T>&,int)**
  Store row. (see Section 2.8.1)
- **row_iterator row_begin (int)**
  Get row iterator. (see Section 2.8.1)
- **const_row_iterator row_begin (int)const**
  Get row iterator. (see Section 2.8.1)
- **row_iterator row_end (int)**
  Get row iterator. (see Section 2.8.1)
- **const_row_iterator row_end (int)const**
  Get row iterator. (see Section 2.8.1)
- **typedef T* row_iterator**
  The iterator type for access by rows. (see Section 2.8.1)
- **int size ()const**
  Get total size (rows times columns). (see Section 2.8.1)
- **void swapColumns (int,int)**
  Exchange columns. (see Section 2.8.1)

- **void swapRows (int,int)**
  Exchange rows. (see Section 2.8.1)
- **Matrix<T> transpose ()const**
  Matrix transpose. (see Section 2.96.1)
- **typedef T value_type**
  The value type of the array. (see Section 2.8.1)
- **~Matrix ()**
  (see Section 2.96.1)

2.96.1 Detailed descriptions

Public members

- **Matrix ()**
  Default constructor.

  Constructs undefined matrix.
- **Matrix (const Array2D<T>&)**
  Conversion.

  From two-dimensional array.
- **Matrix (int,int)**
  Constructor.

  Reserve **rows** x **cols** elements and leave them undefined.
- **Matrix (int,int,const T&)**
  Constructor.

  Reserve **rows** x **cols** elements and set them to t.
- **Matrix (const Matrix&)**
- **Vector<T> dotcv (const Vector<T>&)const**
  Matrix times column vector.
- **Matrix<T> dotm (const Matrix<T>&)const**
  Matrix product.
- **Vector<T> dotrv (const Vector<T>&)const**
  Row vector times matrix.
- **Matrix<T>& operator*= (const T&)**
  Multiply by scalar and assign.
- **Matrix<T>& operator*= (const Matrix<T>&)**
  Multiply by matrix and assign.
- **Matrix<T>& operator+= (const Matrix<T>&)**
  Add matrix and assign..

- **Matrix<T>& operator+= (const T&)**
  Add scalar.

  Add the unit matrix times **t** and assign. Throw SizeError, if matrix is not square.

- **Matrix<T>& operator-= (const Matrix<T>&)**
  Subtract matrix and assign.

- **Matrix<T>& operator-= (const T&)**
  Subtract scalar.

  Subtract the unit matrix times **t** and assign. Throw SizeError, if matrix is not square.

- **Matrix<T>& operator/= (const T&)**
  Divide by scalar and assign.

- **Matrix<T>& operator= (const Matrix<T>&)**

- **Matrix<T>& operator= (const Array2D<T>&)**
  Convert and assign.

- **Matrix<T> transpose ()const**
  Matrix transpose.

- **~Matrix ()**

2.97 class Matrix3D

3-dimensional matrix.
The copy constructor, destructor, and assignment operator generated by the compiler perform the correct operation. For speed reasons they are not implemented.

Type:	Instantiable
Include file:	./BeamlineGeometry/Matrix3D.hh

Synopsis (including inherited members)

Public members

- **static Matrix3D Identity ()**
  Make identity. (see Section 2.97.1)
- **Matrix3D ()**
  Default constructor. (see Section 2.97.1)
- **Matrix3D (const Vector3D&,const Vector3D&,const Vector3D&)**
  Constructor. (see Section 2.97.1)
- **Matrix3D (double,double,double,double,double,double,double,double,double)**
  Constructor. (see Section 2.97.1)
- **Matrix3D inverse ()const**
  Inverse. (see Section 2.97.1)
- **bool isIdentity ()const**
  Test for identity. (see Section 2.97.1)
- **bool operator!= (const Matrix3D&)const**
  (see Section 2.97.1)
- **double& operator() (int,int)**
  Get element. (see Section 2.97.1)
- **double operator() (int,int)const**
  Get element. (see Section 2.97.1)
- **Matrix3D& operator*= (const Matrix3D&)**
  Multiply and assign. (see Section 2.97.1)
- **Matrix3D& operator*= (double)**
  Multiply and assign. (see Section 2.97.1)
- **Matrix3D& operator+= (const Matrix3D&)**
  Add and assign. (see Section 2.97.1)
- **Matrix3D& operator-= (const Matrix3D&)**
  Subtract and assign. (see Section 2.97.1)
- **bool operator== (const Matrix3D&)const**
  (see Section 2.97.1)

- **Matrix3D transpose ()const**
  Transpose. (see Section 2.97.1)

2.97.1 Detailed descriptions

Public members

- **static Matrix3D Identity ()**
  Make identity.
- **Matrix3D ()**
  Default constructor.

  Constructs identity.
- **Matrix3D (const Vector3D&,const Vector3D&,const Vector3D&)**
  Constructor.

  Use the three vectors (a,b,c) as column vectors.
- **Matrix3D (double,double,double,double,double,double,double,double,double)**
  Constructor.

  Use the elements as matrix elements by rows.
- **Matrix3D inverse ()const**
  Inverse.
- **bool isIdentity ()const**
  Test for identity.

  Return true, if **this** is an identity matrix.
- **bool operator!= (const Matrix3D&)const**
- **double& operator() (int,int)**
  Get element.

  Return reference to matrix element (i,k).
- **double operator() (int,int)const**
  Get element.

  Return value of matrix element (i,k).
- **Matrix3D& operator*= (const Matrix3D&)**
  Multiply and assign.
- **Matrix3D& operator*= (double)**
  Multiply and assign.
- **Matrix3D& operator+= (const Matrix3D&)**
  Add and assign.
- **Matrix3D& operator-= (const Matrix3D&)**
  Subtract and assign.

- **bool operator== (const Matrix3D&)const**

- **Matrix3D transpose ()const**
  Transpose.

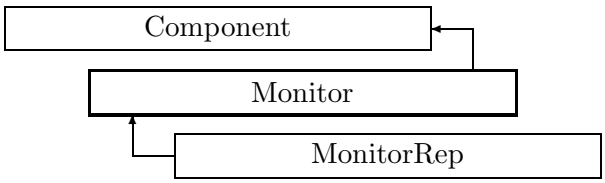

Figure 2.70: Inheritance for class Monitor

2.98 class Monitor

Interface for beam position monitors.
Class Monitor defines the abstract interface for general beam position monitors.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Monitor.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Monitor (const string&)**
  Monitor is off (inactive). Monitor acts on x-plane. Monitor acts on y-plane. Monitor acts on both planes. Constructor with given name. (see Section 2.98.1)
- **Monitor ()**
  (see Section 2.98.1)
- **Monitor (const Monitor&)**
  (see Section 2.98.1)
- **enum Plane**
  Plane selection. (see Section 2.98.1)

- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Monitor. (see Section 2.98.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)

- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.98.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. Version for const object. (see Section 2.98.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Plane getPlane ()const**
  Get plane on which monitor observes. (see Section 2.98.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Monitor ()**
  (see Section 2.98.1)

2.98.1 Detailed descriptions

Public members

- **Monitor (const string&)**
  Monitor is off (inactive). Monitor acts on x-plane. Monitor acts on y-plane. Monitor acts on both planes. Constructor with given name.
- **Monitor ()**

- **Monitor (const Monitor&)**

- **enum Plane**
  Plane selection.
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Monitor.
- **virtual StraightGeometry& getGeometry ()**
  Get geometry.
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. Version for const object.
- **virtual Plane getPlane ()const**
  Get plane on which monitor observes.
- **virtual ~Monitor ()**

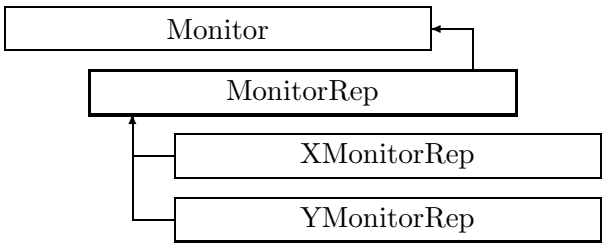

Figure 2.71: Inheritance for class MonitorRep

2.99 class MonitorRep

Representation for an orbit position monitor.
The base class observes both planes.

Type:	Instantiable
Superclasses:	public Monitor
Include file:	./BeamlineCore/MonitorRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **MonitorRep (const string&)**
  Constructor with given name. (see Section 2.99.1)
- **MonitorRep ()**
  (see Section 2.99.1)
- **MonitorRep (const MonitorRep&)**
  (see Section 2.99.1)
- **enum Plane**
  Plane selection. (see Section 2.98.1)

- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Monitor. (see Section 2.98.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.99.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.99.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual NullField& getField ()**
  Get field. (see Section 2.99.1)
- **virtual const NullField& getField ()const**
  Get field. (see Section 2.99.1)

- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.99.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.99.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.99.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Plane getPlane ()const**
  Get planes. (see Section 2.99.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.99.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setActive (bool)**
  Set active flag. (see Section 2.99.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~MonitorRep ()**
  (see Section 2.99.1)

Protected members

- **bool active**
  The active/inactive flag. (see Section 2.99.1)
- **NullField field**
  The zero magnetic field. (see Section 2.99.1)

- **StraightGeometry geometry**
  The monitor geometry. (see Section 2.99.1)
- **RCObject& operator= (const RCObject&)**
  (see Section 2.126.1)
- **mutable bool shareFlag**
  (see Section 2.52.1)

2.99.1 Detailed descriptions

Public members

- **MonitorRep (const string&)**
  Constructor with given name.
- **MonitorRep ()**
- **MonitorRep (const MonitorRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.
- **virtual NullField& getField ()**
  Get field.

  Version for non-constant object.
- **virtual const NullField& getField ()const**
  Get field.

  Version for constant object.
- **virtual StraightGeometry& getGeometry ()**
  Get geometry.

  Return the element geometry. Version for non-constant object.
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.

  Return the element geometry. Version for constant object.
- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual Plane getPlane ()const**
  Get planes.

  Return the plane(s) observed by this monitor.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual void setActive (bool)**
  Set active flag.

  If **flag** is true, the monitor is activated, otherwise it is deactivated.

- **virtual ~MonitorRep ()**

Protected members

- **bool active**
  The active/inactive flag.

- **NullField field**
  The zero magnetic field.

- **StraightGeometry geometry**
  The monitor geometry.

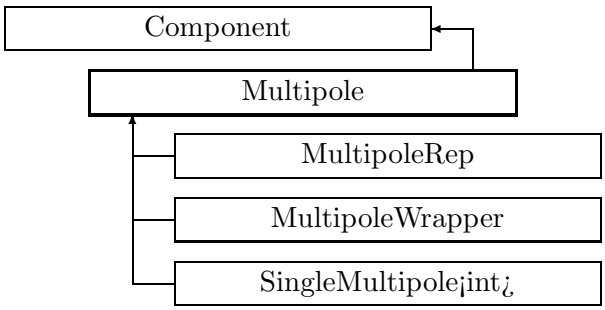

Figure 2.72: Inheritance for class Multipole

2.100 class Multipole

Interface for general multipole.
Class Multipole defines the abstract interface for magnetic multipoles. The order n of multipole components runs from 1 to N and is dynamically adjusted. It is connected with the number of poles by the table

1	dipole
2	quadrupole
3	sextupole
4	octupole
5	decapole
n	multipole with 2*n poles

Units for multipole strengths are Teslas / m**(n-1).

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Multipole.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)

- **Multipole (const string&)**
  Constructor with given name. (see Section 2.100.1)
- **Multipole ()**
  (see Section 2.100.1)
- **Multipole (const Multipole&)**
  (see Section 2.100.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Multipole. (see Section 2.100.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)

- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BMultipoleField& getField ()**
  Get multipole field. (see Section 2.100.1)
- **virtual const BMultipoleField& getField ()const**
  Get multipole field. Version for const object. (see Section 2.100.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.100.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.100.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **double getNormalComponent (int)const**
  Get normal component. (see Section 2.100.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **double getSkewComponent (int)const**
  Get skew component. (see Section 2.100.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)

- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **void setNormalComponent (int,double)**
  Set normal component. (see Section 2.100.1)
- **void setSkewComponent (int,double)**
  Set skew component. (see Section 2.100.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)

- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Multipole ()**
  (see Section 2.100.1)

2.100.1 Detailed descriptions

Public members

- **Multipole (const string&)**
  Constructor with given name.
- **Multipole ()**
- **Multipole (const Multipole&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Multipole.
- **virtual BMultipoleField& getField ()**
  Get multipole field.
- **virtual const BMultipoleField& getField ()const**
  Get multipole field. Version for const object.
- **virtual StraightGeometry& getGeometry ()**
  Get geometry.
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.
- **double getNormalComponent (int)const**
  Get normal component.

  Return the normal component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the return value is zero.
- **double getSkewComponent (int)const**
  Get skew component.

  Return the skew component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the return value is zero.
- **void setNormalComponent (int,double)**
  Set normal component.

  Set the normal component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the component is created.
- **void setSkewComponent (int,double)**
  Set skew component.

  Set the skew component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the component is created.

- **virtual ~Multipole ()**

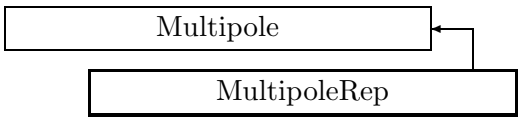

Figure 2.73: Inheritance for class MultipoleRep

2.101 class MultipoleRep

Representation for a general multipole.

Type:	Instantiable
Superclasses:	public Multipole
Include file:	./BeamlineCore/MultipoleRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **MultipoleRep (const string&)**
  Constructor with given name. (see Section 2.101.1)
- **MultipoleRep ()**
  (see Section 2.101.1)
- **MultipoleRep (const MultipoleRep&)**
  (see Section 2.101.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Multipole. (see Section 2.100.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.101.1)

- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.101.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BMultipoleField& getField ()**
  Get field. (see Section 2.101.1)
- **virtual const BMultipoleField& getField ()const**
  Get field. (see Section 2.101.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.101.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.101.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.101.1)

- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **double getNormalComponent (int)const**
  Get normal component. (see Section 2.100.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **double getSkewComponent (int)const**
  Get skew component. (see Section 2.100.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.101.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.101.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)

- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setField (const BMultipoleField&)**
  Set mulitpole field. (see Section 2.101.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **void setNormalComponent (int,double)**
  Set normal component. (see Section 2.100.1)
- **void setSkewComponent (int,double)**
  Set skew component. (see Section 2.100.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ˜MultipoleRep ()**
  (see Section 2.101.1)

2.101.1 Detailed descriptions

Public members

- **MultipoleRep (const string&)**
  Constructor with given name.
- **MultipoleRep ()**

- **MultipoleRep (const MultipoleRep&)**

- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.

- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey**r and returns it. If the attribute does not exist, it returns NULL.

- **virtual BMultipoleField& getField ()**
  Get field.

  Version for non-constant object.

- **virtual const BMultipoleField& getField ()const**
  Get field.

  Version for constant object.

- **virtual StraightGeometry& getGeometry ()**
  Get geometry.

  Return the element geometry. Version for non-constant object.

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.

  Return the element geometry Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors.

  Build a FieldWrapper pointing to the multipole and return a pointer to that wrapper.

- **virtual void setField (const BMultipoleField&)**
  Set mulitpole field.

- **virtual ~MultipoleRep ()**

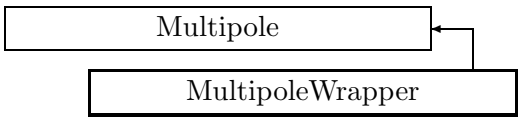

Figure 2.74: Inheritance for class MultipoleWrapper

2.102 class MultipoleWrapper

Representation of a perturbed multipole.
A MultipoleWrapper represents a unique instance of a multipole magnet in the accelerator model. It defines imperfections of the field, related to an "ideal" magnet contained in the wrapper.

Type:	Instantiable
Superclasses:	public Multipole
Include file:	./ComponentWrappers/MultipoleWrapper.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **MultipoleWrapper (Multipole*)**
  Constructor. (see Section 2.102.1)
- **MultipoleWrapper (const MultipoleWrapper&)**
  (see Section 2.102.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to modified multipole. (see Section 2.102.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Make clone. (see Section 2.102.1)

- **virtual ElementBase* copyStructure ()**
  Make structural copy. (see Section 2.102.1)
- **virtual BMultipoleField& errorField ()const**
  Get multipole field error. (see Section 2.102.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const Multipole& getDesign ()const**
  Get design corrector. (see Section 2.102.1)
- **virtual Multipole& getDesign ()**
  Get design corrector. (see Section 2.102.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BMultipoleField& getField ()**
  Get field. (see Section 2.102.1)
- **virtual const BMultipoleField& getField ()const**
  Get field. (see Section 2.102.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.102.1)

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.102.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **double getNormalComponent (int)const**
  Get normal component. (see Section 2.100.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **double getSkewComponent (int)const**
  Get skew component. (see Section 2.100.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.102.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Make wrapper for this multipole. (see Section 2.102.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.102.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.102.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.102.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Remove all wrappers. (see Section 2.102.1)
- **virtual const ElementBase* removeWrappers ()const**
  Remove all wrappers. (see Section 2.102.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **void setNormalComponent (int,double)**
  Set normal component. (see Section 2.100.1)
- **void setSkewComponent (int,double)**
  Set skew component. (see Section 2.100.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~MultipoleWrapper ()**
  (see Section 2.102.1)

2.102.1 Detailed descriptions

Public members

- **MultipoleWrapper (Multipole*)**
  Constructor.

  Constructs a wrapper for its argument. The wrapper is not sharable by default.

- **MultipoleWrapper (const MultipoleWrapper&)**

- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to modified multipole.

- **virtual ElementBase* clone ()const**
  Make clone.

- **virtual ElementBase* copyStructure ()**
  Make structural copy.

- **virtual BMultipoleField& errorField ()const**
  Get multipole field error.

  This method can be used to get or set the error field. The error field offset is mutable, so as to allow changing it in a constant structure.

- **virtual const Multipole& getDesign ()const**
  Get design corrector.

  Version for constant object.

- **virtual Multipole& getDesign ()**
  Get design corrector.

  Version for non-constant object.

- **virtual BMultipoleField& getField ()**
  Get field.

  Return the perturbed field. Version for non-constant object.

- **virtual const BMultipoleField& getField ()const**
  Get field.

  Return the perturbed field. Version for constant object.

- **virtual StraightGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual ElementBase* makeFieldWrapper ()**
  Make wrapper for this multipole.

  Return **this**, since this is already a field wrapper.

- **virtual void makeSharable ()**
  Set sharable flag.

  The whole structure depending on **this** is marked as sharable. After this call a **copyStructure()** call reuses the element.

- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper.

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper.

- **virtual ElementBase* removeWrappers ()**
  Remove all wrappers.

- **virtual const ElementBase* removeWrappers ()const**
  Remove all wrappers.

- **virtual ~MultipoleWrapper ()**

2.103 class NormalForm

Resonance-free normal form.
Find normal form of a truncated Taylor series map. Compute from representation of a map which can be a nil-potent, static or dynamic symplectic map. Implementation of an algorithm described in CENTER M. Berz, E. Forest and J. Irwin,BR Particle Accelerators, 1989, Vol. 24, pp. 91-107. /CENTER

Type:	Instantiable
Include file:	./Algebra/NormalForm.hh

Synopsis (including inherited members)

Public members

- **NormalForm (const VpsInvMap<double>&)**
  Constructor. (see Section 2.103.1)
- **NormalForm ()**
  (see Section 2.103.1)
- **NormalForm (const NormalForm&)**
  (see Section 2.103.1)
- **Matrix<double> anharmonicity ()const**
  Get anharmonicities. (see Section 2.103.1)
- **int degreesOfFreedom ()const**
  Get number of stable degrees of freedom. (see Section 2.103.1)
- **const Vector<complex<double> >& eigenValues ()const**
  Get eigenvalues. (see Section 2.103.1)
- **const Matrix<double>& eigenVectors ()const**
  Get eigenvectors. (see Section 2.103.1)
- **Tps<double> invariant (int)const**
  Get invariant polynomial. (see Section 2.103.1)
- **const Tps<double>& normalForm ()const**
  Get normal-form. (see Section 2.103.1)
- **const Tps<double>& normalisingMap ()const**
  Get normalising map. (see Section 2.103.1)
- **~NormalForm ()**
  (see Section 2.103.1)

Protected members

- **void orderModes (Vector<complex<double> >,Matrix<double>)**
  (see Section 2.103.1)

2.103.1 Detailed descriptions

Public members

- **NormalForm (const VpsInvMap<double>&)**
  Constructor.

  Construct normal-form for **map**.

- **NormalForm ()**

- **NormalForm (const NormalForm&)**

- **Matrix<double> anharmonicity ()const**
  Get anharmonicities.

  Return the anharmonicities as a symmetric matrix.

- **int degreesOfFreedom ()const**
  Get number of stable degrees of freedom.

- **const Vector<complex<double> >& eigenValues ()const**
  Get eigenvalues.

  Return the eigenvalues of the linear part as a complex vector.

- **const Matrix<double>& eigenVectors ()const**
  Get eigenvectors.

  Return the eigenvectors of the linear part in packed form.

- **Tps<double> invariant (int)const**
  Get invariant polynomial.

  Return the invariant polynomial for the mode **i**.

- **const Tps<double>& normalForm ()const**
  Get normal-form.

  Return normal-form map as a Lie transform.

- **const Tps<double>& normalisingMap ()const**
  Get normalising map.

  Return the normalising map as a Lie transform.

- **~NormalForm ()**

Protected members

- **void orderModes (Vector<complex<double> >,Matrix<double>)**

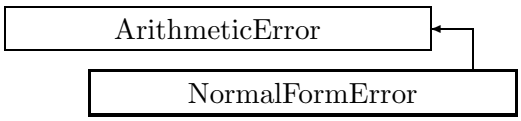

Figure 2.75: Inheritance for class NormalFormError

2.104 class NormalFormError

Normal form error exception.
This exception is thrown, when a normal form routine fails to find the complete normal form, or when it detects an inconsistent call.

Type:	Instantiable
Superclasses:	public ArithmeticError
Include file:	./Utilities/NormalFormError.hh

Synopsis (including inherited members)

Public members

- **NormalFormError (const string&,const string&)**
  The usual constructor. (see Section 2.104.1)
- **NormalFormError (const NormalFormError&)**
  (see Section 2.104.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~NormalFormError ()**
  (see Section 2.104.1)

2.104.1 Detailed descriptions

Public members

- **NormalFormError (const string&,const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  msg the message string identifying the exception
- **NormalFormError (const NormalFormError&)**
- **virtual ~NormalFormError ()**

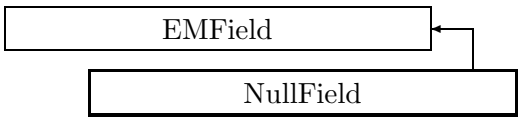

Figure 2.76: Inheritance for class NullField

2.105 class NullField

A zero electromagnetic field.

Type:	Instantiable
Superclasses:	public EMField
Include file:	./Fields/NullField.hh

Synopsis (including inherited members)

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **NullField ()**
  (see Section 2.105.1)
- **NullField (const NullField&)**
  (see Section 2.105.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)
- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)
- **const NullField& operator= (const NullField&)**
  (see Section 2.105.1)

- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)
- **virtual void scale (double)**
  Scale the field. (see Section 2.105.1)
- **virtual ~NullField ()**
  (see Section 2.105.1)

2.105.1 Detailed descriptions

Public members

- **NullField ()**
- **NullField (const NullField&)**
- **const NullField& operator= (const NullField&)**
- **virtual void scale (double)**
  Scale the field.

  Multiply the field by **scalar**. Obviously this method does nothing.
- **virtual ~NullField ()**

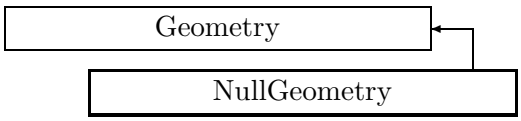

Figure 2.77: Inheritance for class NullGeometry

2.106 class NullGeometry

Geometry representing an identity transform.

Type:	Instantiable
Superclasses:	public Geometry
Include file:	./BeamlineGeometry/NullGeometry.hh

Synopsis (including inherited members)

Public members

- **NullGeometry ()**
  (see Section 2.106.1)
- **NullGeometry (const NullGeometry&)**
  (see Section 2.106.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.106.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.106.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.80.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.80.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.80.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.80.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.80.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.80.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.106.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.80.1)

- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.106.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.106.1)
- **const NullGeometry& operator= (const NullGeometry&)**
  (see Section 2.106.1)
- **const Geometry& operator= (const Geometry&)**
  (see Section 2.80.1)
- **virtual void setElementLength (double)**
  Set geometry length. (see Section 2.80.1)
- **virtual ~NullGeometry ()**
  (see Section 2.106.1)

2.106.1 Detailed descriptions

Public members

- **NullGeometry ()**
- **NullGeometry (const NullGeometry&)**
- **virtual double getArcLength ()const**
  Get arc length.

  Always zero for this class.
- **virtual double getElementLength ()const**
  Get design length.

  Always zero for this class.
- **virtual double getOrigin ()const**
  Get origin position.

  Always zero for this class.
- **virtual Euclid3D getTransform (double,double)const**
  Get transform.

  transform from **fromS** to **toS** is always an identity for this class.
- **virtual Euclid3D getTransform (double)const**
  Get transform.

  Equivalent to getTransform(0.0, s). Return the transform of the local coordinate system from the origin and **s**.
- **const NullGeometry& operator= (const NullGeometry&)**

- **virtual ~NullGeometry ()**

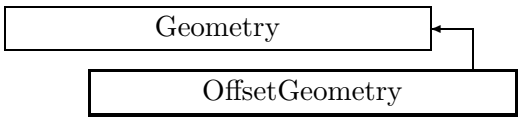

Figure 2.78: Inheritance for class OffsetGeometry

2.107 class OffsetGeometry

A geometry which offset with respect to some global geometry.
OffsetGeometry effectively acts as a bridge between two geometries, designated as the global geometry and the local geometry. The global geometry is used to define a segment of the true global geometry, and so most functions are delegated to the global geometry. The local and global geometries are "offset" with respect to each other by a single transformation (Euclid3D object), which defines the transformation from the global geometry's Local Frame to the local geometry's Local Frame. The two patch functions return the necessary transformations to/from the global/local geometries. OffsetGeometry primary task is to calculate these two patches. Note that when OffsetGeometry is being used to model an alignment error, the local and global geometries can both refer to the same geometry.

Type:	Instantiable
Superclasses:	public Geometry
Include file:	./BeamlineGeometry/OffsetGeometry.hh

Synopsis (including inherited members)

Public members

- **OffsetGeometry (const Geometry&,const Geometry&,const Euclid3D&)**
  Constructor. (see Section 2.107.1)
- **OffsetGeometry (const Geometry&,const Euclid3D&)**
  Constructor. (see Section 2.107.1)
- **OffsetGeometry (const OffsetGeometry&)**
  (see Section 2.107.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.107.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.107.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.107.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.107.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.107.1)

- **virtual double getExit ()const**
  Get exit position. (see Section 2.107.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.107.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.107.1)
- **Euclid3D getGlobalToLocalTransform (double,double)const**
  Transform global to local. (see Section 2.107.1)
- **Euclid3D getGtoL ()const**
  Get displacement. (see Section 2.107.1)
- **virtual double getOrigin ()const**
  Get origin. (see Section 2.107.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.107.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.107.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.107.1)
- **const OffsetGeometry& operator= (const OffsetGeometry&)**
  (see Section 2.107.1)
- **const Geometry& operator= (const Geometry&)**
  (see Section 2.80.1)
- **virtual void setElementLength (double)**
  Set geometry length. (see Section 2.80.1)
- **void setGtoL (const Euclid3D&)**
  Set displacement. (see Section 2.107.1)
- **virtual ~OffsetGeometry ()**
  (see Section 2.107.1)

2.107.1 Detailed descriptions

Public members

- **OffsetGeometry (const Geometry&,const Geometry&,const Euclid3D&)**
  Constructor.

  Assign the **global** and **local** geometries, and the displacement **euclid** between their origins.
- **OffsetGeometry (const Geometry&,const Euclid3D&)**
  Constructor.

  Both global and local geometries are set to **geom**, and the displacement is **euclid**.

- **OffsetGeometry (const OffsetGeometry&)**

- **virtual double getArcLength ()const**
  Get arc length.

  Return the length of the global geometry, measured along its design arc.

- **virtual double getElementLength ()const**
  Get design length.

  Return the design length of the global geometry, measured along its design polygone.

- **virtual double getEntrance ()const**
  Get entrance position.

  Return the arc length from the origin to the entrance of the global geometry (non-positive).

- **virtual Euclid3D getEntranceFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getEntrance()). Return the transform of the local coordinate system from the global origin to the entrance of the element.

- **virtual Euclid3D getEntrancePatch ()const**
  Get patch.

  Return the entrance patch (transformation) which is used to transform the global geometry to the local geometry at entrance.

- **virtual double getExit ()const**
  Get exit position.

  Return the arc length from the origin to the exit of the global geometry (non-negative).

- **virtual Euclid3D getExitFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getExit()). Return the transform of the local coordinate system from the global origin to the exit of the element.

- **virtual Euclid3D getExitPatch ()const**
  Get patch.

  Returns the entrance patch (transformation) which is used to transform the local geometry to the global geometry at exit.

- **Euclid3D getGlobalToLocalTransform (double,double)const**
  Transform global to local.

  Special OffsetGeometry function which calculates the transformation from position **globalS** on the global geometry to position **localS** on the local geometry.

- **Euclid3D getGtoL ()const**
  Get displacement.

  Return the displacement from the global to the local origin.

- **virtual double getOrigin ()const**
  Get origin.

  Return the arc length from the entrance to the origin of the global geometry (non-negative).

- **virtual Euclid3D getTotalTransform ()const**
  Get transform.

  Equivalent to getTransform(getEntrance(), getExit()). Return the transform of the global coordinate system from the entrance to the exit of the element.

- **virtual Euclid3D getTransform (double,double)const**
  Get transform.

  Return the transform of the global coordinate system from the position **fromS** to the position **toS**.

- **virtual Euclid3D getTransform (double)const**
  Get transform.

  Equivalent to getTransform(0.0, s). Return the transform of the local coordinate system from the global origin at **s**.

- **const OffsetGeometry& operator= (const OffsetGeometry&)**

- **void setGtoL (const Euclid3D&)**
  Set displacement.

  Assign the displacement from the global to the local origin.

- **virtual ~OffsetGeometry ()**

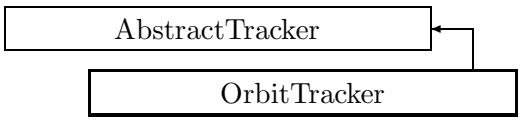

Figure 2.79: Inheritance for class OrbitTracker

2.108 class OrbitTracker

Track closed orbit.
Uses a thick lens method to find the orbit for each element.

Type:	Instantiable
Superclasses:	public AbstractTracker
Include file:	./Algorithms/OrbitTracker.hh

Synopsis (including inherited members)

Public members

- **OrbitTracker (const Beamline&,const PartData&,bool,bool)**
  Constructor. (see Section 2.108.1)
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **const FVector<double,6>& getOrbit ()const**
  Return the current orbit. (see Section 2.108.1)
- **void setOrbit (const FVector<double,6>)**
  Reset the current orbit. (see Section 2.108.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper.. (see Section 2.108.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam. (see Section 2.108.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.108.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.108.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector. (see Section 2.108.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.37.1)

- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic. (see Section 2.108.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift. (see Section 2.108.1)
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.37.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson. (see Section 2.108.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.37.1)
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker. (see Section 2.108.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor. (see Section 2.108.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole. (see Section 2.108.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.37.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.108.1)
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend. (see Section 2.108.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.37.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity. (see Section 2.108.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole. (see Section 2.108.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend. (see Section 2.108.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.37.1)
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator. (see Section 2.108.1)

- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum. (see Section 2.108.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid. (see Section 2.108.1)
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.37.1)
- **virtual ~OrbitTracker ()**
  (see Section 2.108.1)

2.108.1 Detailed descriptions

Public members

- **OrbitTracker (const Beamline&,const PartData&,bool,bool)**
  Constructor.

  The beam line to be tracked is **bl**. The particle reference data are taken from **data**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.
- **const FVector<double,6>& getOrbit ()const**
  Return the current orbit.
- **void setOrbit (const FVector<double,6>)**
  Reset the current orbit.
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper..
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam.
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator.
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component.
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector.
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic.
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift.
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson.
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker.

- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor.
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole.
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch.
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend.
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity.
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole.
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend.
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator.
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum.
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid.
- **virtual ~OrbitTracker ()**

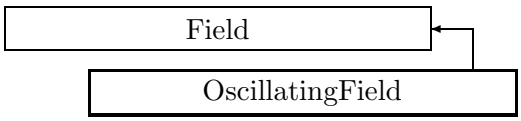

Figure 2.80: Inheritance for class OscillatingField

2.109 template class OscillatingField <class>

An oscillating electromagnetic field.
The field is derived from the corresponding static field. Care must be taken when the static field is not homogeneous, the oscillating values may be incorrect, as they may not fulfill Maxwell's equations. p Note that this class is derived from its template parameter, it inherits the methods and data from that class as well.

Type:	Instantiable
Superclasses:	public Field
Include file:	./Fields/OscillatingField.hh

Synopsis (including inherited members)

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.109.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.109.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.109.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.109.1)
- **OscillatingField ()**
  Default constructor. (see Section 2.109.1)
- **virtual double getFrequency ()const**
  Return the RF frequency in Hz. (see Section 2.109.1)
- **virtual double getPhase ()const**
  Return the RF phase in rad. (see Section 2.109.1)
- **virtual void setFrequency (double)**
  Assign the RF frequency in Hz. (see Section 2.109.1)
- **virtual void setPhase (double)**
  Assign the RF phase in rad. (see Section 2.109.1)
- **virtual ~OscillatingField ()**
  (see Section 2.109.1)

2.109.1 Detailed descriptions

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field.

  Return the constant part of the magnetic field in point **P**.

- **virtual BVector Bfield (const Point3D&,double)const**
  Get field.

  Return the magnetic field at time **t** in point **P**.

- **virtual EVector Efield (const Point3D&)const**
  Get field.

  Return the constant part of the electric field in point **P**.

- **virtual EVector Efield (const Point3D&,double)const**
  Get field.

  Return the electric field at time **t** in point **P**.

- **OscillatingField ()**
  Default constructor.

  Constructs a null field.

- **virtual double getFrequency ()const**
  Return the RF frequency in Hz.

- **virtual double getPhase ()const**
  Return the RF phase in rad.

- **virtual void setFrequency (double)**
  Assign the RF frequency in Hz.

- **virtual void setPhase (double)**
  Assign the RF phase in rad.

- **virtual ~OscillatingField ()**

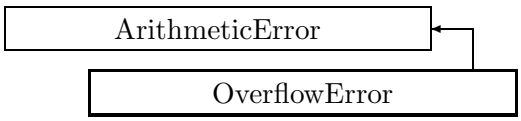

Figure 2.81: Inheritance for class OverflowError

2.110 class OverflowError

Overflow exception.
This exception is thrown, when a function detects an arithmetic overflow.

Type:	Instantiable
Superclasses:	public ArithmeticError
Include file:	./Utilities/OverflowError.hh

Synopsis (including inherited members)

Public members

- **OverflowError (const string&)**
  The usual constructor. (see Section 2.110.1)
- **OverflowError (const OverflowError&)**
  (see Section 2.110.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~OverflowError ()**
  (see Section 2.110.1)

2.110.1 Detailed descriptions

Public members

- **OverflowError (const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  Construction/destruction.
- **OverflowError (const OverflowError&)**
- **virtual ~OverflowError ()**

2.111 template class OwnPtr <class>

A pointer which owns the object pointed at.
When the pointer goes out of scope, the pointee is deleted. When an OwnPtr is assigned the ownership is transferred to the target.

Type:	Instantiable
Include file:	./MemoryManagement/OwnPtr.hh

Synopsis (including inherited members)

Public members

- **OwnPtr ()**
  Default constructor. (see Section 2.111.1)
- **OwnPtr (const OwnPtr&)**
  Copy constructor. (see Section 2.111.1)
- **OwnPtr (Object*)**
  Constructor. (see Section 2.111.1)
- **bool isValid ()const**
  Test for validity. (see Section 2.111.1)
- **Object& operator* ()const**
  Dereferencing operator. (see Section 2.111.1)
- **Object* operator-> ()const**
  Delegation operator. (see Section 2.111.1)
- **OwnPtr& operator= (const OwnPtr&)**
  Assign. (see Section 2.111.1)
- **OwnPtr& operator= (Object*)**
  Assign. (see Section 2.111.1)
- **Object* release ()**
  Release ownership. (see Section 2.111.1)
- **~OwnPtr ()**
  Destructor. (see Section 2.111.1)

2.111.1 Detailed descriptions

Public members

- **OwnPtr ()**
  Default constructor.

  Set the pointer to NULL.
- **OwnPtr (const OwnPtr&)**
  Copy constructor.

  Transmits ownership.

- **OwnPtr (Object*)**
  Constructor.

  Construct pointer to an object just created by "new". Grabs ownership.

- **bool isValid ()const**
  Test for validity.

  Return true, if the pointer is not NULL.

- **Object& operator* ()const**
  Dereferencing operator.

  Return a reference to the object.

- **Object* operator-> ()const**
  Delegation operator.

  Return a C-type pointer to the object.

- **OwnPtr& operator= (const OwnPtr&)**
  Assign.

  Transmit ownership to copy.

- **OwnPtr& operator= (Object*)**
  Assign.

  Grab the ownership of an object just created by "new".

- **Object* release ()**
  Release ownership.

  Return built-in pointer. The calling program must take over the ownership.

- **~OwnPtr ()**
  Destructor.

  Delete the pointed-at object, if the pointer is non-null.

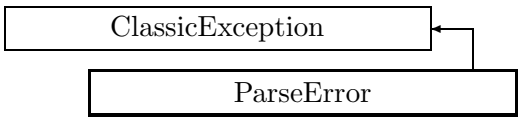

Figure 2.82: Inheritance for class ParseError

2.112 class ParseError

Parse exception.
This exception is thrown by the CLASSIC parser whein it detects an input format error.

Type:	Instantiable
Superclasses:	public ClassicException
Include file:	./Utilities/ParseError.hh

Synopsis (including inherited members)

Public members

- **ParseError (const string&,const string&)**
  The usual constructor. (see Section 2.112.1)
- **ParseError (const ParseError&)**
  (see Section 2.112.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~ParseError ()**
  (see Section 2.112.1)

2.112.1 Detailed descriptions

Public members

- **ParseError (const string&,const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  msg the message string identifying the exception

  Construction/destruction.
- **ParseError (const ParseError&)**
- **virtual ~ParseError ()**

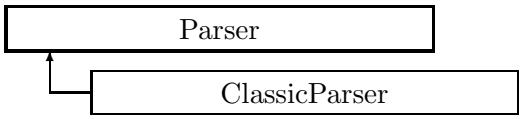

Figure 2.83: Inheritance for class Parser

2.113 class Parser

Interface for abstract language parser.

Type:	abstract
Include file:	./Parser/Parser.hh

Synopsis (including inherited members)

Public members

- **Parser ()**
  (see Section 2.113.1)
- **virtual void parse (Statement&)const**
  Parse and execute the statement. (see Section 2.113.1)
- **virtual Statement* readStatement (TokenStream*)const**
  Read complete statement from token stream (see Section 2.113.1)
- **virtual void run (TokenStream*)const**
  Read statements and parse. (see Section 2.113.1)
- **virtual ~Parser ()**
  (see Section 2.113.1)

2.113.1 Detailed descriptions

Public members

- **Parser ()**
- **virtual void parse (Statement&)const**
  Parse and execute the statement.
- **virtual Statement* readStatement (TokenStream*)const**
  Read complete statement from token stream
- **virtual void run (TokenStream*)const**
  Read statements and parse.

  Read one statement at a time on token stream, parse it, and execute it.
- **virtual ~Parser ()**

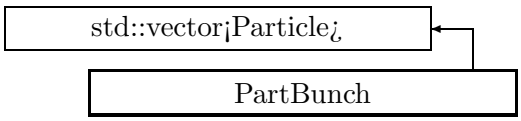

Figure 2.84: Inheritance for class PartBunch

2.114 class PartBunch

Particle Bunch.
A representation of a particle bunch as a vector of particles.

Type:	Instantiable
Superclasses:	public std::vector<Particle>
Include file:	./Algorithms/PartBunch.hh

Synopsis (including inherited members)

Public members

- **PartBunch ()**
  Default constructor. (see Section 2.114.1)
- **PartBunch (const std::vector<Particle>&)**
  Conversion. (see Section 2.114.1)
- **PartBunch (const PartBunch&)**
  (see Section 2.114.1)
- **void beamEllipsoid (FVector<double,6>&,FMatrix<double,6,6>&)const**
  Return bunch distribution. (see Section 2.114.1)
- **void maximumAmplitudes (const FMatrix<double,6,6>&,double&,double&)const**
  Return maximum amplitudes. (see Section 2.114.1)
- **˜PartBunch ()**
  (see Section 2.114.1)

2.114.1 Detailed descriptions

Public members

- **PartBunch ()**
  Default constructor.

  Construct empty bunch.
- **PartBunch (const std::vector<Particle>&)**
  Conversion.
- **PartBunch (const PartBunch&)**

- **void beamEllipsoid (FVector<double,6>&,FMatrix<double,6,6>&)const Return bunch distribution.**

  Return the bunch centroid in **centroid**, and the second moments in **moment**.

- **void maximumAmplitudes (const FMatrix<double,6,6>&,double&,double&)const Return maximum amplitudes.**

  The matrix **D** is used to normalise the first two modes. The maximum normalised amplitudes for these modes are stored in **axmax** and **aymax**.

- **~PartBunch ()**

2.115 class PartData

Particle reference data.
This class encapsulates the reference data for a beam:

charge per particle expressed in proton charges,

mass per particle expressed in eV,

reference momentum per particle expressed in eV.

The copy constructor, destructor, and assignment operator generated by the compiler perform the correct operation. For speed reasons they are not implemented.

Type:	Instantiable
Include file:	./Algorithms/PartData.hh

Synopsis (including inherited members)

Public members

- **PartData (double,double,double)**
  Constructor. (see Section 2.115.1)
- **PartData ()**
  (see Section 2.115.1)
- **double getBeta ()const**
  The relativistic beta per particle. (see Section 2.115.1)
- **double getE ()const**
  The constant reference Energy per particle. (see Section 2.115.1)
- **double getGamma ()const**
  The relativistic gamma per particle. (see Section 2.115.1)
- **double getM ()const**
  The constant mass per particle. (see Section 2.115.1)
- **double getP ()const**
  The constant reference momentum per particle. (see Section 2.115.1)
- **double getQ ()const**
  The constant charge per particle. (see Section 2.115.1)
- **void setBeta (double)**
  Set beta. (see Section 2.115.1)
- **void setE (double)**
  Set reference energy. (see Section 2.115.1)
- **void setGamma (double)**
  Set gamma. (see Section 2.115.1)
- **void setP (double)**
  Set reference momentum. (see Section 2.115.1)

Protected members

- **double beta**
  (see Section 2.115.1)
- **double charge**
  (see Section 2.115.1)
- **double gamma**
  (see Section 2.115.1)
- **double mass**
  (see Section 2.115.1)

2.115.1 Detailed descriptions

Public members

- **PartData (double,double,double)**
  Constructor.

  Inputs are:

  charge The charge per particle in proton charges.

  mass The particle mass in eV.

  momentum The reference momentum per particle in eV.
- **PartData ()**
- **double getBeta ()const**
  The relativistic beta per particle.
- **double getE ()const**
  The constant reference Energy per particle.
- **double getGamma ()const**
  The relativistic gamma per particle.
- **double getM ()const**
  The constant mass per particle.
- **double getP ()const**
  The constant reference momentum per particle.
- **double getQ ()const**
  The constant charge per particle.
- **void setBeta (double)**
  Set beta.

  Input is the relativistic beta = v/c.

- **void setE (double)**
  Set reference energy.

  Input is the energy in eV.

- **void setGamma (double)**
  Set gamma.

  Input is the relativistic gamma = E/(m*c*c).

- **void setP (double)**
  Set reference momentum.

  Input is the momentum in eV.

Protected members

- **double beta**

- **double charge**

- **double gamma**

- **double mass**

2.116 class Particle

Particle position.
This class represents the canonical coordinates of a particle. P NOTE. The order of phase space coordinates is, as opposed to the original CLASSIC definition: CENTER X = 0, PX = 1, Y = 2, PY = 3, T = 4, PT = 5. /CENTER The copy constructor, destructor, and assignment operator generated by the compiler perform the correct operation. For speed reasons they are not implemented.

Type:	Instantiable
Include file:	./Algorithms/Particle.hh

Synopsis (including inherited members)

Public members

- **Particle (double,double,double,double,double,double)**
  Constructor. (see Section 2.116.1)
- **Particle ()**
  (see Section 2.116.1)
- **double& operator[] (int)**
  Get coordinate. (see Section 2.116.1)
- **double operator[] (int)const**
  Get coordinate. (see Section 2.116.1)
- **double& pt ()**
  Get reference to relative momentum error (no dimension). (see Section 2.116.1)
- **double pt ()const**
  Get relative momentum error (no dimension). (see Section 2.116.1)
- **double& px ()**
  Get reference to horizontal momentum (no dimension). (see Section 2.116.1)
- **double px ()const**
  Get horizontal momentum (no dimension). (see Section 2.116.1)
- **double& py ()**
  Get reference to vertical momentum (no dimension). (see Section 2.116.1)
- **double py ()const**
  Get vertical momentum (no dimension). (see Section 2.116.1)
- **double& t ()**
  Get reference to longitudinal displacement c*t in m. (see Section 2.116.1)
- **double t ()const**
  Get longitudinal displacement c*t in m. (see Section 2.116.1)
- **double& x ()**
  Get reference to horizontal position in m. (see Section 2.116.1)

- **double x ()const**
  Get horizontal position in m. (see Section 2.116.1)
- **double& y ()**
  Get reference to vertical displacement in m. (see Section 2.116.1)
- **double y ()const**
  Get vertical displacement in m. (see Section 2.116.1)

2.116.1 Detailed descriptions

Public members

- **Particle (double,double,double,double,double,double)**
  Constructor.

  Construct particle with the given coordinates.
- **Particle ()**

- **double& operator[] (int)**
  Get coordinate.

  Access coordinate by index. Note above order of phase space coordinates!
- **double operator[] (int)const**
  Get coordinate.

  Access coordinate by index for constant particle.
- **double& pt ()**
  Get reference to relative momentum error (no dimension).
- **double pt ()const**
  Get relative momentum error (no dimension).
- **double& px ()**
  Get reference to horizontal momentum (no dimension).
- **double px ()const**
  Get horizontal momentum (no dimension).
- **double& py ()**
  Get reference to vertical momentum (no dimension).
- **double py ()const**
  Get vertical momentum (no dimension).
- **double& t ()**
  Get reference to longitudinal displacement c*t in m.
- **double t ()const**
  Get longitudinal displacement c*t in m.

- **double& x ()**
  Get reference to horizontal position in m.
- **double x ()const**
  Get horizontal position in m.
- **double& y ()**
  Get reference to vertical displacement in m.
- **double y ()const**
  Get vertical displacement in m.

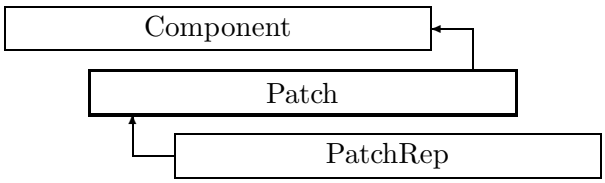

Figure 2.85: Inheritance for class Patch

2.117 class Patch

Interface for a geometric patch.
Class Patch defines the abstract interface for geometric patches, i.e. a geometric transform representing a misalignment of local coordinate systems of two subsequent element with each other.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Patch.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Patch (const string&)**
  Constructor with given name. (see Section 2.117.1)
- **Patch ()**
  (see Section 2.117.1)
- **Patch (const Patch&)**
  (see Section 2.117.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to patch. (see Section 2.117.1)

- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)

- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual const Euclid3D& getPatch ()const**
  Get patch transform. (see Section 2.117.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)

- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Patch ()**
  (see Section 2.117.1)

2.117.1 Detailed descriptions

Public members

- **Patch (const string&)**
  Constructor with given name.
- **Patch ()**
- **Patch (const Patch&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to patch.

- **virtual const Euclid3D& getPatch ()const**
  Get patch transform.
- **virtual ~Patch ()**

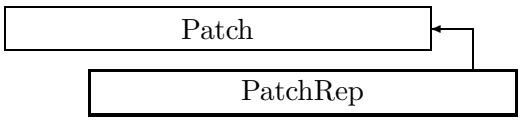

Figure 2.86: Inheritance for class PatchRep

2.118 class PatchRep

Representation for a geometry patch.

Type:	Instantiable
Superclasses:	public Patch
Include file:	./BeamlineCore/PatchRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **PatchRep (const string&)**
  Constructor with given name. (see Section 2.118.1)
- **PatchRep ()**
  (see Section 2.118.1)
- **PatchRep (const PatchRep&)**
  (see Section 2.118.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to patch. (see Section 2.117.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.118.1)

- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.118.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual NullField& getField ()**
  Get field. (see Section 2.118.1)
- **virtual const NullField& getField ()const**
  Get field. (see Section 2.118.1)
- **virtual NullGeometry& getGeometry ()**
  Get geometry. (see Section 2.118.1)
- **virtual const NullGeometry& getGeometry ()const**
  Get geometry. (see Section 2.118.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.118.1)

- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual const Euclid3D& getPatch ()const**
  Get patch. (see Section 2.118.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.118.1)
- **double getVX ()const**
  Get rotation. (see Section 2.118.1)
- **double getVY ()const**
  Get rotation. (see Section 2.118.1)
- **double getVZ ()const**
  Get rotation. (see Section 2.118.1)
- **double getX ()const**
  Get displacement. (see Section 2.118.1)
- **double getY ()const**
  Get displacement. (see Section 2.118.1)
- **double getZ ()const**
  Get displacement. (see Section 2.118.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)

- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **void setPatch (const Euclid3D&)**
  Set patch. (see Section 2.118.1)
- **void setPatch (double,double,double,double,double,double)**
  Set patch. (see Section 2.118.1)
- **void setVX (double)**
  Set rotation. (see Section 2.118.1)
- **void setVY (double)**
  Set rotation. (see Section 2.118.1)
- **void setVZ (double)**
  Set rotation. (see Section 2.118.1)

- **void setX (double)**
  Set displacement. (see Section 2.118.1)
- **void setY (double)**
  Set displacement. (see Section 2.118.1)
- **void setZ (double)**
  Set displacement. (see Section 2.118.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~PatchRep ()**
  (see Section 2.118.1)

2.118.1 Detailed descriptions

Public members

- **PatchRep (const string&)**
  Constructor with given name.
- **PatchRep ()**
- **PatchRep (const PatchRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.
- **virtual NullField& getField ()**
  Get field.

  Version for non-constant object.
- **virtual const NullField& getField ()const**
  Get field.

  Version for constant object.

- **virtual NullGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const NullGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual const Euclid3D& getPatch ()const**
  Get patch.

  Return the geometric patch to transform from input to output.

- **virtual const string& getType ()const**
  Get element type string.

- **double getVX ()const**
  Get rotation.

  Return the rotation around the x-axis

- **double getVY ()const**
  Get rotation.

  Return the rotation around the y-axis

- **double getVZ ()const**
  Get rotation.

  Return the rotation around the z-axis

- **double getX ()const**
  Get displacement.

  Return the x-displacement.

- **double getY ()const**
  Get displacement.

  Return the y-displacement.

- **double getZ ()const**
  Get displacement.

  Return the z-displacement.

- **void setPatch (const Euclid3D&)**
  Set patch.

  Assign the geometric patch to transform from input to output.

- **void setPatch (double,double,double,double,double,double)**
  Set patch.

  Assign the geometric patch to transform from input to output, using the displacement (x,y,z) and the rotation vector (vx,vy,vz).

- **void setVX (double)**
  Set rotation.

  Assign the rotation around the x-axis

- **void setVY (double)**
  Set rotation.

  Assign the rotation around the y-axis

- **void setVZ (double)**
  Set rotation.

  Assign the rotation around the z-axis

- **void setX (double)**
  Set displacement.

  Assign the x-displacement.

- **void setY (double)**
  Set displacement.

  Assign the y-displacement.

- **void setZ (double)**
  Set displacement.

  Assign the z-displacement.

- **virtual ~PatchRep ()**

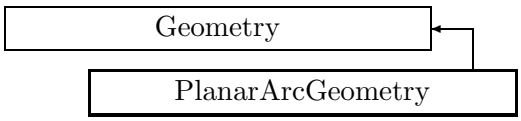

Figure 2.87: Inheritance for class PlanarArcGeometry

2.119 class PlanarArcGeometry

A simple arc in the XZ plane.
A PlanarArcGeometry object represents a Geometry which is as simple circular arc in the local XZ plane with no torsion. The origin is defined at the centre of the geometry, and the geometry extends from -length/2 to +length/2. Transformations generated by this geometry are characterised by a displacement in the XZ plane and a pure Y rotation.
This geometry can have two states:

Length L is non-zero: The geometry is defined as a finite arc in the XZ plane. In this state, the following changes are allowed:

Setting the angle phi: recompute the curvature as phi / L.

Setting the curvature h: recompute the angle as h * L.

Setting the length: if non-zero, recompute the angle as h * L, otherwise set h to zero.

Length is zero, implying curvature also zero: The geometry is defined as sharp bend by phi in the XZ plane. In this state, the following changes are allowed:

Setting the angle phi: leave both h and L as zero.

Setting the curvature h: not accepted.

Setting the length: if non-zero, recompute the angle as h * L.

Type:	Instantiable
Superclasses:	public Geometry
Include file:	./BeamlineGeometry/PlanarArcGeometry.hh

Synopsis (including inherited members)

Public members

- **PlanarArcGeometry (double,double)**
  Constructor. (see Section 2.119.1)
- **PlanarArcGeometry (double)**
  Constructor. (see Section 2.119.1)
- **PlanarArcGeometry (const PlanarArcGeometry&)**
  (see Section 2.119.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.119.1)

- **double getBendAngle ()const**
  Get angle. (see Section 2.119.1)
- **double getCurvature ()const**
  Get curvature. (see Section 2.119.1)
- **virtual double getElementLength ()const**
  Get element length. (see Section 2.119.1)
- **double getEntrance ()const**
  Get entrance. (see Section 2.119.1)
- **Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.119.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.80.1)
- **double getExit ()const**
  Get exit. (see Section 2.119.1)
- **Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.119.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.80.1)
- **virtual double getOrigin ()const**
  Get origin. (see Section 2.119.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.119.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.119.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.119.1)
- **const PlanarArcGeometry& operator= (const PlanarArcGeometry&)**
  (see Section 2.119.1)
- **const Geometry& operator= (const Geometry&)**
  (see Section 2.80.1)
- **virtual void setBendAngle (double)**
  Set angle. (see Section 2.119.1)
- **void setCurvature (double)**
  Set curvature. (see Section 2.119.1)
- **virtual void setElementLength (double)**
  Set length. (see Section 2.119.1)
- **virtual ~PlanarArcGeometry ()**
  (see Section 2.119.1)

2.119.1 Detailed descriptions

Public members

- **PlanarArcGeometry (double,double)**
  Constructor.

  Build PlanarArcGeometry from length **l** and curvature **h**.

- **PlanarArcGeometry (double)**
  Constructor.

  Build PlanarArcGeometry from length zero and angle **phi**.

- **PlanarArcGeometry (const PlanarArcGeometry&)**

- **virtual double getArcLength ()const**
  Get arc length.

  Return arc length along the design arc.

- **double getBendAngle ()const**
  Get angle.

- **double getCurvature ()const**
  Get curvature.

- **virtual double getElementLength ()const**
  Get element length.

  Return element length measured along the design arc.

- **double getEntrance ()const**
  Get entrance.

  Return the arc length from the origin to the entrance of the geometry (non-positive).

- **Euclid3D getEntranceFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getEntrance()). Return the transform of the local coordinate system from the origin to the entrance of the element.

- **double getExit ()const**
  Get exit.

  Return the arc length from the origin to the exit of the geometry (non-negative).

- **Euclid3D getExitFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getExit()). Return the transform of the local coordinate system from the origin to the exit of the element.

- **virtual double getOrigin ()const**
  Get origin.

  Return the distance from the entrance of the geometry to the origin (non-negative).

- **virtual Euclid3D getTotalTransform ()const**
  Get transform.

  Equivalent to getTransform(getEntrance(), getExit()). Return the transform of the local coordinate system from the entrance to the exit of the element.

- **virtual Euclid3D getTransform (double,double)const**
  Get transform.

  Return the transform of the local coordinate system from the position **fromS** to the position **toS**.

- **virtual Euclid3D getTransform (double)const**
  Get transform.

  Equivalent to getTransform(0.0, s). Return the transform of the local coordinate system from the origin and **s**.

- **const PlanarArcGeometry& operator= (const PlanarArcGeometry&)**

- **virtual void setBendAngle (double)**
  Set angle.

- **void setCurvature (double)**
  Set curvature.

- **virtual void setElementLength (double)**
  Set length.

  Assign the arc length for this class.

- **virtual ~PlanarArcGeometry ()**

2.120 class Point3D

A point in 3 dimensions.

Type:	Instantiable
Include file:	./Fields/EMField.hh

Synopsis (including inherited members)

Public members

- **Point3D (double,double,double)**
  Constructor. (see Section 2.120.1)
- **double getX ()const**
  Return coordinate. (see Section 2.120.1)
- **double getY ()const**
  Return coordinate. (see Section 2.120.1)
- **double getZ ()const**
  Return coordinate. (see Section 2.120.1)

2.120.1 Detailed descriptions

Public members

- **Point3D (double,double,double)**
  Constructor.

  Construct the point from its coordinates (x,y,z) in m.
- **double getX ()const**
  Return coordinate.

  Return the x-coordinate for the point in m.
- **double getY ()const**
  Return coordinate.

  Return the y-coordinate for the point in m.
- **double getZ ()const**
  Return coordinate.

  Return the z-coordinate for the point in m.

2.121 template class Pointer <class>

Reference-counted pointer.
This template class implements a pointer with reference counting in collaboration with the class RCObject. It is modelled after the class RCPtr described in: center Scott Meyers, More Effective C++, Addison Wesley, 1996, pg. 195. /center When the pointer goes out of scope, the pointee's reference count is decremented, and, if it becomes zero, the pointee is deleted. When a Pointer is assigned, the pointee's reference count is incremented.

Type:	Instantiable
Include file:	./MemoryManagement/Pointer.hh

Synopsis (including inherited members)

Public members

- **Pointer ()**
  Default constructor. (see Section 2.121.1)
- **Pointer (const Pointer&)**
  Copy constructor. (see Section 2.121.1)
- **Pointer (Object*)**
  Constructor. (see Section 2.121.1)
- **bool isValid ()const**
  Test for validity. (see Section 2.121.1)
- **bool operator!= (const Pointer&)const**
  Pointer inequality. (see Section 2.121.1)
- **Object& operator* ()const**
  Dereferencing operator. (see Section 2.121.1)
- **Object* operator-> ()const**
  Delegation operator. (see Section 2.121.1)
- **Pointer& operator= (const Pointer&)**
  Assign. (see Section 2.121.1)
- **Pointer& operator= (Object*)**
  Assign. (see Section 2.121.1)
- **bool operator== (const Pointer&)const**
  Pointer equality. (see Section 2.121.1)
- **void unique ()**
  Force unique. (see Section 2.121.1)
- **~Pointer ()**
  Destructor. (see Section 2.121.1)

2.121.1 Detailed descriptions

Public members

- **Pointer ()**
  Default constructor.

  Set the pointer to NULL.

- **Pointer (const Pointer&)**
  Copy constructor.

  Increment reference count of pointee.

- **Pointer (Object*)**
  Constructor.

  Grab an object just created by "new" and set its reference count to one.

- **bool isValid ()const**
  Test for validity.

  Return true, if the pointer is not NULL.

- **bool operator!= (const Pointer&)const**
  Pointer inequality.

- **Object& operator* ()const**
  Dereferencing operator.

  Return a reference to the object.

- **Object* operator-> ()const**
  Delegation operator.

  Return a C-type pointer to the object.

- **Pointer& operator= (const Pointer&)**
  Assign.

  Increment reference count of pointee.

- **Pointer& operator= (Object*)**
  Assign.

  Grab an object just created by "new" and set its reference count to one.

- **bool operator== (const Pointer&)const**
  Pointer equality.

- **void unique ()**
  Force unique.

  Copy the object and make *this point to it.

- **~Pointer ()**
  Destructor.

  Decrement reference count and delete object if it reaches zero.

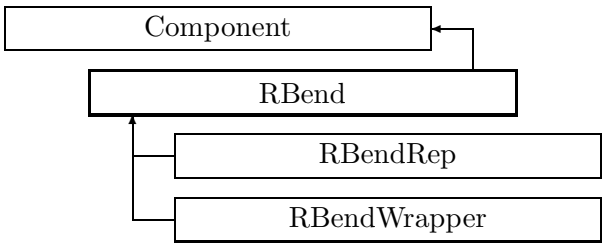

Figure 2.88: Inheritance for class RBend

2.122 class RBend

Interface for rectangular bend.
Class RBend defines the abstract interface for rectangular bend magnets. A rectangular bend magnet has a rectilinear geometry about which its multipole components are specified.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/RBend.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **RBend (const string&)**
  Constructor with given name. (see Section 2.122.1)
- **RBend ()**
  (see Section 2.122.1)
- **RBend (const RBend&)**
  (see Section 2.122.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to RBend. (see Section 2.122.1)

- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getB ()const**
  Get dipole field of RBend. (see Section 2.122.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getEntryFaceCurvature ()const**
  Get entry pole face curvature. (see Section 2.122.1)
- **virtual double getEntryFaceRotation ()const**
  Get pole entry face rotation. (see Section 2.122.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual double getExitFaceCurvature ()const**
  Get exit pole face curvature. (see Section 2.122.1)
- **virtual double getExitFaceRotation ()const**
  Get exit pole face rotation. (see Section 2.122.1)

- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BMultipoleField& getField ()**
  Get multipole expansion of field. (see Section 2.122.1)
- **virtual const BMultipoleField& getField ()const**
  Get multipole expansion of field. (see Section 2.122.1)
- **virtual RBendGeometry& getGeometry ()**
  Get RBend geometry. (see Section 2.122.1)
- **virtual const RBendGeometry& getGeometry ()const**
  Get RBend geometry (see Section 2.122.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **double getNormalComponent (int)const**
  Get normal component. (see Section 2.122.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **double getSkewComponent (int)const**
  Get skew component. (see Section 2.122.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)

- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **void setNormalComponent (int,double)**
  Set normal component. (see Section 2.122.1)
- **void setSkewComponent (int,double)**
  Set skew component. (see Section 2.122.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)

- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~RBend ()**
  (see Section 2.122.1)

2.122.1 Detailed descriptions

Public members

- **RBend (const string&)**
  Constructor with given name.
- **RBend ()**
- **RBend (const RBend&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to RBend.
- **virtual double getB ()const**
  Get dipole field of RBend.
- **virtual double getEntryFaceCurvature ()const**
  Get entry pole face curvature.

  Return the curvature of the entry pole face. A positive curvature creates a convex pole face.
- **virtual double getEntryFaceRotation ()const**
  Get pole entry face rotation.

  Return the rotation of the entry pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.
- **virtual double getExitFaceCurvature ()const**
  Get exit pole face curvature.

  Return the curvature of the exit pole face. A positive curvature creates a convex pole face.
- **virtual double getExitFaceRotation ()const**
  Get exit pole face rotation.

  Return the rotation of the exit pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.
- **virtual BMultipoleField& getField ()**
  Get multipole expansion of field.

  Version for non-constant object.

- **virtual const BMultipoleField& getField ()const**
  Get multipole expansion of field.

  Version for constant object.

- **virtual RBendGeometry& getGeometry ()**
  Get RBend geometry.

  Version for non-constant object.

- **virtual const RBendGeometry& getGeometry ()const**
  Get RBend geometry

  Version for constant object.

- **double getNormalComponent (int)const**
  Get normal component.

  Return the normal component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the return value is zero.

- **double getSkewComponent (int)const**
  Get skew component.

  Return the skew component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the return value is zero.

- **void setNormalComponent (int,double)**
  Set normal component.

  Set the normal component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the component is created.

- **void setSkewComponent (int,double)**
  Set skew component.

  Set the skew component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the component is created.

- **virtual ~RBend ()**

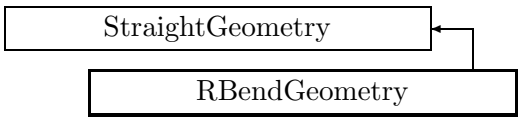

Figure 2.89: Inheritance for class RBendGeometry

2.123 class RBendGeometry

The geometry for a RBend element.
It is a Geometry wrapper which adds two rotations by alpha/2 (local y-axis rotations) to the entrance and exit planes of a StraightGeometry. The y-rotations become part of the global geometry definition. P NOTE: in general the transformations returned include the effects of the y-rotations when the required point specifies the entrance or exit point. Requests for transformations within the geometry (i.e. from s1 to s2, where s1 and/or s2 are not the entrance or exit planes) do not contain the y-rotations. P A RBendGeometry can be seen as an OffsetGeometry, whose global geometry is a PlanarArcGeometry, and whose local geometry is a StraightGeometry.

Type:	Instantiable
Superclasses:	public StraightGeometry
Include file:	./BeamlineGeometry/RBendGeometry.hh

Synopsis (including inherited members)

Public members

- **RBendGeometry (double,double)**
  Constructor. (see Section 2.123.1)
- **RBendGeometry (const RBendGeometry&)**
  (see Section 2.123.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.123.1)
- **virtual double getBendAngle ()const**
  Get angle. (see Section 2.123.1)
- **virtual double getElementLength ()const**
  Get element length. (see Section 2.123.1)
- **double getEntrance ()const**
  Get entrance. (see Section 2.151.1)
- **Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.123.1)
- **Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.123.1)
- **double getExit ()const**
  Get exit. (see Section 2.151.1)

- **Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.123.1)
- **Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.123.1)
- **double getOrigin ()const**
  Get origin. (see Section 2.151.1)
- **Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.123.1)
- **Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.151.1)
- **Euclid3D getTransform (double)const**
  Get transform. (see Section 2.151.1)
- **const RBendGeometry& operator= (const RBendGeometry&)**
  (see Section 2.123.1)
- **const StraightGeometry& operator= (const StraightGeometry&)**
  (see Section 2.151.1)
- **const Geometry& operator= (const Geometry&)**
  (see Section 2.80.1)
- **void setBendAngle (double)**
  Set angle. (see Section 2.123.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.151.1)
- **virtual ~RBendGeometry ()**
  (see Section 2.123.1)

2.123.1 Detailed descriptions

Public members

- **RBendGeometry (double,double)**
  Constructor.

  Construct an RBendGeometry from **length** and **angle**.
- **RBendGeometry (const RBendGeometry&)**

- **virtual double getArcLength ()const**
  Get arc length.

  Return the length measured along a circular arc tangent to the local s-axis at entrance and exit; an approximation to the actual design orbit.

- **virtual double getBendAngle ()const**
  Get angle.

  Return the total bend angle.

- **virtual double getElementLength ()const**
  Get element length.

  Return the straight length of the geometry.

- **Euclid3D getEntranceFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getEntrance()). Return the transform of the local coordinate system from the origin to the entrance of the element.

- **Euclid3D getEntrancePatch ()const**
  Get patch.

  Returns the entrance patch (transformation) which is used to transform the global geometry to the local geometry at entrance.

- **Euclid3D getExitFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getExit()). Return the transform of the local coordinate system from the origin to the exit of the element.

- **Euclid3D getExitPatch ()const**
  Get patch.

  Returns the entrance patch (transformation) which is used to transform the local geometry to the global geometry at exit.

- **Euclid3D getTotalTransform ()const**
  Get transform.

  Equivalent to getTransform(getEntrance(), getExit()). Return the transform of the local coordinate system from the entrance to the exit of the element.

- **const RBendGeometry& operator= (const RBendGeometry&)**

- **void setBendAngle (double)**
  Set angle.

  Assign the bend angle.

- **virtual ~RBendGeometry ()**

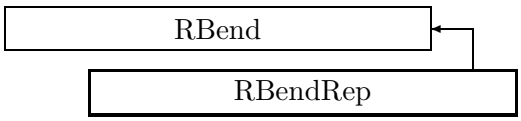

Figure 2.90: Inheritance for class RBendRep

2.124 class RBendRep

Representation for a rectangular bend magnet.
A rectangular bend magnet has a rectilinear geometry about which its multipole components are specified.

Type:	Instantiable
Superclasses:	public RBend
Include file:	./BeamlineCore/RBendRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **RBendRep (const string&)**
  Constructor with given name. (see Section 2.124.1)
- **RBendRep ()**
  (see Section 2.124.1)
- **RBendRep (const RBendRep&)**
  (see Section 2.124.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to RBend. (see Section 2.122.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.124.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getB ()const**
  Get field. (see Section 2.124.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.124.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getEntryFaceCurvature ()const**
  Get entry pole face curvature. (see Section 2.124.1)
- **virtual double getEntryFaceRotation ()const**
  Get pole entry face rotation. (see Section 2.124.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual double getExitFaceCurvature ()const**
  Get exit pole face curvature. (see Section 2.124.1)
- **virtual double getExitFaceRotation ()const**
  Get exit pole face rotation. (see Section 2.124.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)

- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BMultipoleField& getField ()**
  Get field. (see Section 2.124.1)
- **virtual const BMultipoleField& getField ()const**
  Get field. (see Section 2.124.1)
- **virtual RBendGeometry& getGeometry ()**
  Get geometry. (see Section 2.124.1)
- **virtual const RBendGeometry& getGeometry ()const**
  Get geometry. (see Section 2.124.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.124.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **double getNormalComponent (int)const**
  Get normal component. (see Section 2.122.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **double getSkewComponent (int)const**
  Get skew component. (see Section 2.122.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.124.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)

- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.124.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setB (double)**
  Set vertical component. (see Section 2.124.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setEntryFaceCurvature (double)**
  Set entry pole face curvature. (see Section 2.124.1)
- **virtual void setEntryFaceRotation (double)**
  Set pole entry face rotation. (see Section 2.124.1)
- **virtual void setExitFaceCurvature (double)**
  Set exit pole face curvature. (see Section 2.124.1)
- **virtual void setExitFaceRotation (double)**
  Set exit pole face rotation. (see Section 2.124.1)

- **virtual void setField (const BMultipoleField&)**
  Set field. (see Section 2.124.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **void setNormalComponent (int,double)**
  Set normal component. (see Section 2.122.1)
- **void setSkewComponent (int,double)**
  Set skew component. (see Section 2.122.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~RBendRep ()**
  (see Section 2.124.1)

2.124.1 Detailed descriptions

Public members

- **RBendRep (const string&)**
  Constructor with given name.
- **RBendRep ()**
- **RBendRep (const RBendRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual double getB ()const**
  Get field.

  Return the vertical component of the field in Teslas.
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.

- **virtual double getEntryFaceCurvature ()const**
  Get entry pole face curvature.

  Return the curvature of the entry pole face. A positive curvature creates a convex pole face.

- **virtual double getEntryFaceRotation ()const**
  Get pole entry face rotation.

  Return the rotation of the entry pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual double getExitFaceCurvature ()const**
  Get exit pole face curvature.

  Return the curvature of the exit pole face. A positive curvature creates a convex pole face.

- **virtual double getExitFaceRotation ()const**
  Get exit pole face rotation.

  Return the rotation of the exit pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual BMultipoleField& getField ()**
  Get field.

  Version for non-constant object.

- **virtual const BMultipoleField& getField ()const**
  Get field.

  Version for constant object.

- **virtual RBendGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const RBendGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors.

  Build a FieldWrapper pointing to the bend and return a pointer to that wrapper.

- **virtual void setB (double)**
  Set vertical component.

  Assign the vertical component of the field in Teslas.

- **virtual void setEntryFaceCurvature (double)**
  Set entry pole face curvature.

  Return the curvature of the entry pole face. A positive curvature creates a convex pole face.

- **virtual void setEntryFaceRotation (double)**
  Set pole entry face rotation.

  Return the rotation of the entry pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual void setExitFaceCurvature (double)**
  Set exit pole face curvature.

  Return the curvature of the exit pole face. A positive curvature creates a convex pole face.

- **virtual void setExitFaceRotation (double)**
  Set exit pole face rotation.

  Return the rotation of the exit pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual void setField (const BMultipoleField&)**
  Set field.

  Assign the multipole expansion.

- **virtual ~RBendRep ()**

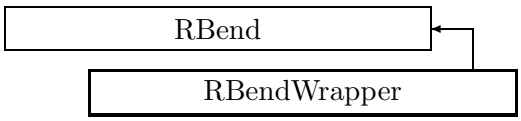

Figure 2.91: Inheritance for class RBendWrapper

2.125 class RBendWrapper

Representation of a perturbed rectangular bend.
A RBendWrapper represents a unique instance of a bend magnet in the accelerator model. It defines imperfections of the field, related to an "ideal" magnet contained in the wrapper.

Type:	Instantiable
Superclasses:	public RBend
Include file:	./ComponentWrappers/RBendWrapper.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **RBendWrapper (RBend*)**
  Constructor. (see Section 2.125.1)
- **RBendWrapper (const RBendWrapper&)**
  (see Section 2.125.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to modified bend. (see Section 2.125.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Make clone. (see Section 2.125.1)

- **virtual ElementBase* copyStructure ()**
  Make structural copy. (see Section 2.125.1)
- **virtual BMultipoleField& errorField ()const**
  Get multipole field error. (see Section 2.125.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getB ()const**
  Get dipole component. (see Section 2.125.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const RBend& getDesign ()const**
  Get design RBend. (see Section 2.125.1)
- **virtual RBend& getDesign ()**
  Get design RBend. (see Section 2.125.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getEntryFaceCurvature ()const**
  Get pole face curvature. (see Section 2.125.1)
- **virtual double getEntryFaceRotation ()const**
  Get pole face rotation. (see Section 2.125.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual double getExitFaceCurvature ()const**
  Get pole face curvature. (see Section 2.125.1)
- **virtual double getExitFaceRotation ()const**
  Get pole face rotation. (see Section 2.125.1)

- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BMultipoleField& getField ()**
  Get field. (see Section 2.125.1)
- **virtual const BMultipoleField& getField ()const**
  Get field. (see Section 2.125.1)
- **virtual RBendGeometry& getGeometry ()**
  Get geometry. (see Section 2.125.1)
- **virtual const RBendGeometry& getGeometry ()const**
  Get geometry. (see Section 2.125.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **double getNormalComponent (int)const**
  Get normal component. (see Section 2.122.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **double getSkewComponent (int)const**
  Get skew component. (see Section 2.122.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.125.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)

- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Make wrapper for this bend. (see Section 2.125.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.125.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.125.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.125.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Remove all wrappers. (see Section 2.125.1)
- **virtual const ElementBase* removeWrappers ()const**
  Remove all wrappers. (see Section 2.125.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **void setNormalComponent (int,double)**
  Set normal component. (see Section 2.122.1)
- **void setSkewComponent (int,double)**
  Set skew component. (see Section 2.122.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)

- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~RBendWrapper ()**
  (see Section 2.125.1)

2.125.1 Detailed descriptions

Public members

- **RBendWrapper (RBend*)**
  Constructor.

  Constructs a wrapper for its argument. The wrapper is not sharable by default.
- **RBendWrapper (const RBendWrapper&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to modified bend.
- **virtual ElementBase* clone ()const**
  Make clone.
- **virtual ElementBase* copyStructure ()**
  Make structural copy.
- **virtual BMultipoleField& errorField ()const**
  Get multipole field error.

  This method can be used to get or set the error field. The error field offset is mutable, so as to allow changing it in a constant structure.
- **virtual double getB ()const**
  Get dipole component.
- **virtual const RBend& getDesign ()const**
  Get design RBend.

  Version for constant object.
- **virtual RBend& getDesign ()**
  Get design RBend.

  Version for non-constant object.
- **virtual double getEntryFaceCurvature ()const**
  Get pole face curvature.

  Return the curvature of the entry pole face. A positive curvature creates a convex pole face.

- **virtual double getEntryFaceRotation ()const**
  Get pole face rotation.

  Return the rotation of the entry pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual double getExitFaceCurvature ()const**
  Get pole face curvature.

  Return the curvature of the exit pole face. A positive curvature creates a convex pole face.

- **virtual double getExitFaceRotation ()const**
  Get pole face rotation.

  Return the rotation of the exit pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual BMultipoleField& getField ()**
  Get field.

  Return the perturbed field. Version for non-constant object.

- **virtual const BMultipoleField& getField ()const**
  Get field.

  Return the perturbed field. Version for constant object.

- **virtual RBendGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const RBendGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual ElementBase* makeFieldWrapper ()**
  Make wrapper for this bend.

  Return **this**, since this is already a field wrapper.

- **virtual void makeSharable ()**
  Set sharable flag.

  The whole structure depending on **this** is marked as sharable. After this call a **copyStructure()** call reuses the element.

- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper.

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper.

- **virtual ElementBase* removeWrappers ()**
  Remove all wrappers.

- **virtual const ElementBase* removeWrappers ()const**
  Remove all wrappers.
- **virtual ~RBendWrapper ()**

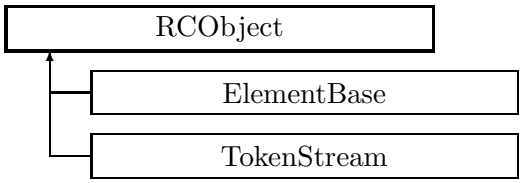

Figure 2.92: Inheritance for class RCObject

2.126 class RCObject

Abstract base class for reference counted objects.
It collaborates with the templace class Pointer<Object>. It is modelled after the class RCObject described in: center Scott Meyers, More Effective C++, Addison Wesley, 1996, pg. 195. /center The idiom `delete this` has been avoided by moving the deletion to class Pointer<Object>. All derived classes must implement a clone() method, to allow the Pointer class to copy the object pointed at. All constructors, the destructor, and the assignment operators are protected, since a stand-alone RCObject makes no sense.

Type:	abstract
Include file:	./MemoryManagement/RCObject.hh

Synopsis (including inherited members)

Public members

- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)

Protected members

- **RCObject ()**
  Default constructor. (see Section 2.126.1)
- **RCObject (const RCObject&)**
  Copy constructor. (see Section 2.126.1)
- **RCObject& operator= (const RCObject&)**
  (see Section 2.126.1)
- **virtual ~RCObject ()**
  (see Section 2.126.1)

2.126.1 Detailed descriptions

Public members

- **int addReference ()const**
  Increment reference count.

  Return the new value of the reference count.

- **bool isShared ()const**
  Test for sharing.

  Return true, if the pointee has more than one reference.

- **int removeReference ()const**
  Decrement the reference count.

  Return the new value of the reference count.

Protected members

- **RCObject ()**
  Default constructor.

  By default a new object is sharable.

- **RCObject (const RCObject&)**
  Copy constructor.

  The copy inherits the sharable state.

- **RCObject& operator= (const RCObject&)**

- **virtual ~RCObject ()**

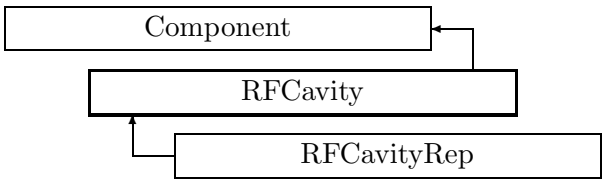

Figure 2.93: Inheritance for class RFCavity

2.127 class RFCavity

Interface for RF cavity.
Class RFCavity defines the abstract interface for RF cavities.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/RFCavity.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **RFCavity (const string&)**
  Constructor with given name. (see Section 2.127.1)
- **RFCavity ()**
  (see Section 2.127.1)
- **RFCavity (const RFCavity&)**
  (see Section 2.127.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to RFCavity. (see Section 2.127.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getAmplitude ()const**
  Get RF amplitude. (see Section 2.127.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual double getFrequency ()const**
  Get RF frequencey. (see Section 2.127.1)

- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual double getPhase ()const**
  Get RF phase. (see Section 2.127.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~RFCavity ()**
  (see Section 2.127.1)

2.127.1 Detailed descriptions

Public members

- **RFCavity (const string&)**
  Constructor with given name.
- **RFCavity ()**
- **RFCavity (const RFCavity&)**

- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to RFCavity.
- **virtual double getAmplitude ()const**
  Get RF amplitude.
- **virtual double getFrequency ()const**
  Get RF frequencey.
- **virtual double getPhase ()const**
  Get RF phase.
- **virtual ~RFCavity ()**

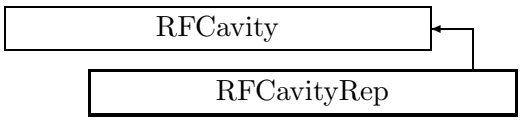

Figure 2.94: Inheritance for class RFCavityRep

2.128 class RFCavityRep

Representation for a RF cavity.

Type:	Instantiable
Superclasses:	public RFCavity
Include file:	./BeamlineCore/RFCavityRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **RFCavityRep (const string&)**
  Constructor with given name. (see Section 2.128.1)
- **RFCavityRep ()**
  (see Section 2.128.1)
- **RFCavityRep (const RFCavityRep&)**
  (see Section 2.128.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to RFCavity. (see Section 2.127.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.128.1)

- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getAmplitude ()const**
  Get amplitude. (see Section 2.128.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.128.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual AcceleratingField& getField ()**
  Get field. (see Section 2.128.1)
- **virtual const AcceleratingField& getField ()const**
  Get field. (see Section 2.128.1)
- **virtual double getFrequency ()const**
  Get frequency. (see Section 2.128.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.128.1)

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.128.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.128.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual double getPhase ()const**
  Get phase. (see Section 2.128.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.128.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)

- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAmplitude (double)**
  Set amplitude. (see Section 2.128.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setFrequency (double)**
  Set frequency. (see Section 2.128.1)
- **static void setIgnore (bool)**
  Set ignore switch. (see Section 2.128.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void setPhase (double)**
  Set phase. (see Section 2.128.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~RFCavityRep ()**
  (see Section 2.128.1)

2.128.1 Detailed descriptions

Public members

- **RFCavityRep (const string&)**
  Constructor with given name.
- **RFCavityRep ()**
- **RFCavityRep (const RFCavityRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual double getAmplitude ()const**
  Get amplitude.

  Return the RF amplitude in Volts.
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.
- **virtual AcceleratingField& getField ()**
  Get field.

  Version for non-constant object.
- **virtual const AcceleratingField& getField ()const**
  Get field.

  Version for constant object.
- **virtual double getFrequency ()const**
  Get frequency.

  Return the RF frequency in Hertz.
- **virtual StraightGeometry& getGeometry ()**
  Get geometry.

  Return the element geometry. Version for non-constant object.
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.

  Return the element geometry Version for constant object.
- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual double getPhase ()const**
  Get phase.

  Return the RF phase in radians.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual void setAmplitude (double)**
  Set amplitude.

  Assign the RF amplitude in Volts.

- **virtual void setFrequency (double)**
  Set frequency.

  Assign the RF frequency in Hertz.

- **static void setIgnore (bool)**
  Set ignore switch.

  If **ignore** is true, cavities are ignored for static calculations.

- **virtual void setPhase (double)**
  Set phase.

  Assing the RF phase in radians.

- **virtual ~RFCavityRep ()**

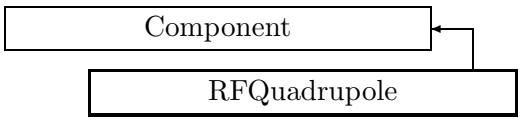

Figure 2.95: Inheritance for class RFQuadrupole

2.129 class RFQuadrupole

Interface for RF Quadrupole.
Class RFQuadrupole defines the abstract interface for a RF Quadrupole.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/RFQuadrupole.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **RFQuadrupole (const string&)**
  Constructor with given name. (see Section 2.129.1)
- **RFQuadrupole ()**
  (see Section 2.129.1)
- **RFQuadrupole (const RFQuadrupole&)**
  (see Section 2.129.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to RFQuadrupole. (see Section 2.129.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)

- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~RFQuadrupole ()**
  (see Section 2.129.1)

2.129.1 Detailed descriptions

Public members

- **RFQuadrupole (const string&)**
  Constructor with given name.
- **RFQuadrupole ()**
- **RFQuadrupole (const RFQuadrupole&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to RFQuadrupole.
- **virtual ~RFQuadrupole ()**

2.130 class Random

The CLASSIC random generator.
This generator is based on:

D. Knuth: The Art of Computer Programming, Vol. 2, 2nd edition.

Type:	Instantiable
Include file:	./Utilities/Random.hh

Synopsis (including inherited members)

Public members

- **Random ()**
  Constructor with default seed. (see Section 2.130.1)
- **Random (int)**
  Constructor with given seed. (see Section 2.130.1)
- **void gauss (double&,double&)**
  Gaussian distribution. (see Section 2.130.1)
- **void init55 (int)**
  Initialise random number generator. (see Section 2.130.1)
- **int integer ()**
  Uniform distribution. (see Section 2.130.1)
- **void reseed (int)**
  Set a new seed. (see Section 2.130.1)
- **double uniform ()**
  Uniform distribution. (see Section 2.130.1)
- **~Random ()**
  (see Section 2.130.1)

2.130.1 Detailed descriptions

Public members

- **Random ()**
  Constructor with default seed.
- **Random (int)**
  Constructor with given seed.

  The argument is the new seed, an integer in the range [0, 1000000000].
- **void gauss (double&,double&)**
  Gaussian distribution.

  Return two real Gaussian pseudo-random numbers with unit standard deviation. The values are placed in the two arguments.

- **void init55 (int)**
  Initialise random number generator.

- **int integer ()**
  Uniform distribution.

  Return an integer pseudo-random number in the range [0,1000000000).

- **void reseed (int)**
  Set a new seed.

  The argument is the new seed, an integer in the range [0, 1000000000].

- **double uniform ()**
  Uniform distribution.

  Return a real pseudo-random number in the range [0,1).

- **~Random ()**

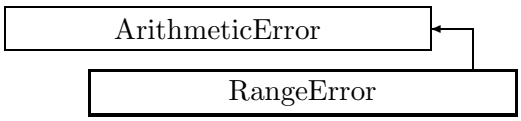

Figure 2.96: Inheritance for class RangeError

2.131 class RangeError

Range error.
This exception is thrown, when a CLASSIC routine or method detects an index out of range.

Type:	Instantiable
Superclasses:	public ArithmeticError
Include file:	./Utilities/RangeError.hh

Synopsis (including inherited members)

Public members

- **RangeError (const string&,const string&)**
  The usual constructor. (see Section 2.131.1)
- **RangeError (const RangeError&)**
  (see Section 2.131.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~RangeError ()**
  (see Section 2.131.1)

2.131.1 Detailed descriptions

Public members

- **RangeError (const string&,const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  msg the message string identifying the exception

  Construction/destruction.
- **RangeError (const RangeError&)**
- **virtual ~RangeError ()**

2.132 class Rotation3D

Rotation in 3-dimensional space.
There are two possible external representations:

Matrix3D R The matrix R is an element of SO(3). Its column vectors define the unit vectors of the rotated coordinate system, expressed in the original system. This representation is used internally and can only be read from a Rotation3D object. To build such an object, use the other representation.

Vector3D V The direction of V defines the axis of rotation, and its length the sine of half the rotation angle. A zero vector represents the identity.

The copy constructor, destructor, and assignment operator generated by the compiler perform the correct operation. For speed reasons they are not implemented.

Type:	Instantiable
Include file:	./BeamlineGeometry/Rotation3D.hh

Synopsis (including inherited members)

Public members

- **static Rotation3D Identity ()**
  Make identity. (see Section 2.132.1)
- **Rotation3D ()**
  Default constructor. (see Section 2.132.1)
- **Rotation3D (double,double,double)**
  Constructor. (see Section 2.132.1)
- **static Rotation3D XRotation (double)**
  Make rotation. (see Section 2.132.1)
- **static Rotation3D YRotation (double)**
  Make rotation. (see Section 2.132.1)
- **static Rotation3D ZRotation (double)**
  Make rotation. (see Section 2.132.1)
- **Vector3D getAxis ()const**
  Get axis vector. (see Section 2.132.1)
- **void getAxis (double&,double&,double&)const**
  Get axis vector. (see Section 2.132.1)
- **Rotation3D inverse ()const**
  Inversion. (see Section 2.132.1)
- **bool isIdentity ()const**
  Test for identity. (see Section 2.132.1)
- **bool isPureXRotation ()const**
  Test for rotation. (see Section 2.132.1)

- **bool isPureYRotation ()const**
  Test for rotation. (see Section 2.132.1)
- **bool isPureZRotation ()const**
  Test for rotation. (see Section 2.132.1)
- **bool operator!= (const Rotation3D&)const**
  (see Section 2.132.1)
- **double operator() (int,int)const**
  Get component. (see Section 2.132.1)
- **Rotation3D operator* (const Rotation3D&)const**
  Multiply. (see Section 2.132.1)
- **Vector3D operator* (const Vector3D&)const**
  Rotate. (see Section 2.132.1)
- **Rotation3D& operator*= (const Rotation3D&)**
  Multiply and assign. (see Section 2.132.1)
- **bool operator== (const Rotation3D&)const**
  (see Section 2.132.1)

2.132.1 Detailed descriptions

Public members

- **static Rotation3D Identity ()**
  Make identity.
- **Rotation3D ()**
  Default constructor.

  Constructs identity.
- **Rotation3D (double,double,double)**
  Constructor.

  Use axis vector (vx,vy,vz).
- **static Rotation3D XRotation (double)**
  Make rotation.

  Construct pure rotation around x-axis.
- **static Rotation3D YRotation (double)**
  Make rotation.

  Construct pure rotation around y-axis.
- **static Rotation3D ZRotation (double)**
  Make rotation.

  Construct pure rotation around z-axis.

- **Vector3D getAxis ()const**
  Get axis vector.

  Return the axis vector.

- **void getAxis (double&,double&,double&)const**
  Get axis vector.

  Return the axis vector in (vx,vy,vz).

- **Rotation3D inverse ()const**
  Inversion.

- **bool isIdentity ()const**
  Test for identity.

- **bool isPureXRotation ()const**
  Test for rotation.

  Return true, if **this** is a pure x-rotation.

- **bool isPureYRotation ()const**
  Test for rotation.

  Return true, if **this** is a pure y-rotation.

- **bool isPureZRotation ()const**
  Test for rotation.

  Return true, if **this** is a pure z-rotation.

- **bool operator!= (const Rotation3D&)const**

- **double operator() (int,int)const**
  Get component.

  Return element (i,k) of the orthogonal matrix.

- **Rotation3D operator* (const Rotation3D&)const**
  Multiply.

- **Vector3D operator* (const Vector3D&)const**
  Rotate.

  Return vector **rhs** rotated by the rotation **this**.

- **Rotation3D& operator*= (const Rotation3D&)**
  Multiply and assign.

- **bool operator== (const Rotation3D&)const**

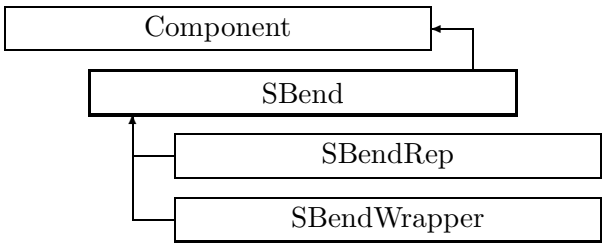

Figure 2.97: Inheritance for class SBend

2.133 class SBend

Interface for sector bend.
This class defines the abstract interface for sector bend magnets. A sector bend magnet has a curved geometry about which its multipole components are specified.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/SBend.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **SBend (const string&)**
  Constructor with given name. (see Section 2.133.1)
- **SBend ()**
  (see Section 2.133.1)
- **SBend (const SBend&)**
  (see Section 2.133.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to SBend. (see Section 2.133.1)

- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getB ()const**
  Get dipole field of SBend. (see Section 2.133.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getEntryFaceCurvature ()const**
  Get entry pole face curvature. (see Section 2.133.1)
- **virtual double getEntryFaceRotation ()const**
  Get pole entry face rotation. (see Section 2.133.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual double getExitFaceCurvature ()const**
  Get exit pole face curvature. (see Section 2.133.1)
- **virtual double getExitFaceRotation ()const**
  Get exit pole face rotation. (see Section 2.133.1)

- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BMultipoleField& getField ()**
  Get multipole expansion of field. (see Section 2.133.1)
- **virtual const BMultipoleField& getField ()const**
  Get multipole expansion of field. (see Section 2.133.1)
- **virtual PlanarArcGeometry& getGeometry ()**
  Get SBend geometry. (see Section 2.133.1)
- **virtual const PlanarArcGeometry& getGeometry ()const**
  Get SBend geometry (see Section 2.133.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **double getNormalComponent (int)const**
  Get normal component. (see Section 2.133.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **double getSkewComponent (int)const**
  Get skew component. (see Section 2.133.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)

- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **void setNormalComponent (int,double)**
  Set normal component. (see Section 2.133.1)
- **void setSkewComponent (int,double)**
  Set skew component. (see Section 2.133.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)

- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~SBend ()**
  (see Section 2.133.1)

2.133.1 Detailed descriptions

Public members

- **SBend (const string&)**
  Constructor with given name.
- **SBend ()**
- **SBend (const SBend&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to SBend.
- **virtual double getB ()const**
  Get dipole field of SBend.
- **virtual double getEntryFaceCurvature ()const**
  Get entry pole face curvature.

  Return the curvature of the entry pole face. A positive curvature creates a convex pole face.
- **virtual double getEntryFaceRotation ()const**
  Get pole entry face rotation.

  Return the rotation of the entry pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.
- **virtual double getExitFaceCurvature ()const**
  Get exit pole face curvature.

  Return the curvature of the exit pole face. A positive curvature creates a convex pole face.
- **virtual double getExitFaceRotation ()const**
  Get exit pole face rotation.

  Return the rotation of the exit pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.
- **virtual BMultipoleField& getField ()**
  Get multipole expansion of field.

  Version for non-constant object.

- **virtual const BMultipoleField& getField ()const**
  Get multipole expansion of field.

  Version for constant object.

- **virtual PlanarArcGeometry& getGeometry ()**
  Get SBend geometry.

  Version for non-constant object.

- **virtual const PlanarArcGeometry& getGeometry ()const**
  Get SBend geometry

  Version for constant object.

- **double getNormalComponent (int)const**
  Get normal component.

  Return the normal component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the return value is zero.

- **double getSkewComponent (int)const**
  Get skew component.

  Return the skew component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the return value is zero.

- **void setNormalComponent (int,double)**
  Set normal component.

  Set the normal component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the component is created.

- **void setSkewComponent (int,double)**
  Set skew component.

  Set the skew component of order **n** in T/m**(n-1). If **n** is larger than the maximum order, the component is created.

- **virtual ~SBend ()**

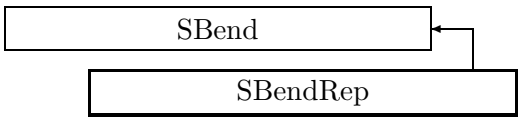

Figure 2.98: Inheritance for class SBendRep

2.134 class SBendRep

Representation for a sector bend magnet.
A sector bend magnet has a planar arc geometry about which its multipole components are specified.

Type:	Instantiable
Superclasses:	public SBend
Include file:	./BeamlineCore/SBendRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **SBendRep (const string&)**
  Constructor with given name. (see Section 2.134.1)
- **SBendRep ()**
  (see Section 2.134.1)
- **SBendRep (const SBendRep&)**
  (see Section 2.134.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to SBend. (see Section 2.133.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.134.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getB ()const**
  Get field. (see Section 2.134.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.134.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getEntryFaceCurvature ()const**
  Get entry pole face curvature. (see Section 2.134.1)
- **virtual double getEntryFaceRotation ()const**
  Get pole entry face rotation. (see Section 2.134.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual double getExitFaceCurvature ()const**
  Get exit pole face curvature. (see Section 2.134.1)
- **virtual double getExitFaceRotation ()const**
  Get exit pole face rotation. (see Section 2.134.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)

- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BMultipoleField& getField ()**
  Get field. (see Section 2.134.1)
- **virtual const BMultipoleField& getField ()const**
  Get field. (see Section 2.134.1)
- **virtual PlanarArcGeometry& getGeometry ()**
  Get geometry. (see Section 2.134.1)
- **virtual const PlanarArcGeometry& getGeometry ()const**
  Get geometry. (see Section 2.134.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.134.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **double getNormalComponent (int)const**
  Get normal component. (see Section 2.133.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **double getSkewComponent (int)const**
  Get skew component. (see Section 2.133.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.134.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)

- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.134.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setB (double)**
  Set vertical component. (see Section 2.134.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setEntryFaceCurvature (double)**
  Set entry pole face curvature. (see Section 2.134.1)
- **virtual void setEntryFaceRotation (double)**
  Set pole entry face rotation. (see Section 2.134.1)
- **virtual void setExitFaceCurvature (double)**
  Set exit pole face curvature. (see Section 2.134.1)
- **virtual void setExitFaceRotation (double)**
  Set exit pole face rotation. (see Section 2.134.1)

- **virtual void setField (const BMultipoleField&)**
  Set field. (see Section 2.134.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **void setNormalComponent (int,double)**
  Set normal component. (see Section 2.133.1)
- **void setSkewComponent (int,double)**
  Set skew component. (see Section 2.133.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~SBendRep ()**
  (see Section 2.134.1)

2.134.1 Detailed descriptions

Public members

- **SBendRep (const string&)**
  Constructor with given name.
- **SBendRep ()**
- **SBendRep (const SBendRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual double getB ()const**
  Get field.

  Return the vertical component of the field in Teslas.
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.

- **virtual double getEntryFaceCurvature ()const**
  Get entry pole face curvature.

  Return the curvature of the entry pole face. A positive curvature creates a convex pole face.

- **virtual double getEntryFaceRotation ()const**
  Get pole entry face rotation.

  Return the rotation of the entry pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual double getExitFaceCurvature ()const**
  Get exit pole face curvature.

  Return the curvature of the exit pole face. A positive curvature creates a convex pole face.

- **virtual double getExitFaceRotation ()const**
  Get exit pole face rotation.

  Return the rotation of the exit pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual BMultipoleField& getField ()**
  Get field.

  Version for non-constant object.

- **virtual const BMultipoleField& getField ()const**
  Get field.

  Version for constant object.

- **virtual PlanarArcGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const PlanarArcGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors.

  Build a FieldWrapper pointing to the bend and return a pointer to that wrapper.

- **virtual void setB (double)**
  Set vertical component.

  Assign the vertical component of the field in Teslas.

- **virtual void setEntryFaceCurvature (double)**
  Set entry pole face curvature.

  Return the curvature of the entry pole face. A positive curvature creates a convex pole face.

- **virtual void setEntryFaceRotation (double)**
  Set pole entry face rotation.

  Return the rotation of the entry pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual void setExitFaceCurvature (double)**
  Set exit pole face curvature.

  Return the curvature of the exit pole face. A positive curvature creates a convex pole face.

- **virtual void setExitFaceRotation (double)**
  Set exit pole face rotation.

  Return the rotation of the exit pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual void setField (const BMultipoleField&)**
  Set field.

  Assign the multipole expansion.

- **virtual ~SBendRep ()**

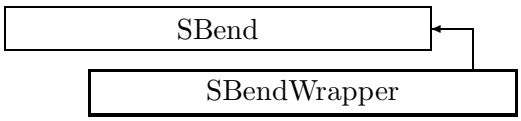

Figure 2.99: Inheritance for class SBendWrapper

2.135 class SBendWrapper

Representation of a perturbed sectorr bend.
A SBendWrapper represents a unique instance of a bend magnet in the accelerator model. It defines imperfections of the field, related to an "ideal" magnet contained in the wrapper.

Type:	Instantiable
Superclasses:	public SBend
Include file:	./ComponentWrappers/SBendWrapper.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **SBendWrapper (SBend*)**
  Constructor. (see Section 2.135.1)
- **SBendWrapper (const SBendWrapper&)**
  (see Section 2.135.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to modified bend. (see Section 2.135.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Make clone. (see Section 2.135.1)

- **virtual ElementBase* copyStructure ()**
  Make structural copy. (see Section 2.135.1)
- **virtual BMultipoleField& errorField ()const**
  Get multipole field error. (see Section 2.135.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getB ()const**
  Get dipole component. (see Section 2.135.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const SBend& getDesign ()const**
  Get design SBend. (see Section 2.135.1)
- **virtual SBend& getDesign ()**
  Get design SBend. (see Section 2.135.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getEntryFaceCurvature ()const**
  Get pole face curvature. (see Section 2.135.1)
- **virtual double getEntryFaceRotation ()const**
  Get pole face rotation. (see Section 2.135.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual double getExitFaceCurvature ()const**
  Get pole face curvature. (see Section 2.135.1)
- **virtual double getExitFaceRotation ()const**
  Get pole face rotation. (see Section 2.135.1)

- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BMultipoleField& getField ()**
  Get field. (see Section 2.135.1)
- **virtual const BMultipoleField& getField ()const**
  Get field. (see Section 2.135.1)
- **virtual PlanarArcGeometry& getGeometry ()**
  Get geometry. (see Section 2.135.1)
- **virtual const PlanarArcGeometry& getGeometry ()const**
  Get geometry. (see Section 2.135.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **double getNormalComponent (int)const**
  Get normal component. (see Section 2.133.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **double getSkewComponent (int)const**
  Get skew component. (see Section 2.133.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.135.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)

- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Make wrapper for this bend. (see Section 2.135.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.135.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.135.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.135.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Remove all wrappers. (see Section 2.135.1)
- **virtual const ElementBase* removeWrappers ()const**
  Remove all wrappers. (see Section 2.135.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **void setNormalComponent (int,double)**
  Set normal component. (see Section 2.133.1)
- **void setSkewComponent (int,double)**
  Set skew component. (see Section 2.133.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)

- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~SBendWrapper ()**
  (see Section 2.135.1)

2.135.1 Detailed descriptions

Public members

- **SBendWrapper (SBend*)**
  Constructor.

  Constructs a wrapper for its argument. The wrapper is not sharable by default.
- **SBendWrapper (const SBendWrapper&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to modified bend.
- **virtual ElementBase* clone ()const**
  Make clone.
- **virtual ElementBase* copyStructure ()**
  Make structural copy.
- **virtual BMultipoleField& errorField ()const**
  Get multipole field error.

  This method can be used to get or set the error field. The error field offset is mutable, so as to allow changing it in a constant structure.
- **virtual double getB ()const**
  Get dipole component.
- **virtual const SBend& getDesign ()const**
  Get design SBend.

  Version for constant object.
- **virtual SBend& getDesign ()**
  Get design SBend.

  Version for non-constant object.
- **virtual double getEntryFaceCurvature ()const**
  Get pole face curvature.

  Return the curvature of the entry pole face. A positive curvature creates a convex pole face.

- **virtual double getEntryFaceRotation ()const**
  Get pole face rotation.

  Return the rotation of the entry pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual double getExitFaceCurvature ()const**
  Get pole face curvature.

  Return the curvature of the exit pole face. A positive curvature creates a convex pole face.

- **virtual double getExitFaceRotation ()const**
  Get pole face rotation.

  Return the rotation of the exit pole face with respect to the x-axis. A positive angle rotates the pole face normal away from the centre of the machine.

- **virtual BMultipoleField& getField ()**
  Get field.

  Return the perturbed field. Version for non-constant object.

- **virtual const BMultipoleField& getField ()const**
  Get field.

  Return the perturbed field. Version for constant object.

- **virtual PlanarArcGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const PlanarArcGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual ElementBase* makeFieldWrapper ()**
  Make wrapper for this bend.

  Return **this**, since this is already a field wrapper.

- **virtual void makeSharable ()**
  Set sharable flag.

  The whole structure depending on **this** is marked as sharable. After this call a **copyStructure()** call reuses the element.

- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper.

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper.

- **virtual ElementBase* removeWrappers ()**
  Remove all wrappers.

- **virtual const ElementBase* removeWrappers ()const**
  Remove all wrappers.
- **virtual ~SBendWrapper ()**

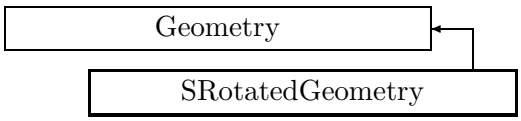

Figure 2.100: Inheritance for class SRotatedGeometry

2.136 class SRotatedGeometry

A Geometry which wraps an arbitrary geometry in two s-rotations.
An SRotatedGeometry object is a Geometry wrapper which adds two arbitrary s-rotations (local z-axis rotations) to the entrance and exit planes of an arbitrary geometry. Any Geometry object can be wrapped, including another SRotatedGeometry. The s-rotations become part of the global geometry definition. Functions for setting the two in- and out-rotations using certain constraints are provided.
NOTE: in general the transformations returned include the effects of the s-rotations when the required distance parameter specified is either the entrance or exit point. Requests for transformations within the geometry (i.e. from s1 to s2, where s1 and/or s2 are not the entrance or exit planes) do not contain the s-rotations.

Type:	Instantiable
Superclasses:	public Geometry
Include file:	./BeamlineGeometry/SRotatedGeometry.hh

Synopsis (including inherited members)

Public members

- **enum BalanceMode**
  Balance mode. (see Section 2.136.1)
- **SRotatedGeometry (const Geometry&,double,double)**
  srotOut is set to -srotIn. srotOut calculated. Constructor. (see Section 2.136.1)
- **SRotatedGeometry (const Geometry&,double,BalanceMode)**
  Constructor. (see Section 2.136.1)
- **SRotatedGeometry (const SRotatedGeometry&)**
  (see Section 2.136.1)
- **void balanceSrots (BalanceMode)**
  Balance rotations. (see Section 2.136.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.136.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.136.1)
- **double getEntrance ()const**
  Get entrance. (see Section 2.136.1)

- **Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.136.1)
- **Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.136.1)
- **double getExit ()const**
  Get exit. (see Section 2.136.1)
- **Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.136.1)
- **Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.136.1)
- **double getOrigin ()const**
  Get origin. (see Section 2.136.1)
- **double getSrotIn ()const**
  Get entrance rotation. (see Section 2.136.1)
- **double getSrotOut ()const**
  Get exit rotation. (see Section 2.136.1)
- **Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.136.1)
- **Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.136.1)
- **Euclid3D getTransform (double)const**
  Get transform. (see Section 2.136.1)
- **const SRotatedGeometry& operator= (const SRotatedGeometry&)**
  (see Section 2.136.1)
- **const Geometry& operator= (const Geometry&)**
  (see Section 2.80.1)
- **virtual void setElementLength (double)**
  Set geometry length. (see Section 2.80.1)
- **void setSrotIn (double)**
  Set entrance rotation. (see Section 2.136.1)
- **void setSrotOut (double)**
  Set exit rotation. (see Section 2.136.1)
- **virtual ~SRotatedGeometry ()**
  (see Section 2.136.1)

2.136.1 Detailed descriptions

Public members

- **enum BalanceMode**
  Balance mode.

- **SRotatedGeometry (const Geometry&,double,double)**
  srotOut is set to -srotIn. srotOut calculated. Constructor.

  srotOut is calculated to be the vlaue which returns the local x-axis after rotation to be in a plane parallel with the local x-axis before application of srotIn. Use the wrapped geometry **geom**, and the two angles **srotIn** and **srotOut**.

- **SRotatedGeometry (const Geometry&,double,BalanceMode)**
  Constructor.

  Use the wrapped geometry **geom**, the entrance rotation **srotIn**, and the balanc mode **mode**.

- **SRotatedGeometry (const SRotatedGeometry&)**

- **void balanceSrots (BalanceMode)**
  Balance rotations.

  Set the exit rotation in one of two modes:

  if `mode==tilt`, then srotOut is set to -srotIn.

  if `mode==balanceX`, then srotOut is calculated to be the value which returns the local x-axis after rotation to be in a parallel plane with the local x-axis before the application of `srotIn`.

- **virtual double getArcLength ()const**
  Get arc length.

  Return or the design length of the embedded geometry. This is the length of the straight line connecting entrance and exit.

- **virtual double getElementLength ()const**
  Get design length.

  Return or the design length of the embedded geometry. Depending on the element this may be the arc length or the straight length.

- **double getEntrance ()const**
  Get entrance.

  Return the arc length from the origin to the entrance of the geometry (non-positive)

- **Euclid3D getEntranceFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getEntrance()). Return the transform of the local coordinate system from the origin to the entrance of the element.

- **Euclid3D getEntrancePatch ()const**
  Get patch.

  Returns the entrance patch (transformation) which is used to transform the global geometry to the local geometry at entrance.

- **double getExit ()const**
  Get exit.

  Return the arc length from the origin to the exit of the geometry (non-negative)

- **Euclid3D getExitFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getExit()). Return the transform of the local coordinate system from the origin to the exit of the element.

- **Euclid3D getExitPatch ()const**
  Get patch.

  Returns the entrance patch (transformation) which is used to transform the local geometry to the global geometry at exit.

- **double getOrigin ()const**
  Get origin.

  Return the arc length from the entrance to the origin of the geometry (non-negative).

- **double getSrotIn ()const**
  Get entrance rotation.

- **double getSrotOut ()const**
  Get exit rotation.

- **Euclid3D getTotalTransform ()const**
  Get transform.

  Equivalent to getTransform(getEntrance(), getExit()). Return the transform of the local coordinate system from the entrance to the exit of the element.

- **Euclid3D getTransform (double,double)const**
  Get transform.

  Return the transform of the local coordinate system from the position **fromS** to the position **toS**.

- **Euclid3D getTransform (double)const**
  Get transform.

  Equivalent to getTransform(0.0, s). Return the transform of the local coordinate system from the origin and **s**.

- **const SRotatedGeometry& operator= (const SRotatedGeometry&)**

- **void setSrotIn (double)**
  Set entrance rotation.

- **void setSrotOut (double)**
  Set exit rotation.
- **virtual ~SRotatedGeometry ()**

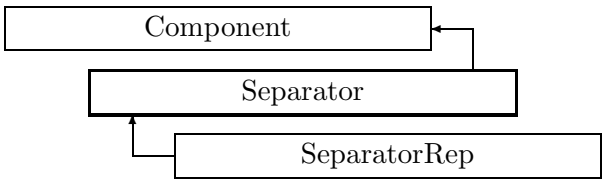

Figure 2.101: Inheritance for class Separator

2.137 class Separator

Interface for electrostatic separator.
Class Separator defines the abstract interface for electrostatic separators.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Separator.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Separator (const string&)**
  Constructor with given name. (see Section 2.137.1)
- **Separator ()**
  (see Section 2.137.1)
- **Separator (const Separator&)**
  (see Section 2.137.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Separator. (see Section 2.137.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getEx ()const**
  Get horizontal component Ex of field in V/m. (see Section 2.137.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getEy ()const**
  Get vertical component Ey of field in V/m. (see Section 2.137.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)

- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)

- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Separator ()**
  (see Section 2.137.1)

2.137.1 Detailed descriptions

Public members

- **Separator (const string&)**
  Constructor with given name.
- **Separator ()**
- **Separator (const Separator&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Separator.

- **virtual double getEx ()const**
  Get horizontal component Ex of field in V/m.
- **virtual double getEy ()const**
  Get vertical component Ey of field in V/m.
- **virtual ~Separator ()**

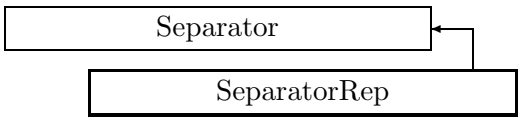

Figure 2.102: Inheritance for class SeparatorRep

2.138 class SeparatorRep

Representation for an electrostatic separator.

Type:	Instantiable
Superclasses:	public Separator
Include file:	./BeamlineCore/SeparatorRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **SeparatorRep (const string&)**
  Constructor with given name. (see Section 2.138.1)
- **SeparatorRep ()**
  (see Section 2.138.1)
- **SeparatorRep (const SeparatorRep&)**
  (see Section 2.138.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Separator. (see Section 2.137.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.138.1)

- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.138.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getEx ()const**
  Get component. (see Section 2.138.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getEy ()const**
  Get component. (see Section 2.138.1)
- **virtual EDipoleField& getField ()**
  Get field. (see Section 2.138.1)
- **virtual const EDipoleField& getField ()const**
  Get field. (see Section 2.138.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.138.1)

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.138.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.138.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.138.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)

- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setEx (double)**
  Set component. (see Section 2.138.1)
- **virtual void setEy (double)**
  Set component. (see Section 2.138.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~SeparatorRep ()**
  (see Section 2.138.1)

2.138.1 Detailed descriptions

Public members

- **SeparatorRep (const string&)**
  Constructor with given name.
- **SeparatorRep ()**
- **SeparatorRep (const SeparatorRep&)**

- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.

- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.

- **virtual double getEx ()const**
  Get component.

  Return the x-component of the electric field in A/m.

- **virtual double getEy ()const**
  Get component.

  Return the y-component of the electric field in A/m.

- **virtual EDipoleField& getField ()**
  Get field.

  Version for non-constant object.

- **virtual const EDipoleField& getField ()const**
  Get field.

  Version for constant object.

- **virtual StraightGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual void setEx (double)**
  Set component.

  Assign the x-component of the electric field in A/m.

- **virtual void setEy (double)**
  Set component.

  Assign the y-component of the electric field in A/m.

- **virtual ~SeparatorRep ()**

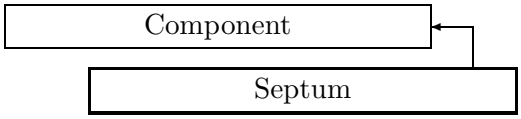

Figure 2.103: Inheritance for class Septum

2.139 class Septum

Interface for septum magnet.
Class Septum defines the abstract interface for a septum magnet.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Septum.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Septum (const string&)**
  Constructor with given name. (see Section 2.139.1)
- **Septum ()**
  (see Section 2.139.1)
- **Septum (const Septum&)**
  (see Section 2.139.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Septum. (see Section 2.139.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)

- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Septum ()**
  (see Section 2.139.1)

2.139.1 Detailed descriptions

Public members

- **Septum (const string&)**
  Constructor with given name.
- **Septum ()**
- **Septum (const Septum&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Septum.
- **virtual ~Septum ()**

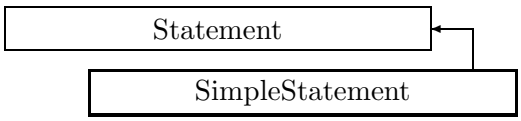

Figure 2.104: Inheritance for class SimpleStatement

2.140 class SimpleStatement

A simple input statement in token form.
The statement is stored as a list of Token's.

Type:	Instantiable
Superclasses:	public Statement
Include file:	./Parser/SimpleStatement.hh

Synopsis (including inherited members)

Public members

- **SimpleStatement (const string&,int)**
  Constructor. (see Section 2.140.1)
- **SimpleStatement (const string&,TokenList&)**
  Constructor. (see Section 2.140.1)
- **typedef std::list<Token> TokenList**
  The type of the enclosed token list. (see Section 2.147.1)
- **void append (const Token&)**
  Append a token. (see Section 2.147.1)
- **bool atEnd ()const**
  Test for end of command. (see Section 2.147.1)
- **bool boolean (bool&)**
  Return boolean value. (see Section 2.147.1)
- **bool delimiter (char)**
  Test for delimiter. (see Section 2.147.1)
- **bool delimiter (const char*)**
  Test for delimiter choice. (see Section 2.147.1)
- **virtual void execute (const Parser&)**
  Execute statement. (see Section 2.140.1)
- **Token& getCurrent ()**
  Return current token and skip it. (see Section 2.147.1)
- **bool integer (int&)**
  Return signed integer. (see Section 2.147.1)

- **bool integer (unsigned&)**
  Return unsigned integer. (see Section 2.147.1)
- **bool keyword (const char*)**
  Test for keyword. (see Section 2.147.1)
- **void mark ()**
  Mark position in command. (see Section 2.147.1)
- **virtual void print (std::ostream&)const**
  Print statement. (see Section 2.147.1)
- **virtual void printWhere (std::ostream&,bool)const**
  Print position. (see Section 2.147.1)
- **bool real (double&)**
  Return real value. (see Section 2.147.1)
- **void restore ()**
  Return to marked position. (see Section 2.147.1)
- **void skip ()**
  Skip. (see Section 2.147.1)
- **void start ()**
  Return to start. (see Section 2.147.1)
- **bool str (string&)**
  Return string value. (see Section 2.147.1)
- **bool word (string&)**
  Return word value. (see Section 2.147.1)
- **virtual ~SimpleStatement ()**
  Destructor. (see Section 2.140.1)

2.140.1 Detailed descriptions

Public members

- **SimpleStatement (const string&,int)**
  Constructor.

  Stores the name of the input stream and the line at which the statement begins.
- **SimpleStatement (const string&,TokenList&)**
  Constructor.

  Stores a name (e.g. for a macro) and the token list.
- **virtual void execute (const Parser&)**
  Execute statement.

  This method provides a hook for structured statement execution. It is called by the parser, and may be overridden to execute e.g. a conditional or a loop statement.

- **virtual ~SimpleStatement ()**
  Destructor.

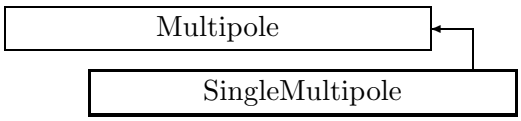

Figure 2.105: Inheritance for class SingleMultipole

2.141 template class SingleMultipole <int>

Representation for single multipoles.
Template for representation of single multipoles. Represents all the basic (design) multipole magnets found in an accelerator. A single multipole has only one multipole component, the pole-number of which cannot be changed (once a quadrupole, always a quadrupole). This differs from a MultipoleRep object which can have an arbitrary number of multipole components.
This template class can be used to instantiate classes like:

Quadrupole (order = 2),

Sextupole (order = 3),

Octupole (order = 4),

SkewQuadrupole (order = -2),

SkewSextupole (order = -3),

SkewOctupole (order = -4).

The order and the skew flag are encoded in the template parameter. A positive **order** implies a normal multipole, A negative **order** implies a skew multipole.

Type:	Instantiable
Superclasses:	public Multipole
Include file:	./BeamlineCore/SingleMultipole.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)

- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **SingleMultipole (const string&)**
  Constructor with given name. (see Section 2.141.1)
- **SingleMultipole ()**
  (see Section 2.141.1)
- **SingleMultipole (const SingleMultipole&)**
  (see Section 2.141.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Multipole. (see Section 2.100.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.141.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.141.1)
- **virtual double getComponent ()const**
  Get component. (see Section 2.141.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)

- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BMultipoleField& getField ()**
  Get field. (see Section 2.141.1)
- **virtual const BMultipoleField& getField ()const**
  Get field. (see Section 2.141.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.141.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.141.1)
- **ElementImage* getImage ()const**
  Construct an image. (see Section 2.141.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **double getNormalComponent (int)const**
  Get normal component. (see Section 2.100.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **double getSkewComponent (int)const**
  Get skew component. (see Section 2.100.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.141.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)

- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setComponent (double)**
  Set component. (see Section 2.141.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **void setNormalComponent (int,double)**
  Set normal component. (see Section 2.100.1)

- **void setSkewComponent (int,double)**
  Set skew component. (see Section 2.100.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~SingleMultipole ()**
  (see Section 2.141.1)

2.141.1 Detailed descriptions

Public members

- **SingleMultipole (const string&)**
  Constructor with given name.
- **SingleMultipole ()**
- **SingleMultipole (const SingleMultipole&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.
- **virtual double getComponent ()const**
  Get component.

  Return the single multipole component in T/m**(n-1).
- **virtual BMultipoleField& getField ()**
  Get field.

  Version for non-constant object.
- **virtual const BMultipoleField& getField ()const**
  Get field.

  Version for constant object.
- **virtual StraightGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual void setComponent (double)**
  Set component.

  Assign the single multipole component in T/m**(n-1).

- **virtual ~SingleMultipole ()**

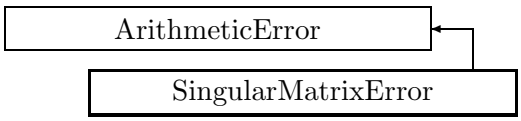

Figure 2.106: Inheritance for class SingularMatrixError

2.142 class SingularMatrixError

Singular matrix exception.
This exception is thrown, when an attempt is made to invert a singular matrix.

Type:	Instantiable
Superclasses:	public ArithmeticError
Include file:	./Utilities/SingularMatrixError.hh

Synopsis (including inherited members)

Public members

- **SingularMatrixError (const string&)**
  The usual constructor. (see Section 2.142.1)
- **SingularMatrixError (const SingularMatrixError&)**
  (see Section 2.142.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~SingularMatrixError ()**
  (see Section 2.142.1)

2.142.1 Detailed descriptions

Public members

- **SingularMatrixError (const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  Construction/destruction.
- **SingularMatrixError (const SingularMatrixError&)**
- **virtual ~SingularMatrixError ()**

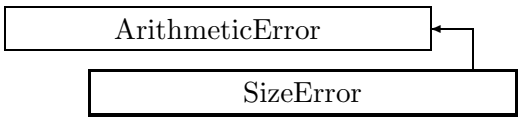

Figure 2.107: Inheritance for class SizeError

2.143 class SizeError

Size error exception.
This exception is thrown, when a CLASSIC routine or method is called with arrays of inconsistent dimensions.

Type:	Instantiable
Superclasses:	public ArithmeticError
Include file:	./Utilities/SizeError.hh

Synopsis (including inherited members)

Public members

- **SizeError (const string&,const string&)**
  The usual constructor. (see Section 2.143.1)
- **SizeError (const SizeError&)**
  (see Section 2.143.1)
- **virtual const string& what ()const**
  Return the message string for the exception. (see Section 2.21.1)
- **virtual const string& where ()const**
  Return the name of the method or function which detected the exception. (see Section 2.21.1)
- **virtual ~SizeError ()**
  (see Section 2.143.1)

2.143.1 Detailed descriptions

Public members

- **SizeError (const string&,const string&)**
  The usual constructor.

  Arguments:

  meth the name of the method or function detecting the exception

  msg the message string identifying the exception

  Construction/destruction.
- **SizeError (const SizeError&)**

- **virtual ~SizeError ()**

2.144 template class SliceIterator <class>

Iterator for array slice.
An iterator permitting to iterate on a non-constant array with a stride different from 1.

Type:	Instantiable
Include file:	./Algebra/SliceIterator.hh

Synopsis (including inherited members)

Public members

- **SliceIterator (T*,ptrdiff_t)**
  Constructor. (see Section 2.144.1)
- **SliceIterator (const SliceIterator<T>&)**
  (see Section 2.144.1)
- **typedef ptrdiff_t difference_type**
  The pointer difference type. (see Section 2.144.1)
- **typedef std::random_access_iterator_tag iterator_category**
  The iterator tag, taken from the standard template library. (see Section 2.144.1)
- **bool operator!= (const SliceIterator<T>&)const**
  (see Section 2.144.1)
- **T& operator* ()const**
  Dereference. (see Section 2.144.1)
- **SliceIterator<T> operator+ (ptrdiff_t)**
  Add multiple stride. (see Section 2.144.1)
- **SliceIterator<T>& operator++ ()**
  Increment (iterate forward). (see Section 2.144.1)
- **SliceIterator<T> operator++ (int)**
  Increment (iterate forward). (see Section 2.144.1)
- **SliceIterator<T>& operator+= (ptrdiff_t)**
  Increment by multiple stride. (see Section 2.144.1)
- **SliceIterator<T> operator- (ptrdiff_t)**
  Subtract multiple stride. (see Section 2.144.1)
- **difference_type operator- (const SliceIterator<T>&)const**
  Difference. (see Section 2.144.1)
- **SliceIterator<T>& operator– ()**
  Decrement (iterate backward). (see Section 2.144.1)
- **SliceIterator<T> operator– (int)**
  Decrement (iterate backward). (see Section 2.144.1)

- **SliceIterator<T>& operator-= (ptrdiff_t)**
  Decrement by multiple stride. (see Section 2.144.1)
- **SliceIterator<T>& operator= (const SliceIterator<T>&)**
  (see Section 2.144.1)
- **bool operator== (const SliceIterator<T>&)const**
  (see Section 2.144.1)
- **T& operator[] (int)const**
  Delegate. (see Section 2.144.1)
- **typedef T* pointer**
  The pointer type. (see Section 2.144.1)
- **typedef T& reference**
  The reference type. (see Section 2.144.1)
- **typedef T value_type**
  The value type. (see Section 2.144.1)

2.144.1 Detailed descriptions

Public members

- **SliceIterator (T*,ptrdiff_t)**
  Constructor.

  Construct iterator for **array**, given a **stride**.
- **SliceIterator (const SliceIterator<T>&)**

- **typedef ptrdiff_t difference_type**
  The pointer difference type.
- **typedef std::random_access_iterator_tag iterator_category**
  The iterator tag, taken from the standard template library.
- **bool operator!= (const SliceIterator<T>&)const**

- **T& operator* ()const**
  Dereference.
- **SliceIterator<T> operator+ (ptrdiff_t)**
  Add multiple stride.
- **SliceIterator<T>& operator++ ()**
  Increment (iterate forward).
- **SliceIterator<T> operator++ (int)**
  Increment (iterate forward).

- **SliceIterator<T>& operator+= (ptrdiff_t)**
  Increment by multiple stride.
- **SliceIterator<T> operator- (ptrdiff_t)**
  Subtract multiple stride.
- **difference_type operator- (const SliceIterator<T>&)const**
  Difference.
- **SliceIterator<T>& operator– ()**
  Decrement (iterate backward).
- **SliceIterator<T> operator– (int)**
  Decrement (iterate backward).
- **SliceIterator<T>& operator-= (ptrdiff_t)**
  Decrement by multiple stride.
- **SliceIterator<T>& operator= (const SliceIterator<T>&)**
- **bool operator== (const SliceIterator<T>&)const**
- **T& operator[] (int)const**
  Delegate.
- **typedef T* pointer**
  The pointer type.
- **typedef T& reference**
  The reference type.
- **typedef T value_type**
  The value type.

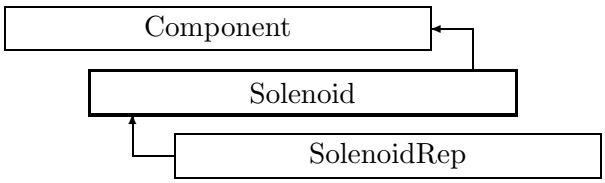

Figure 2.108: Inheritance for class Solenoid

2.145 class Solenoid

Interface for solenoids.
Class Solenoid defines the abstract interface for solenoid magnets.

Type:	abstract
Superclasses:	public Component
Include file:	./AbsBeamline/Solenoid.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **Solenoid (const string&)**
  Constructor with given name. (see Section 2.145.1)
- **Solenoid ()**
  (see Section 2.145.1)
- **Solenoid (const Solenoid&)**
  (see Section 2.145.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Solenoid. (see Section 2.145.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)

- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.52.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getBz ()const**
  Get solenoid field Bz in Teslas. (see Section 2.145.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual EMField& getField ()**
  Return field. (see Section 2.26.1)
- **virtual const EMField& getField ()const**
  Return field. (see Section 2.26.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)

- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)

- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~Solenoid ()**
  (see Section 2.145.1)

2.145.1 Detailed descriptions

Public members

- **Solenoid (const string&)**
  Constructor with given name.
- **Solenoid ()**
- **Solenoid (const Solenoid&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Solenoid.
- **virtual double getBz ()const**
  Get solenoid field Bz in Teslas.

- **virtual ~Solenoid ()**

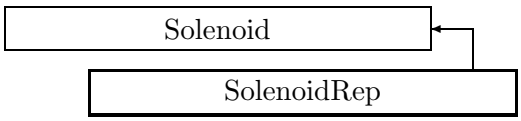

Figure 2.109: Inheritance for class SolenoidRep

2.146 class SolenoidRep

Tepresentation for a solenoid magnet.

Type:	Instantiable
Superclasses:	public Solenoid
Include file:	./BeamlineCore/SolenoidRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **SolenoidRep (const string&)**
  Constructor with given name. (see Section 2.146.1)
- **SolenoidRep ()**
  (see Section 2.146.1)
- **SolenoidRep (const SolenoidRep&)**
  (see Section 2.146.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Solenoid. (see Section 2.145.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.146.1)

- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getBz ()const**
  Get field. (see Section 2.146.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.146.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual ConstBzField& getField ()**
  Get field. (see Section 2.146.1)
- **virtual const ConstBzField& getField ()const**
  Get field. (see Section 2.146.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.146.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.146.1)

- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.146.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.146.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)

- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setBz (double)**
  Set field. (see Section 2.146.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~SolenoidRep ()**
  (see Section 2.146.1)

2.146.1 Detailed descriptions

Public members

- **SolenoidRep (const string&)**
  Constructor with given name.
- **SolenoidRep ()**
- **SolenoidRep (const SolenoidRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.

- **virtual double getBz ()const**
  Get field.

  Return the solenoid field in Teslas.

- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.

- **virtual ConstBzField& getField ()**
  Get field.

  Version for non-constant object.

- **virtual const ConstBzField& getField ()const**
  Get field.

  Version for constant object.

- **virtual StraightGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual void setBz (double)**
  Set field.

  Assign the solenoid field in Teslas.

- **virtual ~SolenoidRep ()**

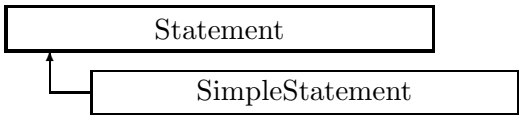

Figure 2.110: Inheritance for class Statement

2.147 class Statement

Interface for statements.
The statement is stored as a list of Token's.

Type:	abstract
Include file:	./Parser/Statement.hh

Synopsis (including inherited members)

Public members

- **Statement (const string&,int)**
  Constructor. (see Section 2.147.1)
- **Statement (const string&,TokenList&)**
  Constructor. (see Section 2.147.1)
- **typedef std::list<Token> TokenList**
  The type of the enclosed token list. (see Section 2.147.1)
- **void append (const Token&)**
  Append a token. (see Section 2.147.1)
- **bool atEnd ()const**
  Test for end of command. (see Section 2.147.1)
- **bool boolean (bool&)**
  Return boolean value. (see Section 2.147.1)
- **bool delimiter (char)**
  Test for delimiter. (see Section 2.147.1)
- **bool delimiter (const char*)**
  Test for delimiter choice. (see Section 2.147.1)
- **virtual void execute (const Parser&)**
  Execute. (see Section 2.147.1)
- **Token& getCurrent ()**
  Return current token and skip it. (see Section 2.147.1)
- **bool integer (int&)**
  Return signed integer. (see Section 2.147.1)
- **bool integer (unsigned&)**
  Return unsigned integer. (see Section 2.147.1)

- **bool keyword (const char*)**
  Test for keyword. (see Section 2.147.1)
- **void mark ()**
  Mark position in command. (see Section 2.147.1)
- **virtual void print (std::ostream&)const**
  Print statement. (see Section 2.147.1)
- **virtual void printWhere (std::ostream&,bool)const**
  Print position. (see Section 2.147.1)
- **bool real (double&)**
  Return real value. (see Section 2.147.1)
- **void restore ()**
  Return to marked position. (see Section 2.147.1)
- **void skip ()**
  Skip. (see Section 2.147.1)
- **void start ()**
  Return to start. (see Section 2.147.1)
- **bool str (string&)**
  Return string value. (see Section 2.147.1)
- **bool word (string&)**
  Return word value. (see Section 2.147.1)
- **virtual ~Statement ()**
  (see Section 2.147.1)

Protected members

- **string buffer_name**
  (see Section 2.147.1)
- **TokenList::iterator curr**
  (see Section 2.147.1)
- **TokenList::iterator keep**
  (see Section 2.147.1)
- **int stat_line**
  (see Section 2.147.1)
- **TokenList tokens**
  (see Section 2.147.1)

2.147.1 Detailed descriptions

Public members

- **Statement (const string&,int)**
  Constructor.

  Store the stream name and the line where the statement begins.

- **Statement (const string&,TokenList&)**
  Constructor.

  Stores a name (e.g. for a macro) and the token list.

- **typedef std::list<Token> TokenList**
  The type of the enclosed token list.

- **void append (const Token&)**
  Append a token.

- **bool atEnd ()const**
  Test for end of command.

  This method is called by the parser in order to find out wether the end of the statement has been reached.

- **bool boolean (bool&)**
  Return boolean value.

  If the next item is a boolean literal:

  > Set **valu**" to its value.
  >
  > Skip the value in the token list.
  >
  > Return true.

  Otherwise return false.

- **bool delimiter (char)**
  Test for delimiter.

  If the next item is the given character, skip it and return true, otherwise return false.

- **bool delimiter (const char*)**
  Test for delimiter choice.

  If the next item is one of the characters in the given string, skip it and return true, otherwise return false.

- **virtual void execute (const Parser&)**
  Execute.

  This method must be specially defined in conditional or loop statements. Normally it just calls the parser to execute the statement.

- **Token& getCurrent ()**
  Return current token and skip it.

- **bool integer (int&)**
  Return signed integer.

  If the next item is an integer:

  > Set **value** to its value.
  >
  > Skip the value in the token list.
  >
  > Return true.

  Otherwise return false.

- **bool integer (unsigned&)**
  Return unsigned integer.

  If the next item is an integer:

  > Set **value** to its value.
  >
  > Skip the value in the token list.
  >
  > Return true.

  Otherwise return false.

- **bool keyword (const char*)**
  Test for keyword.

  If the next item is the keyword **s**, skip it and return true, otherwise return false.

- **void mark ()**
  Mark position in command.

  Parsing can later be resumed by calling restore().

- **virtual void print (std::ostream&)const**
  Print statement.

  Print the statement on **os**.

- **virtual void printWhere (std::ostream&,bool)const**
  Print position.

  Print a message, containing the stream name and its line in the input stream. If **withToken** is true, print also the last token parsed.

- **bool real (double&)**
  Return real value.

  If the next item is a real number:

  > Set **value** to its value.
  >
  > Skip the value in the token list.
  >
  > Return true.

  Otherwise return false.

- **void restore ()**
  Return to marked position.

  Resume parsing at the position wher mark() was last called.

- **void skip ()**
  Skip.

  Skip tokens up to next comma or end of statement, whichever comes first.

- **void start ()**
  Return to start.

  Resume parsing at the beginning of the statement.

- **bool str (string&)**
  Return string value.

  If the next item is a string literal:

  > Set **value** to its value.
  > Skip the value in the token list.
  > Return true.

  Otherwise return false.

- **bool word (string&)**
  Return word value.

  If the next item is a word:

  > Set **value** to its value.
  > Skip the value in the token list.
  > Return true.

  Otherwise return false.

- **virtual ~Statement ()**

Protected members

- **string buffer_name**

- **TokenList::iterator curr**

- **TokenList::iterator keep**

- **int stat_line**

- **TokenList tokens**

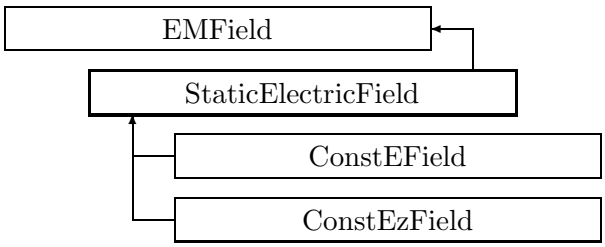

Figure 2.111: Inheritance for class StaticElectricField

2.148 class StaticElectricField

Abstract base class for static electric fields.

Type:	abstract
Superclasses:	public EMField
Include file:	./Fields/StaticElectricField.hh

Synopsis (including inherited members)

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **StaticElectricField ()**
  (see Section 2.148.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)
- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)

- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)
- **virtual void scale (double)**
  Scale the field. (see Section 2.49.1)
- **virtual ~StaticElectricField ()**
  (see Section 2.148.1)

2.148.1 Detailed descriptions

Public members

- **StaticElectricField ()**

- **virtual ~StaticElectricField ()**

2.149 class StaticFixedPoint

Static fixed point of a map.
Class StaticFixedPoint represents the static fixed point of a truncated power series map.

Type:	Instantiable
Include file:	./Algebra/StaticFixedPoint.hh

Synopsis (including inherited members)

Public members

- **StaticFixedPoint (const VpsInvMap<double>&)**
  Constructor. (see Section 2.149.1)
- **StaticFixedPoint ()**
  (see Section 2.149.1)
- **StaticFixedPoint (const StaticFixedPoint&)**
  (see Section 2.149.1)
- **const VpsInvMap<double>& getFixedPoint ()const**
  Get the transformation to the fixed point. (see Section 2.149.1)
- **const VpsInvMap<double>& getFixedPointMap ()const**
  Get map around fixed point. (see Section 2.149.1)
- **˜StaticFixedPoint ()**
  (see Section 2.149.1)

2.149.1 Detailed descriptions

Public members

- **StaticFixedPoint (const VpsInvMap<double>&)**
  Constructor.

  Finds the fixed point for **map**.
- **StaticFixedPoint ()**
- **StaticFixedPoint (const StaticFixedPoint&)**
- **const VpsInvMap<double>& getFixedPoint ()const**
  Get the transformation to the fixed point.

  This map contains the dispersive terms required to eliminate momentum dependence of the fixed point.
- **const VpsInvMap<double>& getFixedPointMap ()const**
  Get map around fixed point.

  Return the map around the fixed point. The dispersive terms are eliminated from this map.

- **~StaticFixedPoint ()**

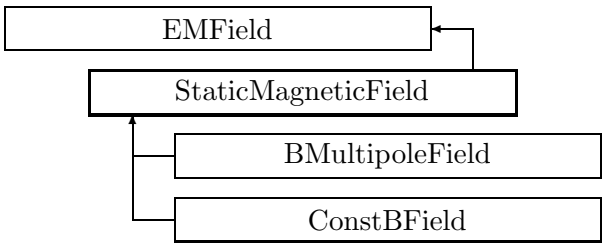

Figure 2.112: Inheritance for class StaticMagneticField

2.150 class StaticMagneticField

Abstract base class for static magnetic fields.

Type:	abstract
Superclasses:	public EMField
Include file:	./Fields/StaticMagneticField.hh

Synopsis (including inherited members)

Public members

- **virtual BVector Bfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual BVector Bfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EBVectors EBfield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&)const**
  Get field. (see Section 2.49.1)
- **virtual EVector Efield (const Point3D&,double)const**
  Get field. (see Section 2.49.1)
- **StaticMagneticField ()**
  (see Section 2.150.1)
- **static const BVector ZeroBfield**
  The constant representing a zero magnetic field. (see Section 2.49.1)
- **static const EBVectors ZeroEBfield**
  The constant representing a zero electromagnetic field. (see Section 2.49.1)
- **static const EVector ZeroEfield**
  The constant representing a zero electric field. (see Section 2.49.1)

- **const EMField& operator= (const EMField&)**
  (see Section 2.49.1)
- **virtual void scale (double)**
  Scale the field. (see Section 2.49.1)
- **virtual ~StaticMagneticField ()**
  (see Section 2.150.1)

2.150.1 Detailed descriptions

Public members

- **StaticMagneticField ()**
- **virtual ~StaticMagneticField ()**

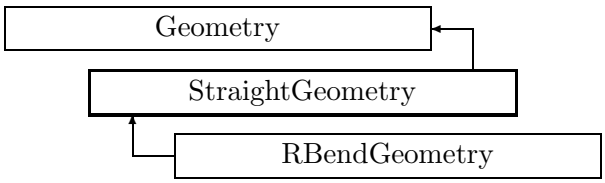

Figure 2.113: Inheritance for class StraightGeometry

2.151 class StraightGeometry

A geometry representing a straight line.
StraightGeometry represents a straight line segment along the local z-axis with no torsion. The origin is defined at the centre of the segment, and the geometry runs from -length/2 to +length/2. All transformations are correspondingly only simple translations along the z-axis.

Type:	Instantiable
Superclasses:	public Geometry
Include file:	./BeamlineGeometry/StraightGeometry.hh

Synopsis (including inherited members)

Public members

- **StraightGeometry (double)**
  Constructor. (see Section 2.151.1)
- **StraightGeometry (const StraightGeometry&)**
  (see Section 2.151.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.151.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.151.1)
- **double getEntrance ()const**
  Get entrance. (see Section 2.151.1)
- **Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.151.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.80.1)
- **double getExit ()const**
  Get exit. (see Section 2.151.1)
- **Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.151.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.80.1)

- **double getOrigin ()const**
  Get origin. (see Section 2.151.1)
- **Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.151.1)
- **Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.151.1)
- **Euclid3D getTransform (double)const**
  Get transform. (see Section 2.151.1)
- **const StraightGeometry& operator= (const StraightGeometry&)**
  (see Section 2.151.1)
- **const Geometry& operator= (const Geometry&)**
  (see Section 2.80.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.151.1)
- **virtual ~StraightGeometry ()**
  (see Section 2.151.1)

2.151.1 Detailed descriptions

Public members

- **StraightGeometry (double)**
  Constructor.

  Using the **length**.
- **StraightGeometry (const StraightGeometry&)**

- **virtual double getArcLength ()const**
  Get arc length.

  Return the arc length, identical to the straight design length.
- **virtual double getElementLength ()const**
  Get design length.

  Return the straight design length.
- **double getEntrance ()const**
  Get entrance.

  Return the arc length from the origin to the entrance of the element (entrance <= 0)
- **Euclid3D getEntranceFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getEntrance()). Return the transform of the local coordinate system from the origin to the entrance of the element.

- **double getExit ()const**
  Get exit.

  Return the arc length from the origin to the exit of the element (exit >= 0)

- **Euclid3D getExitFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getExit()). Return the transform of the local coordinate system from the origin to the exit of the element.

- **double getOrigin ()const**
  Get origin.

  Return the arc length from the entrance to the origin of the element (origin >= 0)

- **Euclid3D getTotalTransform ()const**
  Get transform.

  Equivalent to getTransform(getEntrance(), getExit()). Return the transform of the local coordinate system from the entrance to the exit of the element.

- **Euclid3D getTransform (double,double)const**
  Get transform.

  Return the transform of the local coordinate system from the position **fromS** to the position **toS**.

- **Euclid3D getTransform (double)const**
  Get transform.

  Equivalent to getTransform(0.0, s). Return the transform of the local coordinate system from the origin and **s**.

- **const StraightGeometry& operator= (const StraightGeometry&)**

- **virtual void setElementLength (double)**
  Set design length.

  Assign the straight design length.

- **virtual ~StraightGeometry ()**

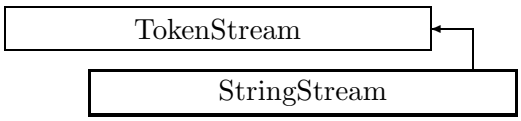

Figure 2.114: Inheritance for class StringStream

2.152 class StringStream

A stream of input tokens.
The source of tokens is a C++ string.

Type:	Instantiable
Superclasses:	public TokenStream
Include file:	./Parser/StringStream.hh

Synopsis (including inherited members)

Public members

- **StringStream (const string&)**
  Constructor. (see Section 2.152.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **int getLine ()const**
  Return line number. (see Section 2.160.1)
- **const string& getName ()const**
  Return stream name. (see Section 2.160.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **void putBack (const Token&)**
  Put token back to stream. (see Section 2.160.1)
- **virtual Token readToken ()**
  Read single token from file. (see Section 2.152.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ~StringStream ()**
  (see Section 2.152.1)

2.152.1 Detailed descriptions

Public members

- **StringStream (const string&)**
  Constructor.

  Initialize stream to read from the named file.

- **virtual Token readToken ()**
  Read single token from file.
- **virtual ~StringStream ()**

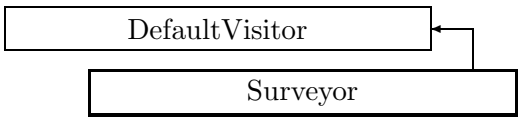

Figure 2.115: Inheritance for class Surveyor

2.153 class Surveyor

Survey algorithm.
This visitor class computes the survey of a beam line.

Type:	Instantiable
Superclasses:	public DefaultVisitor
Include file:	./Algorithms/Surveyor.hh

Synopsis (including inherited members)

Public members

- **Surveyor (Beamline&,bool)**
  Constructor. (see Section 2.153.1)
- **Surveyor (Beamline&,double,double,double,double,double,double,bool)**
  Constructor. (see Section 2.153.1)
- **Surveyor (Beamline&,const Euclid3D&,bool)**
  Constructor. (see Section 2.153.1)
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **void getMap (Euclid3D&)const**
  Return accumulated map. (see Section 2.153.1)
- **void setMap (const Euclid3D&)**
  Reset accumulated map for restart. (see Section 2.153.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper.. (see Section 2.37.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam. (see Section 2.37.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.37.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.37.1)

- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector. (see Section 2.37.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.37.1)
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic. (see Section 2.37.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift. (see Section 2.37.1)
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.37.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson. (see Section 2.37.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.37.1)
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker. (see Section 2.37.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor. (see Section 2.37.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole. (see Section 2.37.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.37.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.153.1)
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend. (see Section 2.37.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.37.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity. (see Section 2.37.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole. (see Section 2.37.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend. (see Section 2.37.1)

- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.37.1)
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator. (see Section 2.37.1)
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum. (see Section 2.37.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid. (see Section 2.37.1)
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.37.1)
- **virtual ~Surveyor ()**
  (see Section 2.153.1)

2.153.1 Detailed descriptions

Public members

- **Surveyor (Beamline&,bool)**
  Constructor.

  Assume zero initial conditions. The beam line to be tracked is **bl**. If **revTrack** is true, track from s = C to s = 0.
- **Surveyor (Beamline&,double,double,double,double,double,double,bool)**
  Constructor.

  Use given initial conditions. The beam line to be tracked is **bl**. If **revTrack** is true, track from s = C to s = 0.
- **Surveyor (Beamline&,const Euclid3D&,bool)**
  Constructor.

  Use given initial conditions in terms of an Euclid3D object. The beam line to be tracked is **bl**. If **revTrack** is true, track from s = C to s = 0.
- **void getMap (Euclid3D&)const**
  Return accumulated map.
- **void setMap (const Euclid3D&)**
  Reset accumulated map for restart.
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch.
- **virtual ~Surveyor ()**

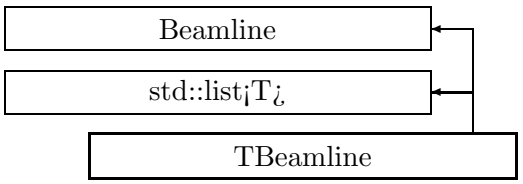

Figure 2.116: Inheritance for class TBeamline

2.154 template class TBeamline <class>

Template class for beam lines.
Instantiation with different T types allows attachment of additional data to each position in the line.

Type:	Instantiable
Superclasses:	public Beamline, public std::list<T>
Include file:	./Beamlines/TBeamline.hh

Synopsis (including inherited members)

Public members

- **TBeamline ()**
  Default constructor. (see Section 2.154.1)
- **TBeamline (const string&)**
  Constructor with given name. (see Section 2.154.1)
- **TBeamline (const TBeamline<T>&)**
  (see Section 2.154.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply BeamlineVisitor to this line. (see Section 2.154.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual void append (const T&)**
  Append a T object. (see Section 2.154.1)
- **virtual TBeamline<T>* clone ()const**
  Make clone. (see Section 2.154.1)
- **virtual TBeamline<T>* copyStructure ()**
  Make structure copy. (see Section 2.154.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.154.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)

- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.154.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.154.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.154.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.154.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.154.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BeamlineGeometry& getGeometry ()**
  Get geometry. (see Section 2.154.1)
- **virtual const BeamlineGeometry& getGeometry ()const**
  Get geometry. (see Section 2.154.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.154.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.154.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.154.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.154.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.154.1)
- **virtual const string& getType ()const**
  Get beamline type string. (see Section 2.154.1)

- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual void iterate (BeamlineVisitor&,bool)const**
  Apply visitor to all elements of the line. (see Section 2.154.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.154.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual void prepend (const T&)**
  Prepend a T object. (see Section 2.154.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)

- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~TBeamline ()**
  (see Section 2.154.1)

Protected members

- **BeamlineGeometry itsGeometry**
  The beamline geometry. (see Section 2.154.1)
- **RCObject& operator= (const RCObject&)**
  (see Section 2.126.1)
- **mutable bool shareFlag**
  (see Section 2.52.1)

2.154.1 Detailed descriptions

Public members

- **TBeamline ()**
  Default constructor.
- **TBeamline (const string&)**
  Constructor with given name.
- **TBeamline (const TBeamline<T>&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply BeamlineVisitor to this line.
- **virtual void append (const T&)**
  Append a T object.
- **virtual TBeamline<T>* clone ()const**
  Make clone.
- **virtual TBeamline<T>* copyStructure ()**
  Make structure copy.
- **virtual double getArcLength ()const**
  Get arc length.

  Return the length of the geometry, measured along the design orbit.

- **virtual double getElementLength ()const**
  Get design length.

  Return the length of the geometry, measured along the design polygone.

- **virtual double getEntrance ()const**
  Get entrance position.

  Return the arc length from the origin to the entrance of the geometry (non-positive).

- **virtual Euclid3D getEntranceFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getEntrance()). Return the transform of the local coordinate system from the origin to the entrance of the element.

- **virtual double getExit ()const**
  Get exit position.

  Return the arc length from the origin to the exit of the geometry (non-negative).

- **virtual Euclid3D getExitFrame ()const**
  Get transform.

  Equivalent to getTransform(0.0, getExit()). Return the transform of the local coordinate system from the origin to the exit of the element.

- **virtual BeamlineGeometry& getGeometry ()**
  Get geometry.

  Version for non-constant object.

- **virtual const BeamlineGeometry& getGeometry ()const**
  Get geometry.

  Version for constant object.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual double getOrigin ()const**
  Get origin position.

  Return the arc length from the entrance to the origin of the geometry (non-negative).

- **virtual Euclid3D getTotalTransform ()const**
  Get transform.

  Equivalent to getTransform(getEntrance(), getExit()). Return the transform of the local coordinate system from the entrance to the exit of the element.

- **virtual Euclid3D getTransform (double,double)const**
  Get transform.

  Return the transform of the local coordinate system from the position **fromS** to the position **toS**.

- **virtual Euclid3D getTransform (double)const**
  Get transform.

  Equivalent to getTransform(0.0, s). Return the transform of the local coordinate system from the origin and **s**.

- **virtual const string& getType ()const**
  Get beamline type string.

- **virtual void iterate (BeamlineVisitor&,bool)const**
  Apply visitor to all elements of the line.

  If the parameter **r2l** is true, the line is traversed from right (s=C) to left (s=0). If any error occurs, this method may throw an exception.

- **virtual void makeSharable ()**
  Set sharable flag.

  The whole beamline and the elements depending on **this** are marked as sharable. After this call a **copyStructure()** call reuses the element.

- **virtual void prepend (const T&)**
  Prepend a T object.

- **virtual ~TBeamline ()**

Protected members

- **BeamlineGeometry itsGeometry**
  The beamline geometry.

  Exists to match the interface for ElementBase.

2.155 template class Taylor <class>

A representation for a Taylor series in one variable,
whose coefficients are of type T. For some operations this class assumes that the type T has particular operations, like PoissonBracket. The class Taylor<T> is mainly meant for implementing a Lie generator with one parameter as a Taylor series in this parameter. This permits easy integration with respect to the parameter.

Type:	Instantiable
Include file:	./FixedAlgebra/Taylor.hh

Synopsis (including inherited members)

Public members

- **Taylor (int)**
  Construct a zero series of a given order. (see Section 2.155.1)
- **Taylor ()**
  (see Section 2.155.1)
- **Taylor (const Taylor&)**
  (see Section 2.155.1)
- **inline T* begin ()**
  Get pointer to beginning of series (zero-order term). (see Section 2.155.1)
- **inline const T* begin ()const**
  Get pointer to beginning of series (zero-order term). (see Section 2.155.1)
- **void clear ()**
  Clear all coefficients. (see Section 2.155.1)
- **inline T* end ()**
  Get pointer to end of series (one beyond highest term). (see Section 2.155.1)
- **inline const T* end ()const**
  Get pointer to end of series (one beyond highest term). (see Section 2.155.1)
- **inline int getOrder ()const**
  Return order of this series. (see Section 2.155.1)
- **Taylor integrate ()const**
  Integrate with respect to the variable. (see Section 2.155.1)
- **Taylor& operator*= (const T&)**
  Multiply by scalar and assign. (see Section 2.155.1)
- **Taylor& operator+= (const Taylor&)**
  Add series and assign. (see Section 2.155.1)
- **Taylor operator- ()const**
  Change sign of series. (see Section 2.155.1)

- **Taylor& operator-= (const Taylor&)**
  Subtract series and assign. (see Section 2.155.1)
- **Taylor& operator/= (const T&)**
  Divide by scalar and assign. (see Section 2.155.1)
- **const Taylor& operator= (const Taylor&)**
  (see Section 2.155.1)
- **T& operator[] (int)**
  Get coefficient. (see Section 2.155.1)
- **const T& operator[] (int)const**
  Get coefficient. (see Section 2.155.1)
- **T sum ()const**
  Return sum of series. (see Section 2.155.1)
- **~Taylor ()**
  (see Section 2.155.1)

2.155.1 Detailed descriptions

Public members

- **Taylor (int)**
  Construct a zero series of a given order.
- **Taylor ()**
- **Taylor (const Taylor&)**
- **inline T* begin ()**
  Get pointer to beginning of series (zero-order term).

  Version for non-constant series.
- **inline const T* begin ()const**
  Get pointer to beginning of series (zero-order term).

  Version for constant series.
- **void clear ()**
  Clear all coefficients.
- **inline T* end ()**
  Get pointer to end of series (one beyond highest term).

  Version for non-constant series.
- **inline const T* end ()const**
  Get pointer to end of series (one beyond highest term).

  Version for constant series.

- **inline int getOrder ()const**
  Return order of this series.
- **Taylor integrate ()const**
  Integrate with respect to the variable.
- **Taylor& operator*= (const T&)**
  Multiply by scalar and assign.
- **Taylor& operator+= (const Taylor&)**
  Add series and assign.
- **Taylor operator- ()const**
  Change sign of series.
- **Taylor& operator-= (const Taylor&)**
  Subtract series and assign.
- **Taylor& operator/= (const T&)**
  Divide by scalar and assign.
- **const Taylor& operator= (const Taylor&)**
- **T& operator[] (int)**
  Get coefficient.

  Return a reference to coefficient of order **n**. Result is undefined, if **n** is out of range.
- **const T& operator[] (int)const**
  Get coefficient.

  Return a reference to constant coefficient of order **n**. Result is undefined, if **n** is out of range.
- **T sum ()const**
  Return sum of series.
- **~Taylor ()**

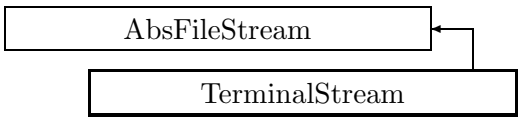

Figure 2.117: Inheritance for class TerminalStream

2.156 class TerminalStream

A stream of input tokens.
The source of tokens is the terminal.

Type:	Instantiable
Superclasses:	public AbsFileStream
Include file:	./Parser/TerminalStream.hh

Synopsis (including inherited members)

Public members

- **TerminalStream (const char[])**
  Constructor. (see Section 2.156.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual bool fillLine ()**
  Read next input line. (see Section 2.156.1)
- **int getLine ()const**
  Return line number. (see Section 2.160.1)
- **const string& getName ()const**
  Return stream name. (see Section 2.160.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **void putBack (const Token&)**
  Put token back to stream. (see Section 2.160.1)
- **virtual Token readToken ()**
  Read single token from file. (see Section 2.2.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ~TerminalStream ()**
  (see Section 2.156.1)

2.156.1 Detailed descriptions

Public members

- **TerminalStream (const char[])**
  Constructor.

  The C-style string program name may be used in the readline package.

- **virtual bool fillLine ()**
  Read next input line.

- **virtual ~TerminalStream ()**

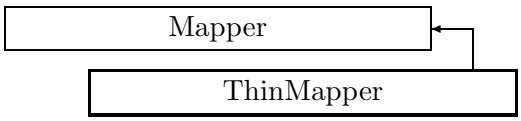

Figure 2.118: Inheritance for class ThinMapper

2.157 class ThinMapper

Construct thin lens map.
The visitor class for building a FVps<double,6> for a beamline using a thin-lens approximation for all elements.
Approximations used:

> All active elements are represented as thin lenses, sandwiched between two drifts, each half of the element length.
>
> Drifts are handled with a second-order approximation.
>
> Geometric transformations ignore rotations about transverse axes and translations along the design orbit and truncate after second order.

Type:	Instantiable
Superclasses:	public Mapper
Include file:	./Algorithms/ThinMapper.hh

Synopsis (including inherited members)

Public members

- **ThinMapper (const Beamline&,const PartData&,bool,bool)**
  Constructor. (see Section 2.157.1)
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **virtual void getMap (LinearMap<double,6>&)const**
  Return the linear part of the accumulated map. (see Section 2.93.1)
- **virtual void getMap (TransportMap<double,6>&)const**
  Return the second-order part of the accumulated map. (see Section 2.93.1)
- **virtual void getMap (FVps<double,6>&)const**
  Return the full map accumulated so far. (see Section 2.93.1)
- **virtual void setMap (const LinearMap<double,6>&)**
  Reset the linear part of the accumulated map for restart. (see Section 2.93.1)
- **virtual void setMap (const TransportMap<double,6>&)**
  Reset the second-order part of the accumulated map for restart. (see Section 2.93.1)
- **virtual void setMap (const FVps<double,6>&)**
  Reset the full map for restart. (see Section 2.93.1)

- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper. (see Section 2.93.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam. (see Section 2.157.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.157.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.93.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector. (see Section 2.157.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.37.1)
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic. (see Section 2.157.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift. (see Section 2.157.1)
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.37.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson. (see Section 2.157.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.93.1)
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker. (see Section 2.157.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor. (see Section 2.157.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole. (see Section 2.157.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.37.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.93.1)

- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend. (see Section 2.157.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.37.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity. (see Section 2.157.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole. (see Section 2.157.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend. (see Section 2.157.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.37.1)
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator. (see Section 2.157.1)
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum. (see Section 2.157.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid. (see Section 2.157.1)
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.37.1)
- **virtual ~ThinMapper ()**
  (see Section 2.157.1)

Protected members

- **void applyDrift (double)**
  (see Section 2.157.1)
- **void applyThinMultipole (const BMultipoleField&,double)**
  Thin multipole kick. (see Section 2.93.1)
- **void applyThinSBend (const BMultipoleField&,double,double)**
  Thin SBend kick. (see Section 2.93.1)
- **void applyTransform (const Euclid3D&,double)**
  Apply transform. (see Section 2.93.1)
- **bool back_beam**
  (see Section 2.37.1)
- **bool back_track**
  (see Section 2.37.1)

- **FTps<double,6> buildMultipoleVectorPotential (const BMultipoleField&)**
  Construct the vector potential for a Multipole. (see Section 2.3.1)
- **FTps<double,6> buildSBendVectorPotential (const BMultipoleField&,double)**
  Construct the vector potential for a SBend. (see Section 2.3.1)
- **double flip_B**
  (see Section 2.37.1)
- **double flip_s**
  (see Section 2.37.1)
- **const Beamline& itsLine**
  (see Section 2.37.1)
- **FVps<double,6> itsMap**
  The transfer map being built. (see Section 2.93.1)
- **const PartData itsReference**
  The reference information. (see Section 2.3.1)
- **bool local_flip**
  (see Section 2.37.1)

2.157.1 Detailed descriptions

Public members

- **ThinMapper (const Beamline&,const PartData&,bool,bool)**
  Constructor.

  The beam line to be tracked is **bl**. The particle reference data are taken from **data**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTracl** is true, we track against the beam.
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam.
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator.
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector.
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic.
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift.
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson.
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker.

- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor.
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole.
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend.
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity.
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole.
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend.
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator.
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum.
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid.
- **virtual ~ThinMapper ()**

Protected members

- **void applyDrift (double)**

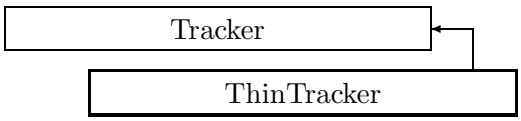

Figure 2.119: Inheritance for class ThinTracker

2.158 class ThinTracker

Track with thin lens algorithm.
The visitor class for tracking a bunch of particles through a beamline using a thin-lens approximation for all elements.
Approximations used:

> All active elements are represented as thin lenses, sandwiched between two drifts, each half of the element length.
>
> Drifts are handled with a second-order approximation.
>
> Geometric transformations ignore rotations about transverse axes and translations along the design orbit and truncate after second order.

Type:	Instantiable
Superclasses:	public Tracker
Include file:	./Algorithms/ThinTracker.hh

Synopsis (including inherited members)

Public members

- **ThinTracker (const Beamline&,const PartData&,bool,bool)**
  Constructor. (see Section 2.158.1)
- **ThinTracker (const Beamline&,const PartBunch&,const PartData&,bool,bool)**
  Constructor. (see Section 2.158.1)
- **void addToBunch (const Particle&)**
  Add particle to bunch. (see Section 2.166.1)
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **const PartBunch& getBunch ()const**
  Return the current bunch. (see Section 2.166.1)
- **typedef PartBunch::iterator iterator**
  (see Section 2.166.1)
- **void setBunch (const PartBunch&)**
  Store the bunch. (see Section 2.166.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper. (see Section 2.166.1)

- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply algorithm to BeamBeam. (see Section 2.158.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply algorithm to Collimator. (see Section 2.158.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.166.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply algorithm to Corrector. (see Section 2.158.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.37.1)
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply algorithm to Diagnostic. (see Section 2.158.1)
- **virtual void visitDrift (const Drift&)**
  Apply algorithm to Drift. (see Section 2.158.1)
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.37.1)
- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply algorithm to Lambertson. (see Section 2.158.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.166.1)
- **virtual void visitMarker (const Marker&)**
  Apply algorithm to Marker. (see Section 2.158.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply algorithm to Monitor. (see Section 2.158.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply algorithm to Multipole. (see Section 2.158.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.37.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.166.1)
- **virtual void visitRBend (const RBend&)**
  Apply algorithm to RBend. (see Section 2.158.1)

- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.37.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply algorithm to RFCavity. (see Section 2.158.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply algorithm to RFQuadrupole. (see Section 2.158.1)
- **virtual void visitSBend (const SBend&)**
  Apply algorithm to SBend. (see Section 2.158.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.37.1)
- **virtual void visitSeparator (const Separator&)**
  Apply algorithm to Separator. (see Section 2.158.1)
- **virtual void visitSeptum (const Septum&)**
  Apply algorithm to Septum. (see Section 2.158.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply algorithm to Solenoid. (see Section 2.158.1)
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.166.1)
- **virtual ~ThinTracker ()**
  (see Section 2.158.1)

2.158.1 Detailed descriptions

Public members

- **ThinTracker (const Beamline&,const PartData&,bool,bool)**
  Constructor.

  The beam line to be tracked is **bl**. The particle reference data are taken from **data**. The particle bunch tracked is initially empty. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.
- **ThinTracker (const Beamline&,const PartBunch&,const PartData&,bool,bool)**
  Constructor.

  The beam line to be tracked is **bl**. The particle reference data are taken from **data**. The particle bunch tracked is taken from **bunch**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply algorithm to BeamBeam.
- **virtual void visitCollimator (const Collimator&)**
  Apply algorithm to Collimator.

- **virtual void visitCorrector (const Corrector&)**
  Apply algorithm to Corrector.
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply algorithm to Diagnostic.
- **virtual void visitDrift (const Drift&)**
  Apply algorithm to Drift.
- **virtual void visitLambertson (const Lambertson&)**
  Apply algorithm to Lambertson.
- **virtual void visitMarker (const Marker&)**
  Apply algorithm to Marker.
- **virtual void visitMonitor (const Monitor&)**
  Apply algorithm to Monitor.
- **virtual void visitMultipole (const Multipole&)**
  Apply algorithm to Multipole.
- **virtual void visitRBend (const RBend&)**
  Apply algorithm to RBend.
- **virtual void visitRFCavity (const RFCavity&)**
  Apply algorithm to RFCavity.
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply algorithm to RFQuadrupole.
- **virtual void visitSBend (const SBend&)**
  Apply algorithm to SBend.
- **virtual void visitSeparator (const Separator&)**
  Apply algorithm to Separator.
- **virtual void visitSeptum (const Septum&)**
  Apply algorithm to Septum.
- **virtual void visitSolenoid (const Solenoid&)**
  Apply algorithm to Solenoid.
- **virtual ~ThinTracker ()**

2.159 class Token

Representation of a single input token.
All tokens contain the name of the input stream and the line number where they came from.

Type:	Instantiable
Include file:	./Parser/Token.hh

Synopsis (including inherited members)

Public members

- **Token ()**
  Constructor. (see Section 2.159.1)
- **Token (const string&,int,Type,char)**
  Constructor. (see Section 2.159.1)
- **Token (const string&,int,Type,const char*)**
  Constructor. (see Section 2.159.1)
- **Token (const string&,int,Type,const string&)**
  Constructor. (see Section 2.159.1)
- **Token (const string&,int,const string&,double)**
  Constructor. (see Section 2.159.1)
- **Token (const string&,int,const string&,int)**
  Constructor. (see Section 2.159.1)
- **Token (const Token&)**
  (see Section 2.159.1)
- **enum Type**
  Possible token types. (see Section 2.159.1)
- **bool getBool ()const**
  Return boolean value. (see Section 2.159.1)
- **const string& getFile ()const**
  Return the token's file name. (see Section 2.159.1)
- **int getInteger ()const**
  Return integer value. (see Section 2.159.1)
- **const string& getLex ()const**
  Return the lexeme. (see Section 2.159.1)
- **int getLine ()const**
  Return the token's line number. (see Section 2.159.1)
- **double getReal ()const**
  Return real value. (see Section 2.159.1)

- **string getString ()const**
  Return string value. (see Section 2.159.1)
- **Type getType ()const**
  Return the token type. (see Section 2.159.1)
- **string getWord ()const**
  Return word value. (see Section 2.159.1)
- **bool isDel (char)const**
  Test for delimiter. (see Section 2.159.1)
- **bool isDel (const char*)const**
  Test for delimiter. (see Section 2.159.1)
- **bool isDel ()const**
  Test for any delimiter. (see Section 2.159.1)
- **bool isEOF ()const**
  Test for end of file. (see Section 2.159.1)
- **bool isError ()const**
  Test for error. (see Section 2.159.1)
- **bool isInteger ()const**
  Test for integer. (see Section 2.159.1)
- **bool isKey (const char*)const**
  Test for keyword. (see Section 2.159.1)
- **bool isReal ()const**
  Test for real number. (see Section 2.159.1)
- **bool isString ()const**
  Test for string. (see Section 2.159.1)
- **bool isWord ()const**
  Test for word. (see Section 2.159.1)
- **const Token& operator= (const Token&)**
  (see Section 2.159.1)
- **~Token ()**
  (see Section 2.159.1)

2.159.1 Detailed descriptions

Public members

- **Token ()**
  Constructor.

  Construct empty token.

- **Token (const string&,int,Type,char)**
  Constructor.

  Construct character token with type **type** and value **c**.

- **Token (const string&,int,Type,const char*)**
  Constructor.

  Construct string token with type **type** and value **s**.

- **Token (const string&,int,Type,const string&)**
  Constructor.

  Construct string token with type **type** and value **lex**.

- **Token (const string&,int,const string&,double)**
  Constructor.

  Construct real numeric token with lexeme **lex** and value **value**.

- **Token (const string&,int,const string&,int)**
  Constructor.

  Construct integer token with lexeme **lex** and value **value**.

- **Token (const Token&)**

- **enum Type**
  Possible token types.

- **bool getBool ()const**
  Return boolean value.

  Throw ParseError, if token is not boolean.

- **const string& getFile ()const**
  Return the token's file name.

- **int getInteger ()const**
  Return integer value.

  Throw ParseError, if token is not numeric.

- **const string& getLex ()const**
  Return the lexeme.

- **int getLine ()const**
  Return the token's line number.

- **double getReal ()const**
  Return real value.

  Throw ParseError, if token is not numeric.

- **string getString ()const**
  Return string value.

  Throw ParseError, if token is not string.

- **Type getType ()const**
  Return the token type.
- **string getWord ()const**
  Return word value.

  Throw ParseError, if token is not word.
- **bool isDel (char)const**
  Test for delimiter.

  Return true, if token is single character **del**.
- **bool isDel (const char*)const**
  Test for delimiter.

  Return true, if token is character string **del**.
- **bool isDel ()const**
  Test for any delimiter.
- **bool isEOF ()const**
  Test for end of file.
- **bool isError ()const**
  Test for error.
- **bool isInteger ()const**
  Test for integer.
- **bool isKey (const char*)const**
  Test for keyword.
- **bool isReal ()const**
  Test for real number.
- **bool isString ()const**
  Test for string.
- **bool isWord ()const**
  Test for word.
- **const Token& operator= (const Token&)**
- **~Token ()**

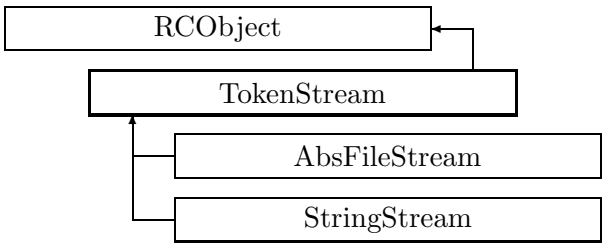

Figure 2.120: Inheritance for class TokenStream

2.160 class TokenStream

Abstract interface for a stream of input tokens.

Type:	abstract
Superclasses:	public RCObject
Include file:	./Parser/TokenStream.hh

Synopsis (including inherited members)

Public members

- **TokenStream (const string&)**
  Constructor. (see Section 2.160.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **int getLine ()const**
  Return line number. (see Section 2.160.1)
- **const string& getName ()const**
  Return stream name. (see Section 2.160.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **void putBack (const Token&)**
  Put token back to stream. (see Section 2.160.1)
- **virtual Token readToken ()**
  Read single token from stream. (see Section 2.160.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ~TokenStream ()**
  (see Section 2.160.1)

Protected members

- **int curr_line**
  (see Section 2.160.1)
- **RCObject& operator= (const RCObject&)**
  (see Section 2.126.1)
- **Token put_back**
  (see Section 2.160.1)
- **bool put_back_flag**
  (see Section 2.160.1)
- **string stream_name**
  (see Section 2.160.1)

2.160.1 Detailed descriptions

Public members

- **TokenStream (const string&)**
  Constructor.

  Store the stream name.
- **int getLine ()const**
  Return line number.
- **const string& getName ()const**
  Return stream name.
- **void putBack (const Token&)**
  Put token back to stream.

  This allows to reparse the token.
- **virtual Token readToken ()**
  Read single token from stream.
- **virtual ~TokenStream ()**

Protected members

- **int curr_line**
- **Token put_back**
- **bool put_back_flag**
- **string stream_name**

2.161 template class Tps <class>

Truncated power series.
Truncated power series in **n** variables of type **T**. All divide operations throw DivideError, if the constant part of the divisor is zero. All operations throw SizeError, if the operands are not consistent. All operations throw RangeError, if an index is out of range.

Type:	Instantiable
Include file:	./Algebra/Tps.hh

Synopsis (including inherited members)

Public members

- **static const int EXACT**
  Representation of infinite precision. (see Section 2.161.1)
- **Tps<T> Taylor (const T[],int)const**
  Taylor series. (see Section 2.161.1)
- **Tps (int,int)**
  Constructor. (see Section 2.161.1)
- **Tps (const T&)**
  Conversion. (see Section 2.161.1)
- **Tps (int)**
  Conversion. (see Section 2.161.1)
- **Tps ()**
  (see Section 2.161.1)
- **Tps (const Tps<T>&)**
  (see Section 2.161.1)
- **void clear ()**
  Set to zero. (see Section 2.161.1)
- **Tps<T> derivative (int)const**
  Partial derivative. (see Section 2.161.1)
- **T evaluate (const Vector<T>&)const**
  Substitute. (see Section 2.161.1)
- **Tps<T> filter (int,int)const**
  Extract orders. (see Section 2.161.1)
- **std::istream& get (std::istream&)**
  Get Tps from the stream is. (see Section 2.161.1)
- **const T getCoefficient (int)const**
  Get coefficient. (see Section 2.161.1)
- **const T getCoefficient (const TpsMonomial&)const**
  Get coefficient. (see Section 2.161.1)

- **const TpsMonomial& getExponents (int)const**
  Get exponents. (see Section 2.161.1)
- **static int getGlobalTruncOrder ()**
  Get global truncation order. (see Section 2.161.1)
- **int getMaxOrder ()const**
  Get maximal order. (see Section 2.161.1)
- **int getOrder (int)const**
  Get order. (see Section 2.161.1)
- **int getSize ()const**
  Get number of coefficients. (see Section 2.161.1)
- **int getSize (int)const**
  Size. (see Section 2.161.1)
- **int getTruncOrder ()const**
  Get truncation order. (see Section 2.161.1)
- **int getVariables ()const**
  Get number of variables. (see Section 2.161.1)
- **Tps<T> integral (int)const**
  Partial integral. (see Section 2.161.1)
- **Tps<T> inverse (int)const**
  Reciprocal value. (see Section 2.161.1)
- **bool isConstant ()const**
  Test for constant. (see Section 2.161.1)
- **static Tps<T> makeMonomial (const TpsMonomial&,const T&)**
  Make monomial. (see Section 2.161.1)
- **static Tps<T> makeVarPower (int,int,int)**
  Make power. (see Section 2.161.1)
- **static Tps<T> makeVariable (int,int)**
  Make variable. (see Section 2.161.1)
- **Tps<T> multiply (const Tps<T>&,int)const**
  Truncated multiplication. (see Section 2.161.1)
- **Tps<T> multiplyVariable (int)const**
  Multiply by variable **var**. (see Section 2.161.1)
- **bool operator!= (const Tps<T>&)const**
  Inequality operator. (see Section 2.161.1)
- **bool operator!= (const T&)const**
  Inequality with constant. (see Section 2.161.1)

- **Tps<T>& operator*= (const Tps<T>&)**
  Multiply and assign. (see Section 2.161.1)
- **Tps<T>& operator*= (const T&)**
  Multiply by constant and assign. (see Section 2.161.1)
- **Tps<T> operator+ ()const**
  Unary plus. (see Section 2.161.1)
- **Tps<T>& operator+= (const Tps<T>&)**
  Add and assign. (see Section 2.161.1)
- **Tps<T>& operator+= (const T&)**
  Add constant and assign. (see Section 2.161.1)
- **Tps<T> operator- ()const**
  Unary minus. (see Section 2.161.1)
- **Tps<T>& operator-= (const Tps<T>&)**
  Subtract and assign. (see Section 2.161.1)
- **Tps<T>& operator-= (const T&)**
  Subtract constant and assign. (see Section 2.161.1)
- **Tps<T>& operator/= (const Tps<T>&)**
  Divide and assign. (see Section 2.161.1)
- **Tps<T>& operator/= (const T&)**
  Divide by constant and assign. (see Section 2.161.1)
- **Tps<T>& operator= (const Tps<T>&)**
  (see Section 2.161.1)
- **Tps<T>& operator= (const T&)**
  Convert and assign. (see Section 2.161.1)
- **bool operator== (const Tps<T>&)const**
  Equality operator. (see Section 2.161.1)
- **bool operator== (const T&)const**
  Equlatity with constant. (see Section 2.161.1)
- **const T operator[] (int)const**
  Get coefficient. (see Section 2.161.1)
- **T& operator[] (int)**
  Get coefficient. (see Section 2.161.1)
- **const T operator[] (const TpsMonomial&)const**
  Get coefficient. (see Section 2.161.1)
- **T& operator[] (const TpsMonomial&)**
  Get coefficient. (see Section 2.161.1)

- **std::ostream& put (std::ostream&)const**
  Put Tps to the stream is. (see Section 2.161.1)
- **Tps<T> scaleMonomials (const Tps<T>&)const**
  Multiply monomial-wise. (see Section 2.161.1)
- **void setCoefficient (int,const T&)**
  Set coefficient. (see Section 2.161.1)
- **void setCoefficient (const TpsMonomial&,const T&)**
  Set coefficient. (see Section 2.161.1)
- **static void setGlobalTruncOrder (int)**
  Set global truncation order. (see Section 2.161.1)
- **Tps<T> substitute (const Matrix<T>&)const**
  Substitute. (see Section 2.161.1)
- **Tps<T> substitute (const VpsMap<T>&)const**
  Substitute. (see Section 2.161.1)
- **Tps<T> truncate (int)**
  Truncate. (see Section 2.161.1)
- **~Tps ()**
  (see Section 2.161.1)

2.161.1 Detailed descriptions

Public members

- **static const int EXACT**
  Representation of infinite precision.
- **Tps<T> Taylor (const T[],int)const**
  Taylor series.

  Expand truncated Taylor series with coefficiens **series** and order **n**.
- **Tps (int,int)**
  Constructor.

  Define maximum order and number of variables, value = 0.
- **Tps (const T&)**
  Conversion.

  From constant **y**.
- **Tps (int)**
  Conversion.

  From int **y**.
- **Tps ()**

- **Tps (const Tps<T>&)**

- **void clear ()**
  Set to zero.

- **Tps<T> derivative (int)const**
  Partial derivative.

  Return partial derivative with respect to variable **var**. Return zero for a constant.

- **T evaluate (const Vector<T>&)const**
  Substitute.

  Substitute vector **v** in Tps, giving a constant.

- **Tps<T> filter (int,int)const**
  Extract orders.

  Return the orders **lowOrder** through **highOrder**, both included.

- **std::istream& get (std::istream&)**
  Get Tps from the stream is.

- **const T getCoefficient (int)const**
  Get coefficient.

  Return the coefficient denoted by the Giorgelli index. Throw RangeError, if index is out of range.

- **const T getCoefficient (const TpsMonomial&)const**
  Get coefficient.

  Return the coefficient denoted by monomial exponents. Throw RangeError, if index is out of range.

- **const TpsMonomial& getExponents (int)const**
  Get exponents.

  Return the exponent array for the given Giorgelli index.

- **static int getGlobalTruncOrder ()**
  Get global truncation order.

- **int getMaxOrder ()const**
  Get maximal order.

- **int getOrder (int)const**
  Get order.

  Return the order of the monomial with the given Giorgelli index.

- **int getSize ()const**
  Get number of coefficients.

- **int getSize (int)const**
  Size.

  Return the number of monomials required for a Tps of order **order**.

- **int getTruncOrder ()const**
  Get truncation order.
- **int getVariables ()const**
  Get number of variables.
- **Tps<T> integral (int)const**
  Partial integral.

  Return partial integral with respect to variable **var**. Throw LogicalError for a constant.
- **Tps<T> inverse (int)const**
  Reciprocal value.
- **bool isConstant ()const**
  Test for constant.
- **static Tps<T> makeMonomial (const TpsMonomial&,const T&)**
  Make monomial.

  Construct the monomial with the exponents in **m** and coefficient **t**.
- **static Tps<T> makeVarPower (int,int,int)**
  Make power.

  Construct power of variable **var**, with total of **nVar** variables.
- **static Tps<T> makeVariable (int,int)**
  Make variable.

  Construct the variable identified by the index **var**, with total of **nVar** variables.
- **Tps<T> multiply (const Tps<T>&,int)const**
  Truncated multiplication.
- **Tps<T> multiplyVariable (int)const**
  Multiply by variable var.

  Throw LogicalError for a constant.
- **bool operator!= (const Tps<T>&)const**
  Inequality operator.
- **bool operator!= (const T&)const**
  Inequality with constant.
- **Tps<T>& operator*= (const Tps<T>&)**
  Multiply and assign.
- **Tps<T>& operator*= (const T&)**
  Multiply by constant and assign.
- **Tps<T> operator+ ()const**
  Unary plus.
- **Tps<T>& operator+= (const Tps<T>&)**
  Add and assign.

- **Tps<T>& operator+= (const T&)**
  Add constant and assign.
- **Tps<T> operator- ()const**
  Unary minus.
- **Tps<T>& operator-= (const Tps<T>&)**
  Subtract and assign.
- **Tps<T>& operator-= (const T&)**
  Subtract constant and assign.
- **Tps<T>& operator/= (const Tps<T>&)**
  Divide and assign.
- **Tps<T>& operator/= (const T&)**
  Divide by constant and assign.
- **Tps<T>& operator= (const Tps<T>&)**
- **Tps<T>& operator= (const T&)**
  Convert and assign.
- **bool operator== (const Tps<T>&)const**
  Equality operator.
- **bool operator== (const T&)const**
  Equlatity with constant.
- **const T operator[] (int)const**
  Get coefficient.

  Return the coefficient denoted by the Giorgelli index. Result is undefined, if index is out of range.
- **T& operator[] (int)**
  Get coefficient.

  REturn a reference to the coefficient denoted by the Giorgelli index. Result is undefined, if index is out of range.
- **const T operator[] (const TpsMonomial&)const**
  Get coefficient.

  Return the coefficient denoted by monomial exponents. Result is undefined, if index is out of range.
- **T& operator[] (const TpsMonomial&)**
  Get coefficient.

  Return reference to the coefficient denoted by monomial exponents. Result is undefined, if index is out of range.
- **std::ostream& put (std::ostream&)const**
  Put Tps to the stream is.

- **Tps<T> scaleMonomials (const Tps<T>&)const**
  Multiply monomial-wise.

  Construct new Tps<T> by multipying corresponding monomials pair-wise.

- **void setCoefficient (int,const T&)**
  Set coefficient.

  Assign the coefficient denoted by the Giorgelli index. Expand size if index is out of range.

- **void setCoefficient (const TpsMonomial&,const T&)**
  Set coefficient.

  Assign the coefficient denoted by monomial exponents. Throw RangeError, if index is out of range.

- **static void setGlobalTruncOrder (int)**
  Set global truncation order.

- **Tps<T> substitute (const Matrix<T>&)const**
  Substitute.

  Substitute matrix **M** in Tps, giving new Tps.

- **Tps<T> substitute (const VpsMap<T>&)const**
  Substitute.

  Substitute map **m** in Tps, giving new Tps.

- **Tps<T> truncate (int)**
  Truncate.

  Change the maximum order to **trunc**; may also reserve more space.

- **~Tps ()**

2.162 class TpsData

Bookkeeping class for Tps<T>.
Internal utility class for Tps class. Not to be used directly in a program.

Type:	Instantiable
Include file:	./Algebra/TpsData.hh

Synopsis (including inherited members)

Public members

- **typedef Array2D<int> BinomialTable**
  (see Section 2.162.1)
- **typedef Array1D<TpsMonomial> ExponentTable**
  (see Section 2.162.1)
- **typedef Array1D<int> ProductRow**
  (see Section 2.162.1)
- **typedef Array1D<ProductRow> ProductTable**
  (see Section 2.162.1)
- **TpsData ()**
  (see Section 2.162.1)
- **const TpsMonomial& getExponents (int)const**
  (see Section 2.162.1)
- **int getOrder (int)const**
  (see Section 2.162.1)
- **const int* getProductArray (int)const**
  (see Section 2.162.1)
- **int getSize (int)const**
  (see Section 2.162.1)
- **const Array1D<TpsSubstitution>& getSubTable ()const**
  (see Section 2.162.1)
- **static TpsData* getTpsData (int,int)**
  (see Section 2.162.1)
- **int getVariables ()const**
  (see Section 2.162.1)
- **int indexMonomial (const TpsMonomial&)const**
  (see Section 2.162.1)
- **~TpsData ()**
  (see Section 2.162.1)

2.162.1 Detailed descriptions

Public members

- **typedef Array2D<int> BinomialTable**
- **typedef Array1D<TpsMonomial> ExponentTable**
- **typedef Array1D<int> ProductRow**
- **typedef Array1D<ProductRow> ProductTable**
- **TpsData ()**
- **const TpsMonomial& getExponents (int)const**
- **int getOrder (int)const**
- **const int* getProductArray (int)const**
- **int getSize (int)const**
- **const Array1D<TpsSubstitution>& getSubTable ()const**
- **static TpsData* getTpsData (int,int)**
- **int getVariables ()const**
- **int indexMonomial (const TpsMonomial&)const**
- **~TpsData ()**

2.163 class TpsMonomial

Exponent array for Tps<T>.
A representation of the exponents for a Tps monomial.

Type:	Instantiable
Include file:	./Algebra/TpsMonomial.hh

Synopsis (including inherited members)

Public members

- **TpsMonomial (int)**
  Constructor. (see Section 2.163.1)
- **TpsMonomial (int,int)**
  Constructor. (see Section 2.163.1)
- **TpsMonomial ()**
  (see Section 2.163.1)
- **TpsMonomial (const TpsMonomial&)**
  (see Section 2.163.1)
- **int getIndex ()const**
  Convert. (see Section 2.163.1)
- **int getOrder ()const**
  Get order. (see Section 2.163.1)
- **int getVariables ()const**
  Get variables. (see Section 2.163.1)
- **TpsMonomial operator* (const TpsMonomial&)const**
  Product. (see Section 2.163.1)
- **const TpsMonomial& operator= (const TpsMonomial&)**
  (see Section 2.163.1)
- **int& operator[] (int)**
  Get exponent. (see Section 2.163.1)
- **int operator[] (int)const**
  Get exponent. (see Section 2.163.1)
- **~TpsMonomial ()**
  (see Section 2.163.1)

2.163.1 Detailed descriptions

Public members

- **TpsMonomial (int)**
  Constructor.

  Array of zeros with length **nVar**, defines **nVar** variables with zero exponent each.

- **TpsMonomial (int,int)**
  Constructor.

  Array with length **nVar**, define variable with index **var**.

- **TpsMonomial ()**

- **TpsMonomial (const TpsMonomial&)**

- **int getIndex ()const**
  Convert.

  Convert the monomial to the corresponding Giorgelli index.

- **int getOrder ()const**
  Get order.

  Compute the monomial's order, i.e. the sum of all exponents.

- **int getVariables ()const**
  Get variables.

  Return the monomial's number of variables, i.e. the number of exponents.

- **TpsMonomial operator* (const TpsMonomial&)const**
  Product.

  Return the exponent set for the product of the monomials denoted by **this** and **rhs**, i.e. the sum of the exponents for each variable.

- **const TpsMonomial& operator= (const TpsMonomial&)**

- **int& operator[] (int)**
  Get exponent.

  Return a reference to the exponent of variable **index**.

- **int operator[] (int)const**
  Get exponent.

  Return value of the exponent of variable **index**.

- **~TpsMonomial ()**

2.164 template class TpsRep <class>

Type:	Instantiable

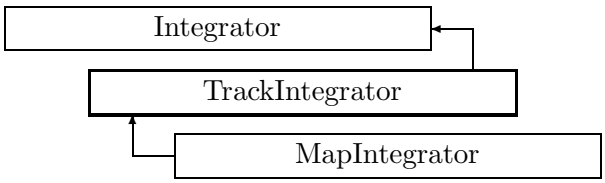

Figure 2.121: Inheritance for class TrackIntegrator

2.165 class TrackIntegrator

Integrate particle.
An abstract base class for all integrators capable of tracking particles or particle bunches through a beam element. It is assumed that this integrator has no maps available. It can therefore not track a map.

Type:	abstract
Superclasses:	public Integrator
Include file:	./Algorithms/TrackIntegrator.hh

Synopsis (including inherited members)

Public members

- **TrackIntegrator (ElementBase*)**
  (see Section 2.165.1)
- **TrackIntegrator (const TrackIntegrator&)**
  (see Section 2.165.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor. (see Section 2.165.1)
- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual TrackIntegrator* clone ()const**
  Make a clone. (see Section 2.165.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.52.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)

- **inline ElementBase* getElement ()const**
  Return the embedded element. (see Section 2.84.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual Geometry& getGeometry ()**
  Get geometry. (see Section 2.52.1)
- **virtual const Geometry& getGeometry ()const**
  Get geometry. (see Section 2.52.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.52.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.52.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)

- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.84.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track a particle bunch. (see Section 2.84.1)

- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.165.1)
- **virtual void trackParticle (Particle&,const PartData&,bool,bool)const**
  Track a particle. (see Section 2.84.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ˜TrackIntegrator ()**
  (see Section 2.165.1)

2.165.1 Detailed descriptions

Public members

- **TrackIntegrator (ElementBase*)**
- **TrackIntegrator (const TrackIntegrator&)**
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor.
- **virtual TrackIntegrator* clone ()const**
  Make a clone.
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map.

  The map is stored in **map**. The particle reference data are taken from **data**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam. This version throws LogicalError, since the integrator has no map.
- **virtual ˜TrackIntegrator ()**

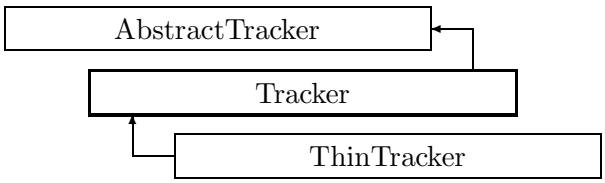

Figure 2.122: Inheritance for class Tracker

2.166 class Tracker

Track particles or bunches.
An abstract base class for all visitors capable of tracking particles through a beam element.
Phase space coordinates (in this order):

x: horizontal displacement (metres).

p_x/p_r: horizontal canonical momentum (no dimension).

y: vertical displacement (metres).

p_y/p_r: vertical canonical momentum (no dimension).

delta_p/p_r: relative momentum error (no dimension).

v*delta_t: time difference delta_t w.r.t. the reference frame which moves with uniform velocity

v_r = c*beta_r = p_r/m

along the design orbit, multiplied by the instantaneous velocity v of the particle (metres).

Where

p_r: is the constant reference momentum defining the reference frame velocity.

m: is the rest mass of the particles.

Other units used:

reference momentum: electron-volts.

accelerating voltage: volts.

separator voltage: volts.

frequencies: hertz.

phase lags: multiples of (2*pi).

Type:	abstract
Superclasses:	public AbstractTracker
Include file:	./Algorithms/Tracker.hh

Synopsis (including inherited members)

Public members

- **Tracker (const Beamline&,const PartData&,bool,bool)**
  Constructor. (see Section 2.166.1)
- **Tracker (const Beamline&,const PartBunch&,const PartData&,bool,bool)**
  Constructor. (see Section 2.166.1)
- **void addToBunch (const Particle&)**
  Add particle to bunch. (see Section 2.166.1)
- **virtual void execute ()**
  Apply the algorithm to the top-level beamline. (see Section 2.37.1)
- **const PartBunch& getBunch ()const**
  Return the current bunch. (see Section 2.166.1)
- **typedef PartBunch::iterator iterator**
  (see Section 2.166.1)
- **void setBunch (const PartBunch&)**
  Store the bunch. (see Section 2.166.1)
- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper. (see Section 2.166.1)
- **virtual void visitBeamBeam (const BeamBeam&)**
  Apply the algorithm to a beam-beam. (see Section 2.4.1)
- **virtual void visitBeamline (const Beamline&)**
  Apply the algorithm to a beam line. (see Section 2.37.1)
- **virtual void visitCollimator (const Collimator&)**
  Apply the algorithm to a collimator. (see Section 2.4.1)
- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component. (see Section 2.166.1)
- **virtual void visitCorrector (const Corrector&)**
  Apply the algorithm to a corrector. (see Section 2.4.1)
- **virtual void visitCorrectorWrapper (const CorrectorWrapper&)**
  Apply the algorithm to an corrector wrapper.. (see Section 2.37.1)
- **virtual void visitDiagnostic (const Diagnostic&)**
  Apply the algorithm to a diagnostic. (see Section 2.4.1)
- **virtual void visitDrift (const Drift&)**
  Apply the algorithm to a drift. (see Section 2.4.1)
- **virtual void visitFlaggedElmPtr (const FlaggedElmPtr&)**
  Apply the algorithm to a FlaggedElmPtr. (see Section 2.37.1)

- **virtual void visitIntegrator (const Integrator&)**
  Apply the algorithm to a generic integrator. (see Section 2.37.1)
- **virtual void visitLambertson (const Lambertson&)**
  Apply the algorithm to a Lambertson. (see Section 2.4.1)
- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping. (see Section 2.166.1)
- **virtual void visitMarker (const Marker&)**
  Apply the algorithm to a marker. (see Section 2.4.1)
- **virtual void visitMonitor (const Monitor&)**
  Apply the algorithm to a monitor. (see Section 2.4.1)
- **virtual void visitMultipole (const Multipole&)**
  Apply the algorithm to a multipole. (see Section 2.4.1)
- **virtual void visitMultipoleWrapper (const MultipoleWrapper&)**
  Apply the algorithm to an multipole wrapper.. (see Section 2.37.1)
- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch. (see Section 2.166.1)
- **virtual void visitRBend (const RBend&)**
  Apply the algorithm to a rectangular bend. (see Section 2.4.1)
- **virtual void visitRBendWrapper (const RBendWrapper&)**
  Apply the algorithm to an RBend wrapper.. (see Section 2.37.1)
- **virtual void visitRFCavity (const RFCavity&)**
  Apply the algorithm to a RF cavity. (see Section 2.4.1)
- **virtual void visitRFQuadrupole (const RFQuadrupole&)**
  Apply the algorithm to a RF quadrupole. (see Section 2.4.1)
- **virtual void visitSBend (const SBend&)**
  Apply the algorithm to a sector bend. (see Section 2.4.1)
- **virtual void visitSBendWrapper (const SBendWrapper&)**
  Apply the algorithm to an SBend wrapper.. (see Section 2.37.1)
- **virtual void visitSeparator (const Separator&)**
  Apply the algorithm to a separator. (see Section 2.4.1)
- **virtual void visitSeptum (const Septum&)**
  Apply the algorithm to a septum. (see Section 2.4.1)
- **virtual void visitSolenoid (const Solenoid&)**
  Apply the algorithm to a solenoid. (see Section 2.4.1)
- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking. (see Section 2.166.1)
- **virtual ~Tracker ()**
  (see Section 2.166.1)

Protected members

- **void applyDrift (double)**
  Apply a drift length. (see Section 2.166.1)
- **void applyThinMultipole (const BMultipoleField&,double)**
  (see Section 2.166.1)
- **void applyThinSBend (const BMultipoleField&,double,double)**
  (see Section 2.166.1)
- **void applyTransform (const Euclid3D&,double)**
  Apply a geometric transformation. (see Section 2.166.1)
- **bool back_beam**
  (see Section 2.37.1)
- **bool back_track**
  (see Section 2.37.1)
- **FTps<double,2> buildMultipoleVectorPotential (const BMultipoleField&)**
  Construct vector potential for a Multipole. (see Section 2.166.1)
- **FTps<double,2> buildSBendVectorPotential (const BMultipoleField&,double)**
  Construct vector potential for a SBend. (see Section 2.166.1)
- **double flip_B**
  (see Section 2.37.1)
- **double flip_s**
  (see Section 2.37.1)
- **PartBunch itsBunch**
  The bunch of particles to be tracked. (see Section 2.166.1)
- **const Beamline& itsLine**
  (see Section 2.37.1)
- **const PartData itsReference**
  The reference information. (see Section 2.4.1)
- **bool local_flip**
  (see Section 2.37.1)

2.166.1 Detailed descriptions

Public members

- **Tracker (const Beamline&,const PartData&,bool,bool)**
  Constructor.

  The beam line to be tracked is **bl**. The particle reference data are taken from **data**. The particle bunch is initially empty. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.

- **Tracker (const Beamline&,const PartBunch&,const PartData&,bool,bool)**
  Constructor.

  The beam line to be tracked is **bl**. The particle reference data are taken from **data**. The particle bunch is taken from **bunch**. If **revBeam** is true, the beam runs from s = C to s = 0. If **revTrack** is true, we track against the beam.

- **void addToBunch (const Particle&)**
  Add particle to bunch.

- **const PartBunch& getBunch ()const**
  Return the current bunch.

- **typedef PartBunch::iterator iterator**

- **void setBunch (const PartBunch&)**
  Store the bunch.

- **virtual void visitAlignWrapper (const AlignWrapper&)**
  Apply the algorithm to an align wrapper.

- **virtual void visitComponent (const Component&)**
  Apply the algorithm to an arbitrary component.

  This override calls the component to track the bunch.

- **virtual void visitMapIntegrator (const MapIntegrator&)**
  Apply the algorithm to an integrator capable of mapping.

- **virtual void visitPatch (const Patch&)**
  Apply the algorithm to a patch.

- **virtual void visitTrackIntegrator (const TrackIntegrator&)**
  Apply the algorithm to an integrator capable of tracking.

- **virtual ~Tracker ()**

Protected members

- **void applyDrift (double)**
  Apply a drift length.

- **void applyThinMultipole (const BMultipoleField&,double)**

- **void applyThinSBend (const BMultipoleField&,double,double)**

- **void applyTransform (const Euclid3D&,double)**
  Apply a geometric transformation.

- **FTps<double,2> buildMultipoleVectorPotential (const BMultipoleField&)**
  Construct vector potential for a Multipole.

- **FTps<double,2> buildSBendVectorPotential (const BMultipoleField&,double)**
  Construct vector potential for a SBend.

- **PartBunch itsBunch**
  The bunch of particles to be tracked.

2.167 template class TransportFun <class,int>

Transport function in N variables of type T.
All divide operations throw DivideError, if the constant part of the divisor is zero. The copy constructor, destructor, and assignment operator generated by the compiler perform the correct operation. For speed reasons they are not implemented.

Type:	Instantiable
Include file:	./FixedAlgebra/TransportFun.hh

Synopsis (including inherited members)

Public members

- **TransportFun ()**
  Default constructor. (see Section 2.167.1)
- **TransportFun (const T&)**
  Conversion. (see Section 2.167.1)
- **TransportFun (int)**
  Conversion. (see Section 2.167.1)
- **T evaluate (const FVector<T,N>&)const**
  Evaluate TransportFun at point. (see Section 2.167.1)
- **std::istream& get (std::istream&)**
  Read TransportFun on the stream **is**. (see Section 2.167.1)
- **const T getCoefficient (int)const**
  Get coefficient. (see Section 2.167.1)
- **const T getCoefficient (int,int)const**
  Get coefficient. (see Section 2.167.1)
- **TransportFun inverse ()const**
  Approximate reciprocal value 1/(*this). (see Section 2.167.1)
- **static TransportFun makeVariable (int)**
  Make variable. (see Section 2.167.1)
- **TransportFun multiply (const TransportFun&)const**
  Multiplication truncated to order one. (see Section 2.167.1)
- **TransportFun<T,N> multiplyVariable (int)const**
  Multiply by variable var. (see Section 2.167.1)
- **bool operator!= (const TransportFun&)const**
  Inequality operator. (see Section 2.167.1)
- **bool operator!= (const T&)const**
  Inequality with constant. (see Section 2.167.1)
- **inline const T operator() (int,int)const**
  Get coefficient. (see Section 2.167.1)

- **inline T& operator() (int,int)**
  Get coefficient. (see Section 2.167.1)
- **TransportFun& operator*= (const TransportFun&)**
  Multiply and assign. (see Section 2.167.1)
- **TransportFun& operator*= (const T&)**
  Multiply by constant and assign. (see Section 2.167.1)
- **TransportFun operator+ ()const**
  Unary plus. (see Section 2.167.1)
- **TransportFun& operator+= (const TransportFun&)**
  Add and assign. (see Section 2.167.1)
- **TransportFun& operator+= (const T&)**
  Add constant and assign. (see Section 2.167.1)
- **TransportFun operator- ()const**
  Unary minus. (see Section 2.167.1)
- **TransportFun& operator-= (const TransportFun&)**
  Subtract and assign. (see Section 2.167.1)
- **TransportFun& operator-= (const T&)**
  Subtract constant and assign. (see Section 2.167.1)
- **TransportFun& operator/= (const TransportFun&)**
  Approximate division and assignation. (see Section 2.167.1)
- **TransportFun& operator/= (const T&)**
  Divide by constant and assign. (see Section 2.167.1)
- **TransportFun& operator= (const T&)**
  Convert and assign. (see Section 2.167.1)
- **bool operator== (const TransportFun&)const**
  Equality operator. (see Section 2.167.1)
- **bool operator== (const T&)const**
  Equality with constant. (see Section 2.167.1)
- **inline const T operator[] (int)const**
  Get coefficient. (see Section 2.167.1)
- **inline T& operator[] (int)**
  Get coefficient. (see Section 2.167.1)
- **std::ostream& put (std::ostream&)const**
  Write TransportFun on the stream **os**. (see Section 2.167.1)
- **void setCoefficient (int,const T&)**
  Set coefficient. (see Section 2.167.1)

- **void setCoefficient (int,int,const T&)**
  Set coefficient. (see Section 2.167.1)
- **TransportFun<T,N> substitute (const TransportMap<T,N>&)const**
  Substitute. (see Section 2.167.1)
- **TransportFun<T,N> substitute (const FMatrix<T,N,N>&)const**
  Substitute. (see Section 2.167.1)
- **TransportFun taylor (const T[2])const**
  Taylor series. (see Section 2.167.1)

2.167.1 Detailed descriptions

Public members

- **TransportFun ()**
  Default constructor.

  Construct zero value.
- **TransportFun (const T&)**
  Conversion.
- **TransportFun (int)**
  Conversion.
- **T evaluate (const FVector<T,N>&)const**
  Evaluate TransportFun at point.
- **std::istream& get (std::istream&)**
  Read TransportFun on the stream is.
- **const T getCoefficient (int)const**
  Get coefficient.

  Return value of the coefficient denoted by the index. The index zero stands for the constant term, any other value selects the derivative by the variable index+1. Return zero, if index is out of range.
- **const T getCoefficient (int,int)const**
  Get coefficient.

  Return value of the second-order coefficient denoted by indices. Return zero, if indices are out of range.
- **TransportFun inverse ()const**
  Approximate reciprocal value 1/(*this).
- **static TransportFun makeVariable (int)**
  Make variable.

  Construct the variable identified by the index **var**, with total of **nVar** variables.
- **TransportFun multiply (const TransportFun&)const**
  Multiplication truncated to order one.

- **TransportFun<T,N> multiplyVariable (int)const**
  Multiply by variable var.
- **bool operator!= (const TransportFun&)const**
  Inequality operator.
- **bool operator!= (const T&)const**
  Inequality with constant.
- **inline const T operator() (int,int)const**
  Get coefficient.

  Return value of the second-order coefficient denoted by indices. NOTE: i1 <= i2. Result is undefined, if indices are out of range.
- **inline T& operator() (int,int)**
  Get coefficient.

  Return a reference to the second-order coefficient denoted by indices. NOTE: i1 <= i2. Result is undefined, if indices are out of range.
- **TransportFun& operator*= (const TransportFun&)**
  Multiply and assign.
- **TransportFun& operator*= (const T&)**
  Multiply by constant and assign.
- **TransportFun operator+ ()const**
  Unary plus.
- **TransportFun& operator+= (const TransportFun&)**
  Add and assign.
- **TransportFun& operator+= (const T&)**
  Add constant and assign.
- **TransportFun operator- ()const**
  Unary minus.
- **TransportFun& operator-= (const TransportFun&)**
  Subtract and assign.
- **TransportFun& operator-= (const T&)**
  Subtract constant and assign.
- **TransportFun& operator/= (const TransportFun&)**
  Approximate division and assignation.
- **TransportFun& operator/= (const T&)**
  Divide by constant and assign.
- **TransportFun& operator= (const T&)**
  Convert and assign.

- **bool operator== (const TransportFun&)const**
  Equality operator.

- **bool operator== (const T&)const**
  Equality with constant.

- **inline const T operator[] (int)const**
  Get coefficient.

  Return value of the coefficient denoted by the index. The index zero stands for the constant term, any other value selects the derivative by the variable index+1. Result is undefined, if index is out of range.

- **inline T& operator[] (int)**
  Get coefficient.

  Return a reference to the coefficient denoted by the index. Result is undefined, if index is out of range. The index zero stands for the constant term, any other value selects the derivative by the variable index+1.

- **std::ostream& put (std::ostream&)const**
  Write TransportFun on the stream os.

- **void setCoefficient (int,const T&)**
  Set coefficient.

  Assign value of the coefficient denoted by the index. The index zero stands for the constant term, any other value selects the derivative by the variable index+1. Ignore, if index is out of range.

- **void setCoefficient (int,int,const T&)**
  Set coefficient.

  Assign value of the second-order coefficient denoted by indices. Ignore, if indices are out of range.

- **TransportFun<T,N> substitute (const TransportMap<T,N>&)const**
  Substitute.

  Substitute map **m** in TransportFun, giving new TransportFun.

- **TransportFun<T,N> substitute (const FMatrix<T,N,N>&)const**
  Substitute.

  Substitute matrix **M** in TransportFun, giving new TransportFun.

- **TransportFun taylor (const T[2])const**
  Taylor series.

  Expand Taylor series with coefficiens **series** and order one.

2.168 template class TransportMap <class,int>

Transport map with values of type T in N variables.
The copy constructor, destructor, and assignment operator generated by the compiler perform the correct operation. For speed reasons they are not implemented.

Type:	Instantiable
Include file:	./FixedAlgebra/TransportMap.hh

Synopsis (including inherited members)

Public members

- **TransportMap ()**
  Default constructor. (see Section 2.168.1)
- **TransportMap (const FVps<T,N>&)**
  Convert. (see Section 2.168.1)
- **TransportMap (const FMatrix<T,N,N>&)**
  Convert from matrix. (see Section 2.168.1)
- **TransportMap (const FVector<T,N>&)**
  Convert from vector. (see Section 2.168.1)
- **FVector<T,N> constantTerm (const FVector<T,N>&)const**
  Evaluate map at point **P**. (see Section 2.168.1)
- **FVector<T,N> constantTerm ()const**
  Evaluate map at origin. (see Section 2.168.1)
- **const TransportFun<T,N>& getComponent (int)const**
  Get component. (see Section 2.168.1)
- **void identity ()**
  Set to identity. (see Section 2.168.1)
- **TransportMap inverse ()const**
  Inverse. (see Section 2.168.1)
- **FMatrix<T,N,N> linearTerms ()const**
  Extract Transport terms at origin. (see Section 2.168.1)
- **TransportMap& operator*= (const TransportFun<T,N>&)**
  Multiply and assign. (see Section 2.168.1)
- **TransportMap& operator*= (const T&)**
  Multiply and assign. (see Section 2.168.1)
- **TransportMap operator+ ()const**
  Unary plus. (see Section 2.168.1)
- **TransportMap& operator+= (const TransportMap&)**
  Add. (see Section 2.168.1)

- **TransportMap& operator+= (const FVector<T,N>&)**
  Add and assign. (see Section 2.168.1)
- **TransportMap operator- ()const**
  Unary minus. (see Section 2.168.1)
- **TransportMap& operator-= (const TransportMap&)**
  Subtract. (see Section 2.168.1)
- **TransportMap& operator-= (const FVector<T,N>&)**
  Subtract and assign. (see Section 2.168.1)
- **TransportMap& operator/= (const TransportFun<T,N>&)**
  Divide and assign. (see Section 2.168.1)
- **TransportMap& operator/= (const T&)**
  Divide and assign. (see Section 2.168.1)
- **TransportFun<T,N>& operator[] (int)**
  Get component. (see Section 2.168.1)
- **const TransportFun<T,N>& operator[] (int)const**
  Get Component. (see Section 2.168.1)
- **void setComponent (int,const TransportFun<T,N>&)**
  Set component. (see Section 2.168.1)
- **TransportMap substitute (const FMatrix<T,N,N>&)const**
  Substitute matrix into map. (see Section 2.168.1)
- **TransportMap substitute (const TransportMap&)const**
  Substitute map into map. (see Section 2.168.1)
- **TransportMap substituteInto (const FMatrix<T,N,N>&)const**
  Substitute map into matrix. (see Section 2.168.1)

Protected members

- **static const int SIZE**
  Size of a component. (see Section 2.168.1)
- **TransportFun<T,N> data [N]**
  (see Section 2.168.1)

2.168.1 Detailed descriptions

Public members

- **TransportMap ()**
  Default constructor.

  Construct identity map.
- **TransportMap (const FVps<T,N>&)**
  Convert.

- **TransportMap (const FMatrix<T,N,N>&)**
  Convert from matrix.

  The constant part is set to zero. The Transport part is filled from **M**.

- **TransportMap (const FVector<T,N>&)**
  Convert from vector.

  The constant part is filled from **V**. The Transport part is set to the identity.

- **FVector<T,N> constantTerm (const FVector<T,N>&)const**
  Evaluate map at point P.

- **FVector<T,N> constantTerm ()const**
  Evaluate map at origin.

  / This is equivalent to extracting constant part.

- **const TransportFun<T,N>& getComponent (int)const**
  Get component.

  Return value of component **n**. Throw RangeError, if **n** is out of range.

- **void identity ()**
  Set to identity.

- **TransportMap inverse ()const**
  Inverse.

- **FMatrix<T,N,N> linearTerms ()const**
  Extract Transport terms at origin.

  This is equivalent to extracting Transport part.

- **TransportMap& operator*= (const TransportFun<T,N>&)**
  Multiply and assign.

- **TransportMap& operator*= (const T&)**
  Multiply and assign.

- **TransportMap operator+ ()const**
  Unary plus.

- **TransportMap& operator+= (const TransportMap&)**
  Add.

- **TransportMap& operator+= (const FVector<T,N>&)**
  Add and assign.

- **TransportMap operator- ()const**
  Unary minus.

- **TransportMap& operator-= (const TransportMap&)**
  Subtract.

- **TransportMap& operator-= (const FVector<T,N>&)**
  Subtract and assign.

- **TransportMap& operator/= (const TransportFun<T,N>&)**
  Divide and assign.

  Throw DivideError if constant part of **rhs** is zero.

- **TransportMap& operator/= (const T&)**
  Divide and assign.

  Throw DivideError if **rhs** is zero.

- **TransportFun<T,N>& operator[] (int)**
  Get component.

  Return reference to component **n**. Result is undefined, if index is out of range.

- **const TransportFun<T,N>& operator[] (int)const**
  Get Component.

  Return constant reference to component **n**. Result is undefined, if index is out of range.

- **void setComponent (int,const TransportFun<T,N>&)**
  Set component.

  Assign value of component **n**. Throw RangeError, if **n** is out of range.

- **TransportMap substitute (const FMatrix<T,N,N>&)const**
  Substitute matrix into map.

- **TransportMap substitute (const TransportMap&)const**
  Substitute map into map.

- **TransportMap substituteInto (const FMatrix<T,N,N>&)const**
  Substitute map into matrix.

Protected members

- **static const int SIZE**
  Size of a component.

- **TransportFun<T,N> data [N]**

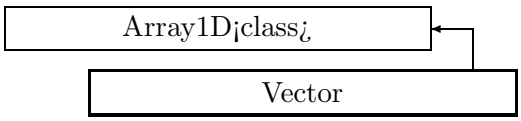

Figure 2.123: Inheritance for class Vector

2.169 template class Vector <class>

Vector.
A templated representation for vectors. This class implements the arithmetic operations.

Type:	Instantiable
Superclasses:	public Array1D<class>
Include file:	./Algebra/Vector.hh

Synopsis (including inherited members)

Public members

- **Vector ()**
  Default constructor. (see Section 2.169.1)
- **Vector (int)**
  Constructor. (see Section 2.169.1)
- **Vector (int,const T&)**
  Constructor. (see Section 2.169.1)
- **Vector (const Array1D<T>&)**
  Conversion. (see Section 2.169.1)
- **Vector (const Vector<T>&)**
  (see Section 2.169.1)
- **iterator begin ()**
  Get beginning of data. (see Section 2.7.1)
- **const_iterator begin ()const**
  Get beginning of data. (see Section 2.7.1)
- **typedef const T* const_iterator**
  The iterator type for constant array. (see Section 2.7.1)
- **iterator end ()**
  Get end of data. (see Section 2.7.1)
- **const_iterator end ()const**
  Get end of data. (see Section 2.7.1)
- **typedef T* iterator**
  The iterator type for the array. (see Section 2.7.1)

- **T& operator() (int)**
  Get reference to element. (see Section 2.7.1)
- **const T& operator() (int)const**
  Get value of element. (see Section 2.7.1)
- **Vector<T>& operator*= (const T&)**
  Multiply by scalar and assign. (see Section 2.169.1)
- **Vector<T>& operator+= (const Vector<T>&)**
  Add vector and assign. (see Section 2.169.1)
- **Vector<T> operator- ()const**
  Change sign of vector. (see Section 2.169.1)
- **Vector<T>& operator-= (const Vector<T>&)**
  Subtract vector and assign. (see Section 2.169.1)
- **Vector<T>& operator/= (const T&)**
  Divide by scalar and assign. (see Section 2.169.1)
- **Vector<T>& operator= (const Vector<T>&)**
  (see Section 2.169.1)
- **Array1D<T>& operator= (const Array1D<T>&)**
  (see Section 2.7.1)
- **T& operator[] (int)**
  Get reference to element. (see Section 2.7.1)
- **const T& operator[] (int)const**
  Get value of element. (see Section 2.7.1)
- **void resize (int)**
  Change array size. (see Section 2.7.1)
- **int size ()const**
  Get array size. (see Section 2.7.1)
- **typedef T value_type**
  The value type of this array. (see Section 2.7.1)
- **~Vector ()**
  (see Section 2.169.1)

2.169.1 Detailed descriptions

Public members

- **Vector ()**
  Default constructor.

  Constructs vector of zero length.

- **Vector (int)**
  Constructor.

  Reserve **n** members and leave them undefined.

- **Vector (int,const T&)**
  Constructor.

  Reserve **n** members and set them to **t**.

- **Vector (const Array1D<T>&)**
  Conversion.

- **Vector (const Vector<T>&)**

- **Vector<T>& operator*= (const T&)**
  Multiply by scalar and assign.

- **Vector<T>& operator+= (const Vector<T>&)**
  Add vector and assign.

- **Vector<T> operator- ()const**
  Change sign of vector.

- **Vector<T>& operator-= (const Vector<T>&)**
  Subtract vector and assign.

- **Vector<T>& operator/= (const T&)**
  Divide by scalar and assign.

- **Vector<T>& operator= (const Vector<T>&)**

- **~Vector ()**

2.170 class Vector3D

A 3-dimension vector.
The copy constructor, destructor, and assignment operator generated by the compiler perform the correct operation. For speed reasons they are not implemented.

Type:	Instantiable
Include file:	./BeamlineGeometry/Vector3D.hh

Synopsis (including inherited members)

Public members

- **Vector3D ()**
  Default constructor. (see Section 2.170.1)
- **Vector3D (double,double,double)**
  Constructor. (see Section 2.170.1)
- **void clear ()**
  Set to zero. (see Section 2.170.1)
- **void getComponents (double&,double&,double&)const**
  Get components. (see Section 2.170.1)
- **double getX ()const**
  Get component. (see Section 2.170.1)
- **double getY ()const**
  Get component. (see Section 2.170.1)
- **double getZ ()const**
  Get component. (see Section 2.170.1)
- **bool isZero ()const**
  Test for zero. (see Section 2.170.1)
- **bool operator!= (const Vector3D&)const**
  (see Section 2.170.1)
- **double& operator() (int)**
  Get component. (see Section 2.170.1)
- **double operator() (int)const**
  Get component. (see Section 2.170.1)
- **Vector3D& operator*= (double)**
  Scale and assign. (see Section 2.170.1)
- **Vector3D& operator+= (const Vector3D&)**
  Add and assign. (see Section 2.170.1)
- **Vector3D operator- ()const**
  Negative vector. (see Section 2.170.1)

- **Vector3D& operator-= (const Vector3D&)**
  Subtract and assign. (see Section 2.170.1)
- **bool operator== (const Vector3D&)const**
  (see Section 2.170.1)
- **void setX (double)**
  Set component. (see Section 2.170.1)
- **void setY (double)**
  Set component. (see Section 2.170.1)
- **void setZ (double)**
  Set component. (see Section 2.170.1)

Protected members

- **double v [3]**
  (see Section 2.170.1)

2.170.1 Detailed descriptions

Public members

- **Vector3D ()**
  Default constructor.

  Construct null vector.
- **Vector3D (double,double,double)**
  Constructor.

  Use components (x,y,z).
- **void clear ()**
  Set to zero.
- **void getComponents (double&,double&,double&)const**
  Get components.

  Return the components (x,y,z).
- **double getX ()const**
  Get component.

  Return the component **x**.
- **double getY ()const**
  Get component.

  Return the component **y**.
- **double getZ ()const**
  Get component.

  Return the component **z**.

- **bool isZero ()const**
  Test for zero.
- **bool operator!= (const Vector3D&)const**
- **double& operator() (int)**
  Get component.

  Return a reference to component **i**.
- **double operator() (int)const**
  Get component.

  Return the value of component **i**.
- **Vector3D& operator*= (double)**
  Scale and assign.
- **Vector3D& operator+= (const Vector3D&)**
  Add and assign.
- **Vector3D operator- ()const**
  Negative vector.
- **Vector3D& operator-= (const Vector3D&)**
  Subtract and assign.
- **bool operator== (const Vector3D&)const**
- **void setX (double)**
  Set component.

  Assign the component **x**.
- **void setY (double)**
  Set component.

  Assign the component **y**.
- **void setZ (double)**
  Set component.

  Assign the component **z**.

Protected members

- **double v [3]**

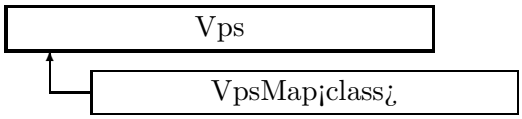

Figure 2.124: Inheritance for class Vps

2.171 template class Vps <class>

Vector truncated power series.
A vector of truncated power series with coefficients of type **T** in **n** variables. This class serves as the base class for maps.

Type:	Instantiable
Include file:	./Algebra/Vps.hh

Synopsis (including inherited members)

Public members

- **Vps (int,int)**
  Constructor. (see Section 2.171.1)
- **Vps (const Matrix<T>&)**
  Convert. (see Section 2.171.1)
- **Vps (const Vector<T>&)**
  Convert from vector. (see Section 2.171.1)
- **Vps ()**
  (see Section 2.171.1)
- **Vps (const Vps<T>&)**
  (see Section 2.171.1)
- **void check ()const**
  Check consistency. (see Section 2.171.1)
- **Vps<T> filter (int,int)const**
  Extract range of orders, set others to zero. (see Section 2.171.1)
- **std::istream& get (std::istream&)**
  Get a Vps<T> from stream is. (see Section 2.171.1)
- **const Tps<T>& getComponent (int)const**
  Get component. (see Section 2.171.1)
- **int getDimension ()const**
  Get dimension (number of Tps<T> components). (see Section 2.171.1)
- **int getTopOrder ()const**
  Get highest order contained in any component. (see Section 2.171.1)

- **int getTruncOrder ()const**
  Get lowest truncation order in any component. (see Section 2.171.1)
- **int getVariables ()const**
  Get number of variables (the same in all components). (see Section 2.171.1)
- **Vps<T>& operator*= (const Tps<T>&)**
  Multiply by Tps<T> and assign. (see Section 2.171.1)
- **Vps<T>& operator*= (const T&)**
  Multiply by constant and assign. (see Section 2.171.1)
- **Vps<T> operator+ ()const**
  Unary plus. (see Section 2.171.1)
- **Vps<T>& operator+= (const Vps<T>&)**
  Addition. (see Section 2.171.1)
- **Vps<T>& operator+= (const Vector<T>&)**
  Add and assign. (see Section 2.171.1)
- **Vps<T> operator- ()const**
  Unary minus. (see Section 2.171.1)
- **Vps<T>& operator-= (const Vps<T>&)**
  Subtraction. (see Section 2.171.1)
- **Vps<T>& operator-= (const Vector<T>&)**
  Subtract and assign. (see Section 2.171.1)
- **Vps<T>& operator/= (const Tps<T>&)**
  Divide by Tps<T> and assign. (see Section 2.171.1)
- **Vps<T>& operator/= (const T&)**
  Divide by constant and assign. (see Section 2.171.1)
- **Vps<T>& operator= (const Vps<T>&)**
  (see Section 2.171.1)
- **Tps<T>& operator[] (int)**
  Get component. (see Section 2.171.1)
- **const Tps<T>& operator[] (int)const**
  Set component. (see Section 2.171.1)
- **std::ostream& put (std::ostream&)const**
  Put a Vps<T> to stream os. (see Section 2.171.1)
- **void setComponent (int,const Tps<T>&)**
  Set component. (see Section 2.171.1)
- **Vps<T> truncate (int)**
  Truncate, may also increase truncation order. (see Section 2.171.1)
- **~Vps ()**
  (see Section 2.171.1)

Protected members

- **Array1D<Tps<T> > data**
  (see Section 2.171.1)
- **mutable int variables**
  (see Section 2.171.1)

2.171.1 Detailed descriptions

Public members

- **Vps (int,int)**
  Constructor.

  Construct **nDim** series in **nVar** variables each.
- **Vps (const Matrix<T>&)**
  Convert.

  The constant part is zero. The linear part is set to the contents of **M**.
- **Vps (const Vector<T>&)**
  Convert from vector.

  The constant part is set to the contents of **V**. The linear part is zero.
- **Vps ()**
- **Vps (const Vps<T>&)**
- **void check ()const**
  Check consistency.
- **Vps<T> filter (int,int)const**
  Extract range of orders, set others to zero.
- **std::istream& get (std::istream&)**
  Get a Vps<T> from stream is.
- **const Tps<T>& getComponent (int)const**
  Get component.

  Return the component **index**. Throw RangeError, if **index** is out of range.
- **int getDimension ()const**
  Get dimension (number of Tps<T> components).
- **int getTopOrder ()const**
  Get highest order contained in any component.
- **int getTruncOrder ()const**
  Get lowest truncation order in any component.

- **int getVariables ()const**
  Get number of variables (the same in all components).
- **Vps<T>& operator*= (const Tps<T>&)**
  Multiply by Tps<T> and assign.
- **Vps<T>& operator*= (const T&)**
  Multiply by constant and assign.
- **Vps<T> operator+ ()const**
  Unary plus.
- **Vps<T>& operator+= (const Vps<T>&)**
  Addition.
- **Vps<T>& operator+= (const Vector<T>&)**
  Add and assign.
- **Vps<T> operator- ()const**
  Unary minus.
- **Vps<T>& operator-= (const Vps<T>&)**
  Subtraction.
- **Vps<T>& operator-= (const Vector<T>&)**
  Subtract and assign.
- **Vps<T>& operator/= (const Tps<T>&)**
  Divide by Tps<T> and assign.

  Throw DivideError if constant part of **y** is zero.
- **Vps<T>& operator/= (const T&)**
  Divide by constant and assign.

  Throw DivideError if **y** is zero.
- **Vps<T>& operator= (const Vps<T>&)**
- **Tps<T>& operator[] (int)**
  Get component.

  Return the component **index**. Result is undefined, if **index** is out of range.
- **const Tps<T>& operator[] (int)const**
  Set component.

  Set the component **index**. Result is undefined, if **index** is out of range.
- **std::ostream& put (std::ostream&)const**
  Put a Vps<T> to stream os.
- **void setComponent (int,const Tps<T>&)**
  Set component.

  Set the component **index**. Throw RangeError, if **index** is out of range.

- **Vps<T> truncate (int)**
  Truncate, may also increase truncation order.
- **~Vps ()**

Protected members

- **Array1D<Tps<T> > data**
- **mutable int variables**

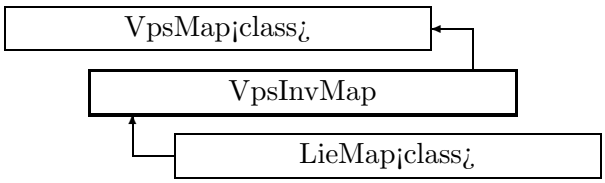

Figure 2.125: Inheritance for class VpsInvMap

2.172 template class VpsInvMap <class>

Invertible power series map
An invertible truncated power series map. The number of variables and the dimension are equal.

Type:	Instantiable
Superclasses:	public VpsMap<class>
Include file:	./Algebra/VpsInvMap.hh

Synopsis (including inherited members)

Public members

- **VpsInvMap (int)**
  Constructor. (see Section 2.172.1)
- **VpsInvMap (const VpsMap<T>&)**
  Convert. (see Section 2.172.1)
- **VpsInvMap (const Vps<T>&)**
  Convert. (see Section 2.172.1)
- **VpsInvMap (int,const Tps<T>[])**
  Constructor. (see Section 2.172.1)
- **VpsInvMap (const Matrix<T>&)**
  Convert. (see Section 2.172.1)
- **VpsInvMap (const Vector<T>&)**
  Convert. (see Section 2.172.1)
- **VpsInvMap ()**
  (see Section 2.172.1)
- **VpsInvMap (const VpsInvMap<T>&)**
  (see Section 2.172.1)
- **void check ()const**
  Check consistency. (see Section 2.172.1)
- **Vector<T> constantTerm (const Vector<T>&)const**
  Evaluate map at point. (see Section 2.173.1)
- **Vector<T> constantTerm ()const**
  Evaluate map at origin. (see Section 2.173.1)

- **VpsMap<T> derivative (int)const**
  Derivative with respect to variable **var**. (see Section 2.173.1)
- **Vps<T> filter (int,int)const**
  Extract range of orders, set others to zero. (see Section 2.171.1)
- **std::istream& get (std::istream&)**
  Get a Vps<T> from stream is. (see Section 2.171.1)
- **const Tps<T>& getComponent (int)const**
  Get component. (see Section 2.171.1)
- **int getDimension ()const**
  Get dimension (number of Tps<T> components). (see Section 2.171.1)
- **int getTopOrder ()const**
  Get highest order contained in any component. (see Section 2.171.1)
- **int getTruncOrder ()const**
  Get lowest truncation order in any component. (see Section 2.171.1)
- **int getVariables ()const**
  Get number of variables (the same in all components). (see Section 2.171.1)
- **static VpsInvMap<T> identity (int)**
  Set to identity. (see Section 2.172.1)
- **VpsMap<T> integral (int)const**
  Integral with respect to variable **var**. (see Section 2.173.1)
- **VpsInvMap<T> inverse ()const**
  Inverse. (see Section 2.172.1)
- **Matrix<T> linearTerms (const Vector<T>&)const**
  Extract linear terms at point. (see Section 2.173.1)
- **Matrix<T> linearTerms ()const**
  Extract linear terms at origin. (see Section 2.173.1)
- **Vps<T>& operator*= (const Tps<T>&)**
  Multiply by Tps<T> and assign. (see Section 2.171.1)
- **Vps<T>& operator*= (const T&)**
  Multiply by constant and assign. (see Section 2.171.1)
- **Vps<T> operator+ ()const**
  Unary plus. (see Section 2.171.1)
- **Vps<T>& operator+= (const Vps<T>&)**
  Addition. (see Section 2.171.1)
- **Vps<T>& operator+= (const Vector<T>&)**
  Add and assign. (see Section 2.171.1)

- **Vps<T> operator- ()const**
  Unary minus. (see Section 2.171.1)
- **Vps<T>& operator-= (const Vps<T>&)**
  Subtraction. (see Section 2.171.1)
- **Vps<T>& operator-= (const Vector<T>&)**
  Subtract and assign. (see Section 2.171.1)
- **Vps<T>& operator/= (const Tps<T>&)**
  Divide by Tps<T> and assign. (see Section 2.171.1)
- **Vps<T>& operator/= (const T&)**
  Divide by constant and assign. (see Section 2.171.1)
- **VpsInvMap<T>& operator= (const VpsInvMap<T>&)**
  (see Section 2.172.1)
- **VpsMap<T>& operator= (const VpsMap<T>&)**
  (see Section 2.173.1)
- **Vps<T>& operator= (const Vps<T>&)**
  (see Section 2.171.1)
- **Tps<T>& operator[] (int)**
  Get component. (see Section 2.171.1)
- **const Tps<T>& operator[] (int)const**
  Set component. (see Section 2.171.1)
- **std::ostream& put (std::ostream&)const**
  Put a Vps<T> to stream os. (see Section 2.171.1)
- **void setComponent (int,const Tps<T>&)**
  Set component. (see Section 2.171.1)
- **VpsMap<T> substitute (const VpsMap<T>&)const**
  Substitute. (see Section 2.173.1)
- **VpsMap<T> substitute (const Matrix<T>&)const**
  Substitute. (see Section 2.173.1)
- **VpsMap<T> substitute (const VpsMap<T>&,int)const**
  Substitute and truncate. (see Section 2.173.1)
- **VpsMap<T> substituteInto (const Matrix<T>&)const**
  Substitute. (see Section 2.173.1)
- **Vps<T> truncate (int)**
  Truncate, may also increase truncation order. (see Section 2.171.1)
- **~VpsInvMap ()**
  (see Section 2.172.1)

2.172.1 Detailed descriptions

Public members

- **VpsInvMap (int)**
  Constructor.

  Construct identity in **nDim** variables.

- **VpsInvMap (const VpsMap<T>&)**
  Convert.

  Throw SizeError if **rhs** is not R**n –> R**n.

- **VpsInvMap (const Vps<T>&)**
  Convert.

  Throw SizeError if **rhs** is not R**n –> R**n.

- **VpsInvMap (int,const Tps<T>[])**
  Constructor.

  Construct map of dimension **nDim** and fill from **rhs**.

- **VpsInvMap (const Matrix<T>&)**
  Convert.

  The constant terms are zero. The linear terms are taken from **M**. Throw SizeError if **M** is not square.

- **VpsInvMap (const Vector<T>&)**
  Convert.

  The constant terms are taken from **V**. The linear terms are the identity map.

- **VpsInvMap ()**

- **VpsInvMap (const VpsInvMap<T>&)**

- **void check ()const**
  Check consistency.

- **static VpsInvMap<T> identity (int)**
  Set to identity.

- **VpsInvMap<T> inverse ()const**
  Inverse.

- **VpsInvMap<T>& operator= (const VpsInvMap<T>&)**

- **~VpsInvMap ()**

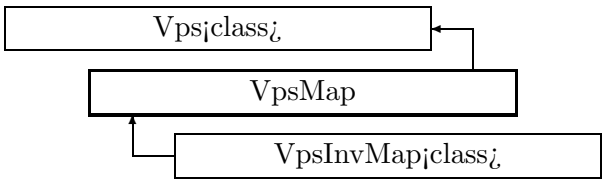

Figure 2.126: Inheritance for class VpsMap

2.173 template class VpsMap <class>

Truncate power series map.
This class includes substitution operations. The input and output dimensions need not be the same.

Type:	Instantiable
Superclasses:	public Vps<class>
Include file:	./Algebra/VpsMap.hh

Synopsis (including inherited members)

Public members

- **VpsMap (int,int)**
  Constructor. (see Section 2.173.1)
- **VpsMap (const Vps<T>&)**
  Convert. (see Section 2.173.1)
- **VpsMap (int,const Tps<T>[])**
  Constructor. (see Section 2.173.1)
- **VpsMap (const Matrix<T>&)**
  Convert. (see Section 2.173.1)
- **VpsMap (const Vector<T>&)**
  Convert. (see Section 2.173.1)
- **VpsMap ()**
  (see Section 2.173.1)
- **VpsMap (const VpsMap<T>&)**
  (see Section 2.173.1)
- **void check ()const**
  Check consistency. (see Section 2.171.1)
- **Vector<T> constantTerm (const Vector<T>&)const**
  Evaluate map at point. (see Section 2.173.1)
- **Vector<T> constantTerm ()const**
  Evaluate map at origin. (see Section 2.173.1)

- **VpsMap<T> derivative (int)const**
  Derivative with respect to variable **var**. (see Section 2.173.1)
- **Vps<T> filter (int,int)const**
  Extract range of orders, set others to zero. (see Section 2.171.1)
- **std::istream& get (std::istream&)**
  Get a Vps<T> from stream is. (see Section 2.171.1)
- **const Tps<T>& getComponent (int)const**
  Get component. (see Section 2.171.1)
- **int getDimension ()const**
  Get dimension (number of Tps<T> components). (see Section 2.171.1)
- **int getTopOrder ()const**
  Get highest order contained in any component. (see Section 2.171.1)
- **int getTruncOrder ()const**
  Get lowest truncation order in any component. (see Section 2.171.1)
- **int getVariables ()const**
  Get number of variables (the same in all components). (see Section 2.171.1)
- **VpsMap<T> integral (int)const**
  Integral with respect to variable **var**. (see Section 2.173.1)
- **Matrix<T> linearTerms (const Vector<T>&)const**
  Extract linear terms at point. (see Section 2.173.1)
- **Matrix<T> linearTerms ()const**
  Extract linear terms at origin. (see Section 2.173.1)
- **Vps<T>& operator*= (const Tps<T>&)**
  Multiply by Tps<T> and assign. (see Section 2.171.1)
- **Vps<T>& operator*= (const T&)**
  Multiply by constant and assign. (see Section 2.171.1)
- **Vps<T> operator+ ()const**
  Unary plus. (see Section 2.171.1)
- **Vps<T>& operator+= (const Vps<T>&)**
  Addition. (see Section 2.171.1)
- **Vps<T>& operator+= (const Vector<T>&)**
  Add and assign. (see Section 2.171.1)
- **Vps<T> operator- ()const**
  Unary minus. (see Section 2.171.1)
- **Vps<T>& operator-= (const Vps<T>&)**
  Subtraction. (see Section 2.171.1)

- **Vps<T>& operator-= (const Vector<T>&)**
  Subtract and assign. (see Section 2.171.1)
- **Vps<T>& operator/= (const Tps<T>&)**
  Divide by Tps<T> and assign. (see Section 2.171.1)
- **Vps<T>& operator/= (const T&)**
  Divide by constant and assign. (see Section 2.171.1)
- **VpsMap<T>& operator= (const VpsMap<T>&)**
  (see Section 2.173.1)
- **Vps<T>& operator= (const Vps<T>&)**
  (see Section 2.171.1)
- **Tps<T>& operator[] (int)**
  Get component. (see Section 2.171.1)
- **const Tps<T>& operator[] (int)const**
  Set component. (see Section 2.171.1)
- **std::ostream& put (std::ostream&)const**
  Put a Vps<T> to stream os. (see Section 2.171.1)
- **void setComponent (int,const Tps<T>&)**
  Set component. (see Section 2.171.1)
- **VpsMap<T> substitute (const VpsMap<T>&)const**
  Substitute. (see Section 2.173.1)
- **VpsMap<T> substitute (const Matrix<T>&)const**
  Substitute. (see Section 2.173.1)
- **VpsMap<T> substitute (const VpsMap<T>&,int)const**
  Substitute and truncate. (see Section 2.173.1)
- **VpsMap<T> substituteInto (const Matrix<T>&)const**
  Substitute. (see Section 2.173.1)
- **Vps<T> truncate (int)**
  Truncate, may also increase truncation order. (see Section 2.171.1)
- **~VpsMap ()**
  (see Section 2.173.1)

2.173.1 Detailed descriptions

Public members

- **VpsMap (int,int)**
  Constructor.

  Construct **nDim** series in **nVar** variables each.

- **VpsMap (const Vps<T>&)**
  Convert.

- **VpsMap (int,const Tps<T>[])**
  Constructor.

  Assign the **nDim** components from **rhs**.

- **VpsMap (const Matrix<T>&)**
  Convert.

  The constant terms are zero. The linear terms are taken from **M**.

- **VpsMap (const Vector<T>&)**
  Convert.

  The constant terms are taken from **V**. The linear terms are zero.

- **VpsMap ()**

- **VpsMap (const VpsMap<T>&)**

- **Vector<T> constantTerm (const Vector<T>&)const**
  Evaluate map at point.

  Return the point **y** mapped by the map.

- **Vector<T> constantTerm ()const**
  Evaluate map at origin.

  Extract constant part.

- **VpsMap<T> derivative (int)const**
  Derivative with respect to variable var.

- **VpsMap<T> integral (int)const**
  Integral with respect to variable var.

- **Matrix<T> linearTerms (const Vector<T>&)const**
  Extract linear terms at point.

  Return the linear terms of the map around the point **y**, i.e. the linear part of the partial differentials at **y**.

- **Matrix<T> linearTerms ()const**
  Extract linear terms at origin.

  Return the linear part.

- **VpsMap<T>& operator= (const VpsMap<T>&)**

- **VpsMap<T> substitute (const VpsMap<T>&)const**
  Substitute.

- **VpsMap<T> substitute (const Matrix<T>&)const Substitute.**
- **VpsMap<T> substitute (const VpsMap<T>&,int)const Substitute and truncate.**
- **VpsMap<T> substituteInto (const Matrix<T>&)const Substitute.**
- **~VpsMap ()**

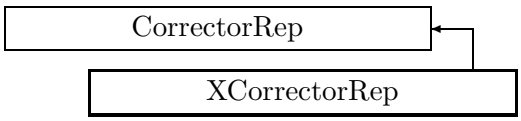

Figure 2.127: Inheritance for class XCorrectorRep

2.174 class XCorrectorRep

Representation for an orbit corrector.
This derived class acts on the horizontal plane.

Type:	Instantiable
Superclasses:	public CorrectorRep
Include file:	./BeamlineCore/XCorrectorRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **enum Plane**
  Plane selection. (see Section 2.34.1)
- **XCorrectorRep (const string&)**
  Constructor with given name. (see Section 2.174.1)
- **XCorrectorRep ()**
  (see Section 2.174.1)
- **XCorrectorRep (const XCorrectorRep&)**
  (see Section 2.174.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply a visitor to Corrector. (see Section 2.34.1)

- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.174.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getBx ()const**
  Get field. (see Section 2.174.1)
- **virtual double getBy ()const**
  Get vertical field component in Teslas. (see Section 2.35.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.174.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BDipoleField& getField ()**
  Get corrector field. (see Section 2.35.1)

- **virtual const BDipoleField& getField ()const**
  Get corrector field. Version for const corrector. (see Section 2.35.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.35.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.35.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.174.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Plane getPlane ()const**
  Get plane of action. (see Section 2.174.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.174.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.35.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)

- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setActive (bool)**
  Set active flag. (see Section 2.35.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setBx (double)**
  Set field. (see Section 2.174.1)
- **virtual void setBy (double)**
  Set vertical field component in Teslas. (see Section 2.35.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~XCorrectorRep ()**
  (see Section 2.174.1)

2.174.1 Detailed descriptions

Public members

- **XCorrectorRep (const string&)**
  Constructor with given name.
- **XCorrectorRep ()**
- **XCorrectorRep (const XCorrectorRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual double getBx ()const**
  Get field.

  Return horizontal component (always zero).
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.
- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.
- **virtual Plane getPlane ()const**
  Get plane of action.

  Return the x-plane for this class.
- **virtual const string& getType ()const**
  Get element type string.
- **virtual void setBx (double)**
  Set field.

  Ignore the horizontal field value.
- **virtual ~XCorrectorRep ()**

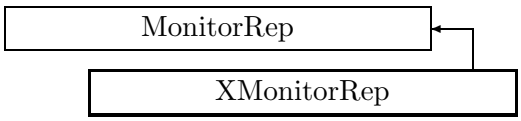

Figure 2.128: Inheritance for class XMonitorRep

2.175 class XMonitorRep

Representation for a orbit position monitor.
Acts on the horizontal plane.

Type:	Instantiable
Superclasses:	public MonitorRep
Include file:	./BeamlineCore/XMonitorRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **enum Plane**
  Plane selection. (see Section 2.98.1)
- **XMonitorRep (const string&)**
  Constructor with given name. (see Section 2.175.1)
- **XMonitorRep ()**
  (see Section 2.175.1)
- **XMonitorRep (const XMonitorRep&)**
  (see Section 2.175.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Monitor. (see Section 2.98.1)

- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.175.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.175.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual NullField& getField ()**
  Get field. (see Section 2.99.1)
- **virtual const NullField& getField ()const**
  Get field. (see Section 2.99.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.99.1)

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.99.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.175.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Plane getPlane ()const**
  Get plane. (see Section 2.175.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.175.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)

- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setActive (bool)**
  Set active flag. (see Section 2.99.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~XMonitorRep ()**
  (see Section 2.175.1)

2.175.1 Detailed descriptions

Public members

- **XMonitorRep (const string&)**
  Constructor with given name.
- **XMonitorRep ()**
- **XMonitorRep (const XMonitorRep&)**

- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.

- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual Plane getPlane ()const**
  Get plane.

  Return the x-plane for this class.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual ~XMonitorRep ()**

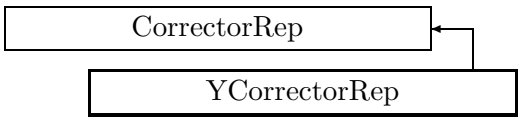

Figure 2.129: Inheritance for class YCorrectorRep

2.176 class YCorrectorRep

Representation for an orbit corrector.
Acts on the vertical plane.

Type:	Instantiable
Superclasses:	public CorrectorRep
Include file:	./BeamlineCore/YCorrectorRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **enum Plane**
  Plane selection. (see Section 2.34.1)
- **YCorrectorRep (const string&)**
  Constructor with given name. (see Section 2.176.1)
- **YCorrectorRep ()**
  (see Section 2.176.1)
- **YCorrectorRep (const YCorrectorRep&)**
  (see Section 2.176.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply a visitor to Corrector. (see Section 2.34.1)

- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.176.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual double getBx ()const**
  Get horizontal field component in Teslas. (see Section 2.35.1)
- **virtual double getBy ()const**
  Get field. (see Section 2.176.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.176.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual BDipoleField& getField ()**
  Get corrector field. (see Section 2.35.1)

- **virtual const BDipoleField& getField ()const**
  Get corrector field. Version for const corrector. (see Section 2.35.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.35.1)
- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.35.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.176.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Plane getPlane ()const**
  Get plane. (see Section 2.176.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.176.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.35.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)

- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)
- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setActive (bool)**
  Set active flag. (see Section 2.35.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setBx (double)**
  Set horizontal field component in Teslas. (see Section 2.35.1)
- **virtual void setBy (double)**
  Set field. (see Section 2.176.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~YCorrectorRep ()**
  (see Section 2.176.1)

2.176.1 Detailed descriptions

Public members

- **YCorrectorRep (const string&)**
  Constructor with given name.
- **YCorrectorRep ()**
- **YCorrectorRep (const YCorrectorRep&)**
- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.
- **virtual double getBy ()const**
  Get field.

  Return the vertical field component (always zero).
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.
- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.
- **virtual Plane getPlane ()const**
  Get plane.

  Return the y-plane for this class.
- **virtual const string& getType ()const**
  Get element type string.
- **virtual void setBy (double)**
  Set field.

  Ignore the vertical field component.
- **virtual ~YCorrectorRep ()**

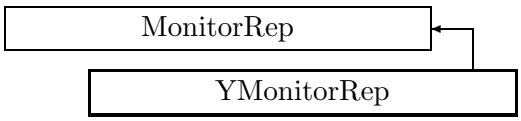

Figure 2.130: Inheritance for class YMonitorRep

2.177 class YMonitorRep

Representation for a orbit position monitor.
Acts on the vertical plane.

Type:	Instantiable
Superclasses:	public MonitorRep
Include file:	./BeamlineCore/YMonitorRep.hh

Synopsis (including inherited members)

Public members

- **BVector Bfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **BVector Bfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EBVectors EBfield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&)const**
  Return the field in a point. (see Section 2.26.1)
- **EVector Efield (const Point3D&,double)const**
  Return the field in a point. (see Section 2.26.1)
- **enum Plane**
  Plane selection. (see Section 2.98.1)
- **YMonitorRep (const string&)**
  Constructor with given name. (see Section 2.177.1)
- **YMonitorRep ()**
  (see Section 2.177.1)
- **YMonitorRep (const YMonitorRep&)**
  (see Section 2.177.1)
- **virtual void accept (BeamlineVisitor&)const**
  Apply visitor to Monitor. (see Section 2.98.1)

- **int addReference ()const**
  Increment reference count. (see Section 2.126.1)
- **virtual ElementBase* clone ()const**
  Return clone. (see Section 2.177.1)
- **virtual ElementBase* copyStructure ()**
  Make a structural copy. (see Section 2.52.1)
- **virtual double getArcLength ()const**
  Get arc length. (see Section 2.52.1)
- **virtual double getAttribute (const string&)const**
  Get attribute value. (see Section 2.52.1)
- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel. (see Section 2.177.1)
- **virtual const ConstChannel* getConstChannel (const string&)const**
  Construct a read-only channel. (see Section 2.52.1)
- **virtual const ElementBase& getDesign ()const**
  Return design element. (see Section 2.26.1)
- **virtual double getElementLength ()const**
  Get design length. (see Section 2.52.1)
- **virtual double getEntrance ()const**
  Get entrance position. (see Section 2.52.1)
- **virtual Euclid3D getEntranceFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getEntrancePatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual double getExit ()const**
  Get exit position. (see Section 2.52.1)
- **virtual Euclid3D getExitFrame ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getExitPatch ()const**
  Get patch. (see Section 2.52.1)
- **virtual NullField& getField ()**
  Get field. (see Section 2.99.1)
- **virtual const NullField& getField ()const**
  Get field. (see Section 2.99.1)
- **virtual StraightGeometry& getGeometry ()**
  Get geometry. (see Section 2.99.1)

- **virtual const StraightGeometry& getGeometry ()const**
  Get geometry. (see Section 2.99.1)
- **virtual ElementImage* getImage ()const**
  Construct an image. (see Section 2.177.1)
- **virtual const string& getName ()const**
  Get element name. (see Section 2.52.1)
- **virtual double getOrigin ()const**
  Get origin position. (see Section 2.52.1)
- **virtual Plane getPlane ()const**
  Get plane. (see Section 2.177.1)
- **virtual Euclid3D getTotalTransform ()const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double,double)const**
  Get transform. (see Section 2.52.1)
- **virtual Euclid3D getTransform (double)const**
  Get transform. (see Section 2.52.1)
- **virtual const string& getType ()const**
  Get element type string. (see Section 2.177.1)
- **virtual bool hasAttribute (const string&)const**
  Test for existence of an attribute. (see Section 2.52.1)
- **bool isSharable ()const**
  Test if the element can be shared. (see Section 2.52.1)
- **bool isShared ()const**
  Test for sharing. (see Section 2.126.1)
- **virtual ElementBase* makeAlignWrapper ()**
  Allow misalignment. (see Section 2.52.1)
- **virtual ElementBase* makeFieldWrapper ()**
  Allow field errors. (see Section 2.52.1)
- **virtual void makeSharable ()**
  Set sharable flag. (see Section 2.52.1)
- **virtual ElementBase* makeWrappers ()**
  Allow errors. (see Section 2.52.1)
- **virtual ElementBase* removeAlignWrapper ()**
  Remove align wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeAlignWrapper ()const**
  Remove align wrapper. (see Section 2.52.1)

- **virtual void removeAttribute (const string&)**
  Remove an existing attribute. (see Section 2.52.1)
- **virtual ElementBase* removeFieldWrapper ()**
  Remove field wrapper. (see Section 2.52.1)
- **virtual const ElementBase* removeFieldWrapper ()const**
  Remove field wrapper. (see Section 2.52.1)
- **int removeReference ()const**
  Decrement the reference count. (see Section 2.126.1)
- **virtual ElementBase* removeWrappers ()**
  Return the design element. (see Section 2.52.1)
- **virtual const ElementBase* removeWrappers ()const**
  Return the design element. (see Section 2.52.1)
- **virtual void setActive (bool)**
  Set active flag. (see Section 2.99.1)
- **virtual void setAttribute (const string&,double)**
  Set value of an attribute. (see Section 2.52.1)
- **virtual void setElementLength (double)**
  Set design length. (see Section 2.52.1)
- **virtual void setName (const string&)**
  Set element name. (see Section 2.52.1)
- **virtual void trackBunch (PartBunch&,const PartData&,bool,bool)const**
  Track particle bunch. (see Section 2.26.1)
- **virtual void trackMap (FVps<double,6>&,const PartData&,bool,bool)const**
  Track a map. (see Section 2.26.1)
- **bool update (const AttributeSet&)**
  Update element. (see Section 2.52.1)
- **virtual ~YMonitorRep ()**
  (see Section 2.177.1)

2.177.1 Detailed descriptions

Public members

- **YMonitorRep (const string&)**
  Constructor with given name.
- **YMonitorRep ()**
- **YMonitorRep (const YMonitorRep&)**

- **virtual ElementBase* clone ()const**
  Return clone.

  Return an identical deep copy of the element.

- **virtual Channel* getChannel (const string&)**
  Construct a read/write channel.

  This method constructs a Channel permitting read/write access to the attribute **aKey** and returns it. If the attribute does not exist, it returns NULL.

- **virtual ElementImage* getImage ()const**
  Construct an image.

  Return the image of the element, containing the name and type string of the element, and a copy of the user-defined attributes.

- **virtual Plane getPlane ()const**
  Get plane.

  Return y-plane for this class.

- **virtual const string& getType ()const**
  Get element type string.

- **virtual ~YMonitorRep ()**

2.178 template class complex <class>

Type:	Instantiable

Bibliography

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.